Documentazione Word Automata
Ingegneria del software

Zeno

Dimitri

A.A. 2023/2024

Indice

1 Requisiti e interazione con l'utente

1.1 Casi d'uso

Il programma non prevede distinzione tra utenti (come specificato dal testo dell'elaborato), di conseguenza i diagrammi riportano "User" come utilizzatore generale del software. Di seguito il diagramma dei casi d'uso con le opportune specifiche.

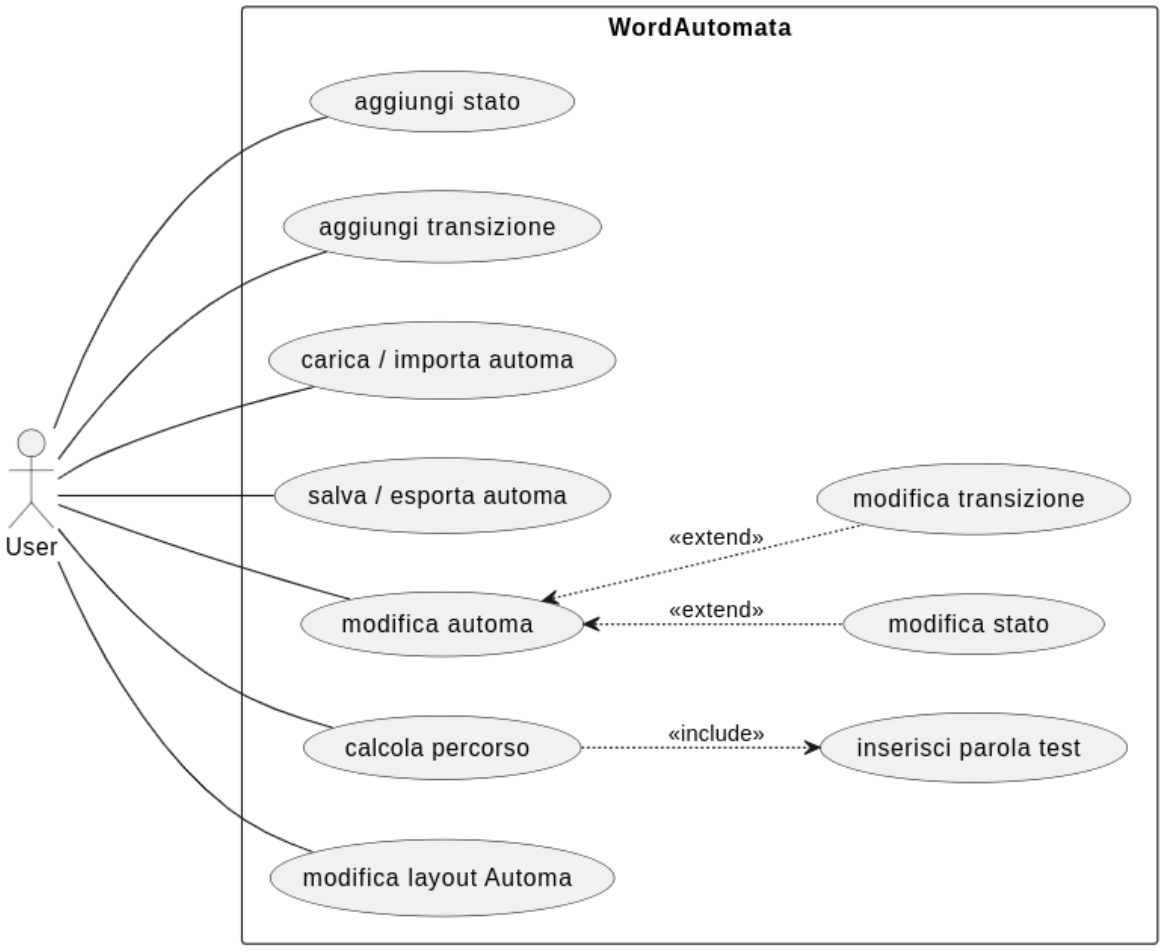


Figura 1.1: Diagramma dei casi d'uso

1.2 Specifiche dei casi d'uso

Più avanti nella documentazione vengono riportate descrizioni più approfondite delle diverse funzionalità, in questa sezione sono invece presenti i diagrammi delle attività dei casi d'uso più importanti.

Caso d'uso	Aggiungi stato
Attore	User
Descrizione	L'utente aggiunge uno stato all'automa
Precondizioni	Il programma è in esecuzione
Passi	1. L'utente esegue un doppio click sul pannello dell'automa. 2. L'utente visualizza il popup di creazione dello stato. 3. L'utente inserisce le informazioni desiderate per lo stato da creare. 4. Se l'utente inserisce un nome già appartenente ad un altro stato oppure un nome vuoto, il sistema restituisce un messaggio di errore specifico, invitando l'utente a riprovare.
Postcondizioni	È stato aggiunto uno stato all'automa

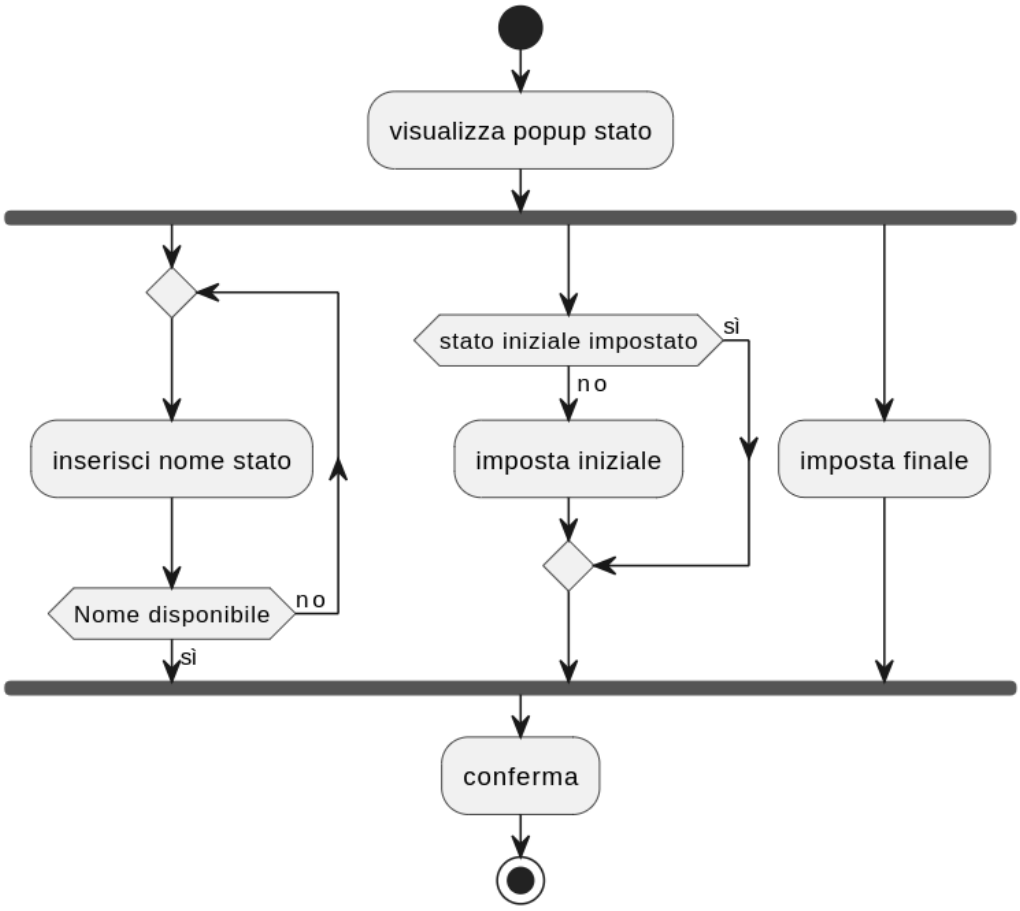


Figura 1.2: Diagramma delle attività del caso d'uso **Aggiungi stato**

Caso d'uso	Aggiungi transizione
Attore	User
Descrizione	L'utente aggiunge una transizione all'automa
Precondizioni	È stato creato almeno uno stato
Passi	1. L'utente seleziona uno stato 2. L'utente tenendo premuto CTRL fa click su uno stato (uno qualsiasi, anche lo stato già selezionato). 3. L'utente visualizza il popup di creazione della transizione. 4. L'utente inserisce il nome desiderato per la transizione da creare. 5. Se l'utente inserisce un nome già appartenente ad un'altra transizione con lo stesso stato di partenza, il sistema restituisce un messaggio di errore specifico, invitando l'utente a riprovare.
Postcondizioni	È stata aggiunta una transizione all'automa

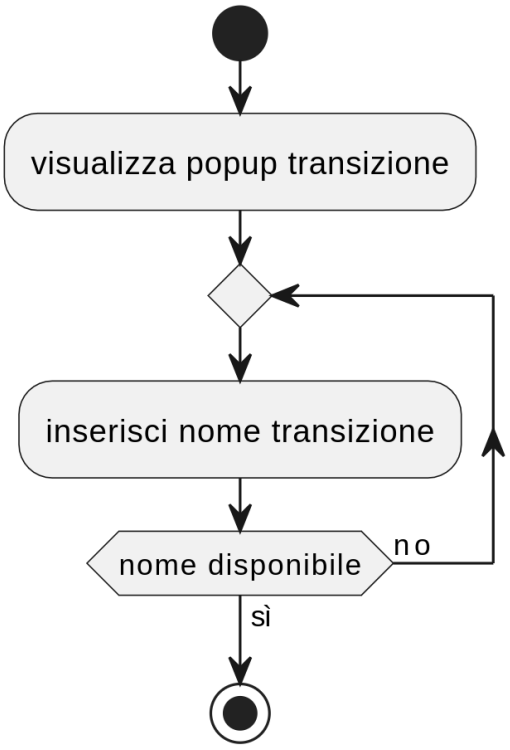


Figura 1.3: Diagramma delle attività del caso d'uso **Aggiungi transizione**

Caso d'uso	Carica / importa automa
Attore	User
Descrizione	L'utente carica un automa da file
Precondizioni	Il programma è in esecuzione
Passi	1. L'utente seleziona l'opzione carica automa. 2. L'utente visualizza la finestra del file manager del proprio sistema operativo. 3. L'utente seleziona un file. 4. Se l'utente seleziona un file non compatibile, il sistema restituisce un messaggio di errore specifico.
Postcondizioni	È stato caricato un automa

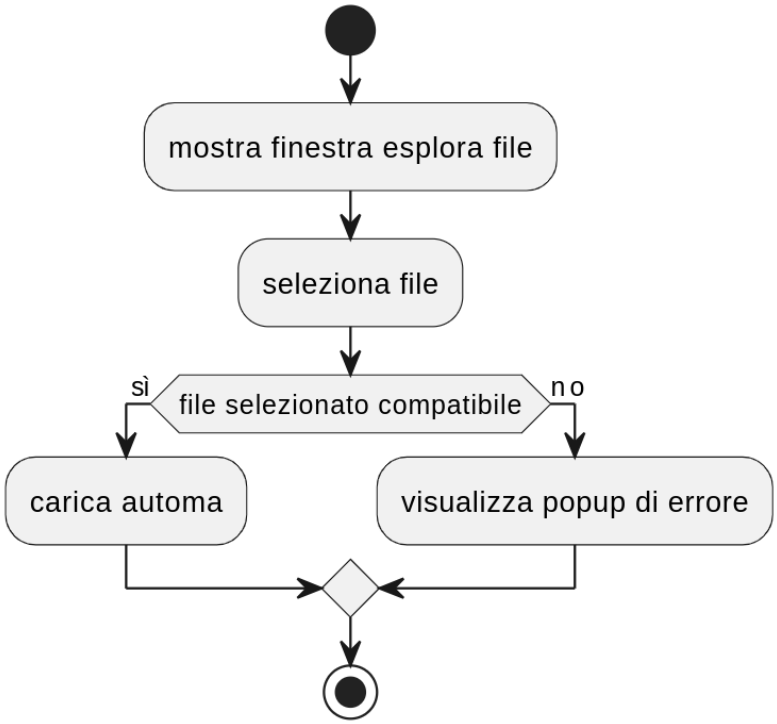


Figura 1.4: Diagramma delle attività del caso d'uso **Carica / importa automa**

Caso d'uso	Salva / esporta automa
Attore	User
Descrizione	L'utente salva un automa creato
Precondizioni	Il programma è in esecuzione
Passi	1. L'utente seleziona l'opzione per salvare un automata. 2. L'utente visualizza la finestra del file manager del proprio sistema operativo. 3. L'utente inserisce il nome con cui desidera salvare l'automa. 4. Se esiste già un file identico, viene chiesto all'utente se vuole sovrascriverlo. 5. Se l'utente decide di sovrascrivere il file questo viene salvato, altrimenti viene richiesto di scegliere un altro nome.
Postcondizioni	È stato salvato un automa su file

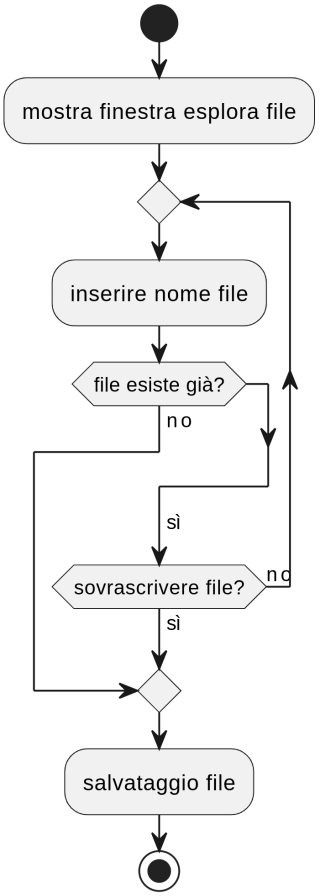


Figura 1.5: Diagramma delle attività del caso d'uso **Salva / esporta automa**

Caso d'uso	Modifica automa
Attore	User
Descrizione	L'utente modifica l'automa
Precondizioni	L'automa corrente contiene elementi modificabili (stati e transizioni)
Passi	1. L'utente apre il pannello di modifica dell'automa. 2. L'utente seleziona un elemento modificabile. 3. a. L'utente inserisce un nuovo nome nel campo relativo all'elemento selezionato. b. L'utente preme il pulsante elimina relativo all'elemento selezionato 4. a. Se il nome inserito è disponibile, l'elemento viene modificato altrimenti viene mostrato un popup di errore. b. Il nodo selezionato viene eliminato.
Postcondizioni	L'automa è stato modificato

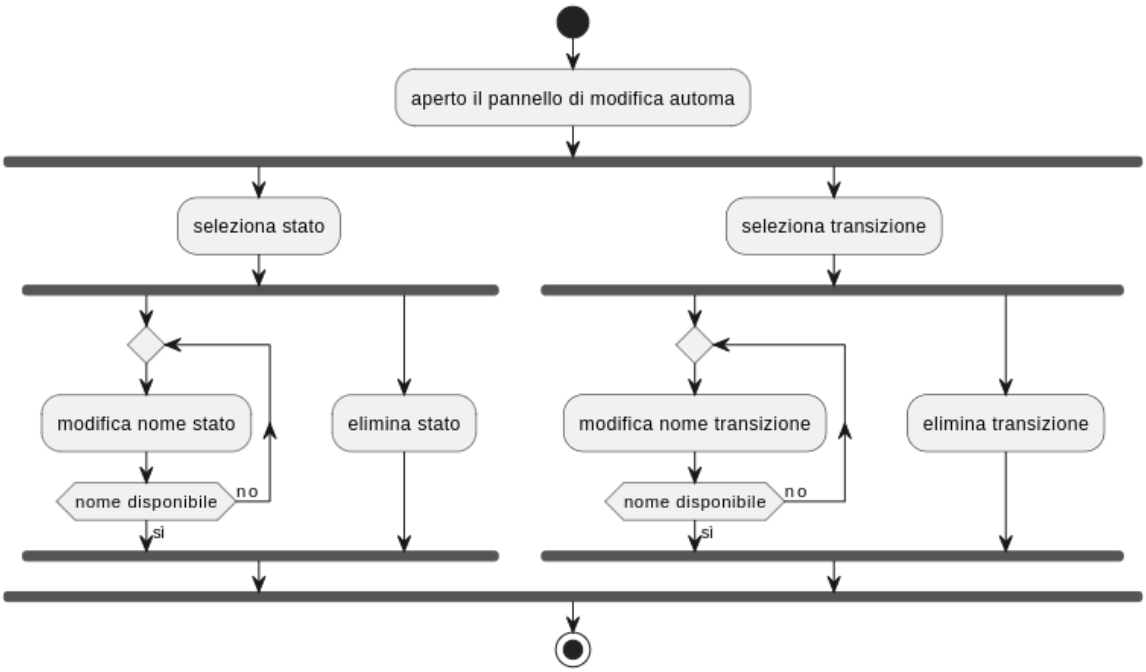


Figura 1.6: Diagramma delle attività del caso d'uso **Modifica automa**

nota: se uno stato viene eliminato non può più essere modificato, vale invece il contrario.

Caso d'uso	Calcola percorso
Attore	User
Descrizione	L'utente inserisce un percorso, il programma cerca di trovare questo percorso.
Precondizioni	Esiste uno stato iniziale e uno o più finali
Passi	1. L'utente apre il pannello di modifica dell'automa. 2. L'utente inserisce un percorso nell'apposita casella di testo. 3. a. Il percorso è stato trovato e viene mostrato all'utente. b. Il percorso non è stato trovato e si avvisa l'utente.
Postcondizioni	È stato calcolato e mostrato un percorso

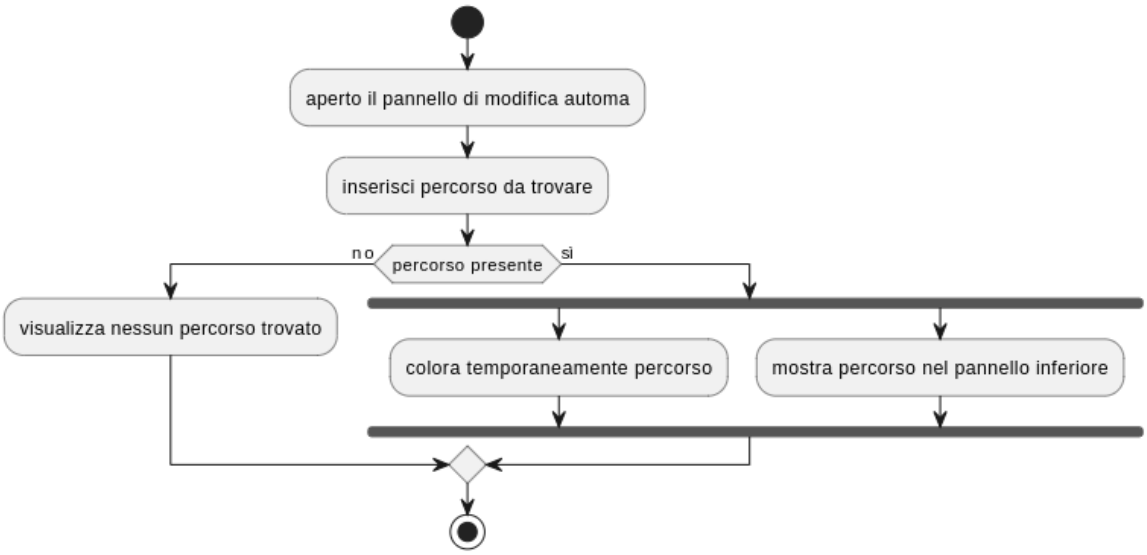


Figura 1.7: Diagramma delle attività del caso d'uso **Calcola percorso**

2 Sviluppo

2.1 Sviluppo generale

Il progetto è stato sviluppato combinando i modelli di approccio allo sviluppo software, incrementale e agile. Le diverse funzionalità sono state definite in fase di analisi dei requisiti e non sono più state modificate, a differenza della loro implementazione che durante lo sviluppo ha subito diversi cambiamenti. Seguendo questi modelli ed alcune caratteristiche dell'extreme programming abbiamo proseguito sviluppando inizialmente un design semplice con le funzionalità di base, seguito da piccole e frequenti release contenenti nuove funzionalità, bug-fix e cambiamenti all'interfaccia utente. Sono stati eseguiti ad ogni release dei test su tutto il programma per verificare la corretta implementazione delle nuove funzionalità e controllare che quelle precedentemente implementate non fossero state compromesse.

Durante lo sviluppo sono state ampiamente utilizzate tecniche come il pair programming, il refactoring, il planning delle task incrementale, fondamentale per procedere nello sviluppo del software in modo ordinato.
Si è infatti cominciato dall'implementazione di feature semplici e veloci da realizzare, in modo da ottenere una base del software funzionante. Una volta raggiunta una solida base su cui poter lavorare, sono stati divisi i compiti in modo da poter procedere simultaneamente allo sviluppo di componenti diverse, nonostante questo ci si è sempre tenuti aggiornati sull'intero sviluppo del software con frequenti sessioni di revisione comune del programma in modo da favorire un'ampia conoscenza del codice e una minore possibilità di errore.

Per garantire una produzione parallela del codice è stato utilizzato il sistema di versioning distribuito Git e la sua implementazione online git-hub. Il progetto è stato scritto in java-21 utilizzando JavaFX per l'interfaccia grafica insieme a librerie supplementari, atte alla realizzazione di alcune componenti specifiche (es: jackson per il salvataggio degli automi e AtlantaFX per uno stile di partenza).

La documentazione è stata realizzata una volta finito il progetto per concentrare le forze sulla scrittura del codice durante la fase di sviluppo, questo ha anche permesso di realizzare una singola volta i diagrammi UML proposti, i quali sono stati realizzati in parte automaticamente tramite l'utilizzo dell'estensione PlantUML disponibile per il code editor Visual Studio Code.

2.2 Pattern Architetturali e di design

La scelta dei pattern architetturali e di design è stata pesantemente influenzata dall'utilizzo della libreria grafica JavaFX e della libreria SmartGraph per la visualizzazione di grafi, la quale è stata pesantemente modificata per adattarsi alla struttura del nostro programma poichè di natura open source con licenza MIT. Di seguito la spiegazione dei pattern utilizzati e di riferimento:

Model-View-Controller l'architettura generale del programma ha visto implementato il pattern architetturale Model-View-Controller per mantenere il più possibile distinte e ordinate le varie componenti del programma. Il componente view è stato implementato tramite la libreria javafx e i componenti fxml, i quali compongono l'interfaccia grafica con cui l'utente interagisce. Il componente Controller invece si occupa di gestire le interazioni tra l'interfaccia e la parte di backend del software. Infine, la componente Model si occupa di gestire i dati e le operazioni su di essi, permette quindi il passaggio delle informazioni tra varie componenti controller. La natura delle interazioni tra queste commponenti verrà chiarita dagli schemi successivi.

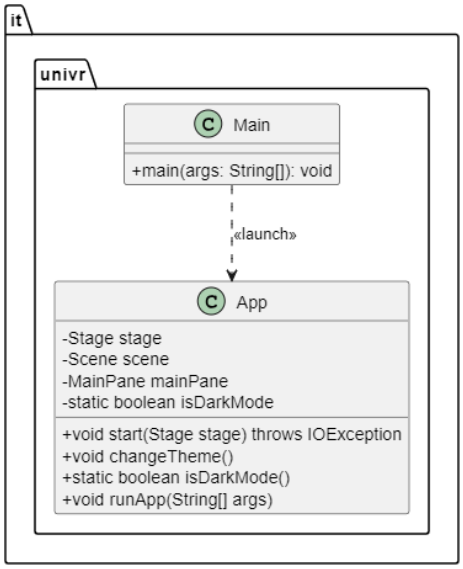
Singleton la classe SceneReference è stata strutturata utilizzando i concetti del pattern creazionale singleton, è stato deciso però di utilizzare tutte e sole variabili e metodi statici e di rendere il costrut-

tore privato anziché implementare un vero e proprio singleton, per un più semplice, chiaro e veloce utilizzo. Lo scopo però rimane lo stesso poiché disponendo di soli metodi e variabili statiche non è stato necessario creare alcuna istanza all'interno dell'intero programma.

Facade sono state utilizzate delle caratteristiche del pattern strutturale Facade, per permettere l'esecuzione di operazioni sui dati nascondendone la complessità; non sono state istanziate classi puramente Facade bensì sono presenti queste caratteristiche all'interno di vari componenti del Model.

2.3 Diagrammi delle classi

Nelle seguenti pagine riportiamo i diagrammi delle diverse classi che compongono il progetto.



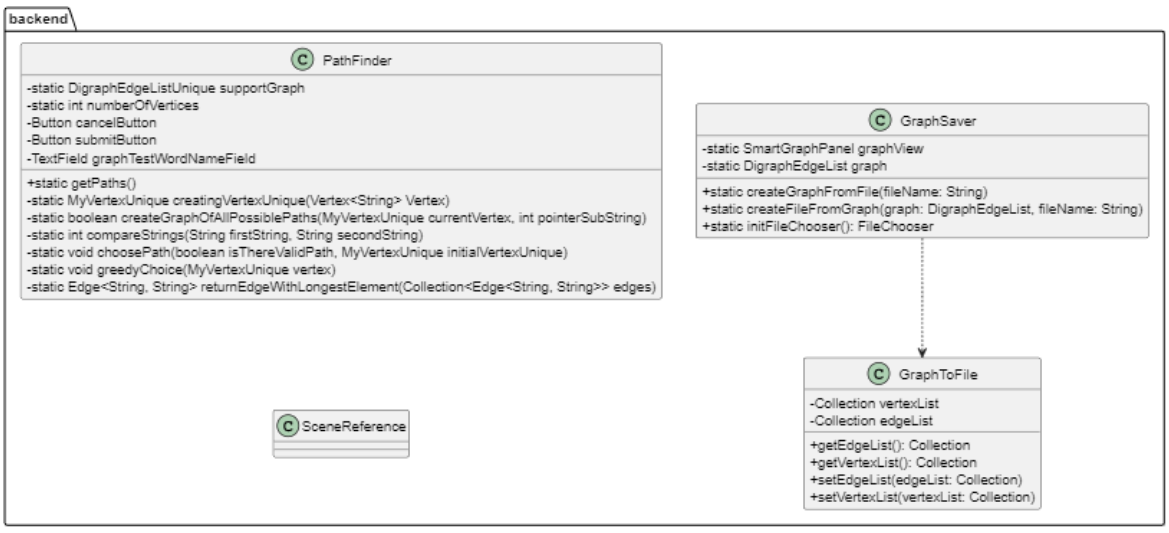


Figura 2.1: Diagramma delle classi del backend (SceneReference viene solo menzionato poichè contiene troppi metodi per essere visualizzato in una pagina sola)

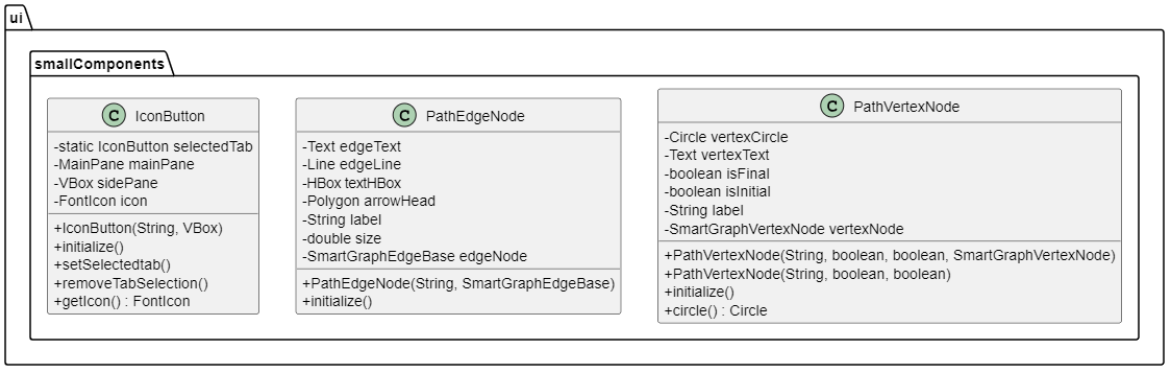


Figura 2.2: Diagramma delle classi dei piccoli componenti

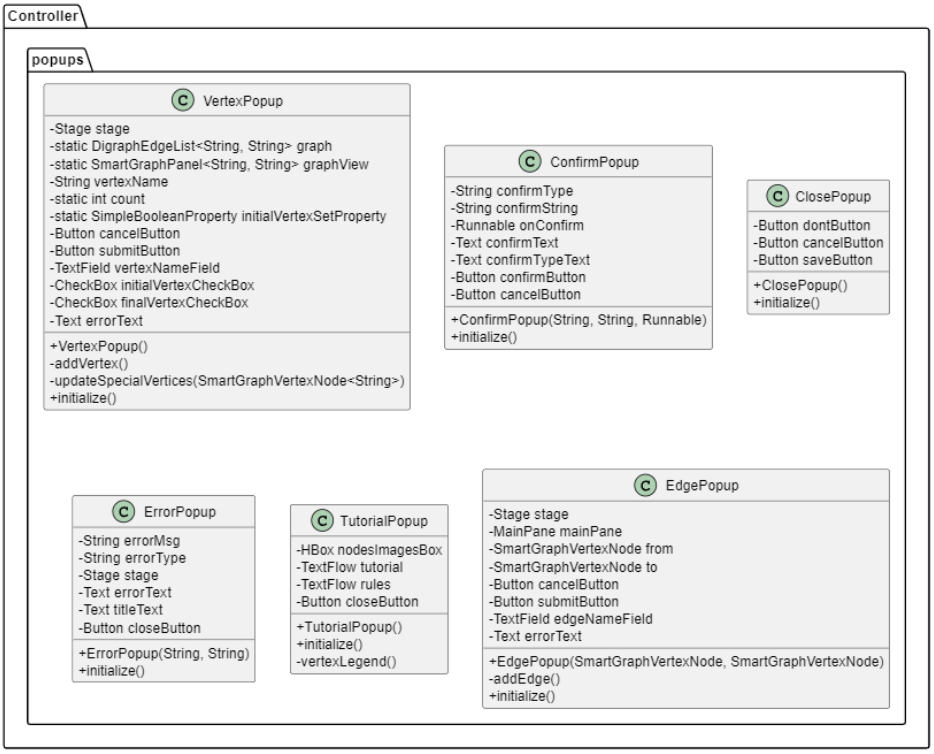


Figura 2.3: Diagramma delle classi dei popup

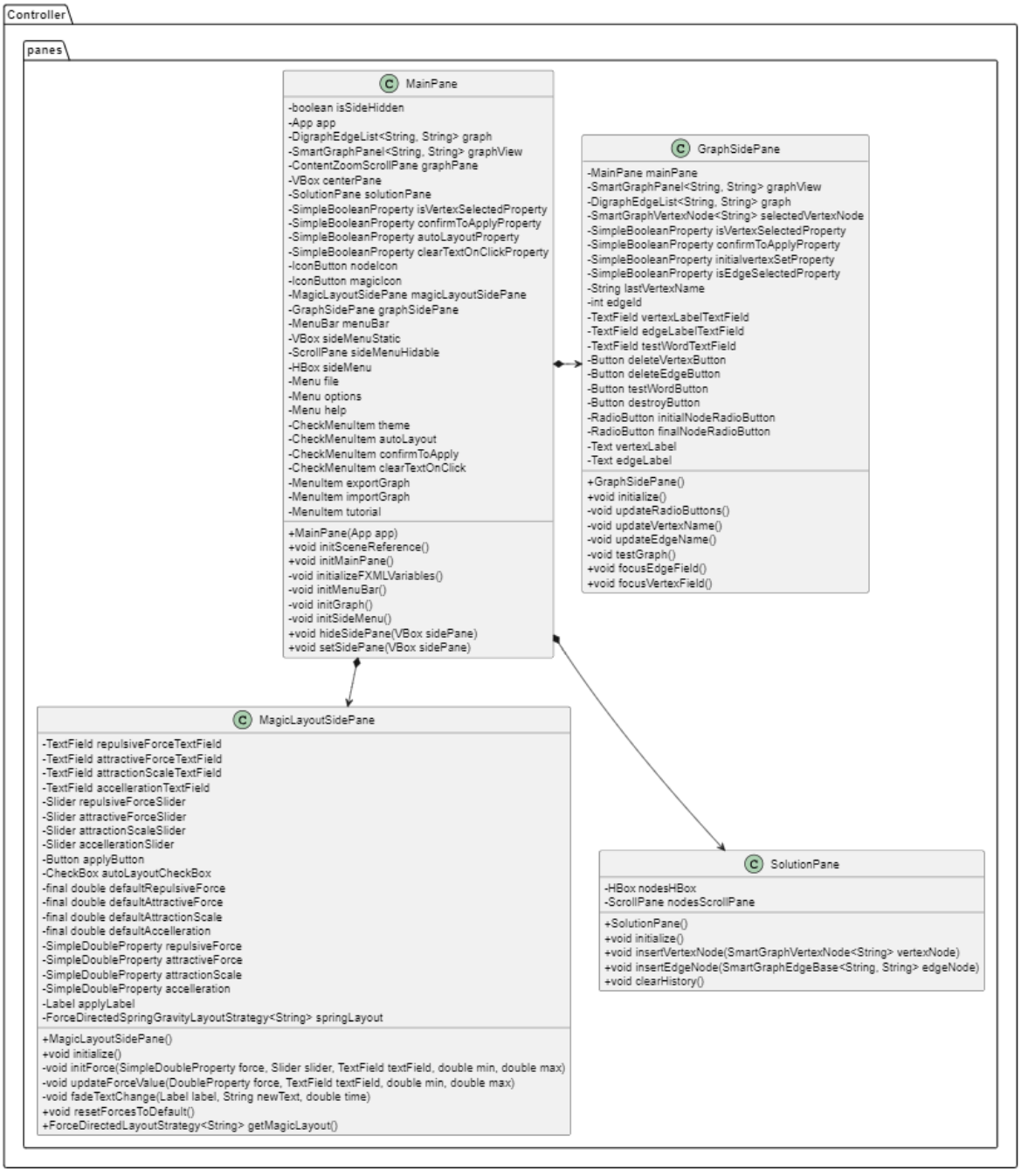


Figura 2.4: Diagramma delle classi dei pannelli

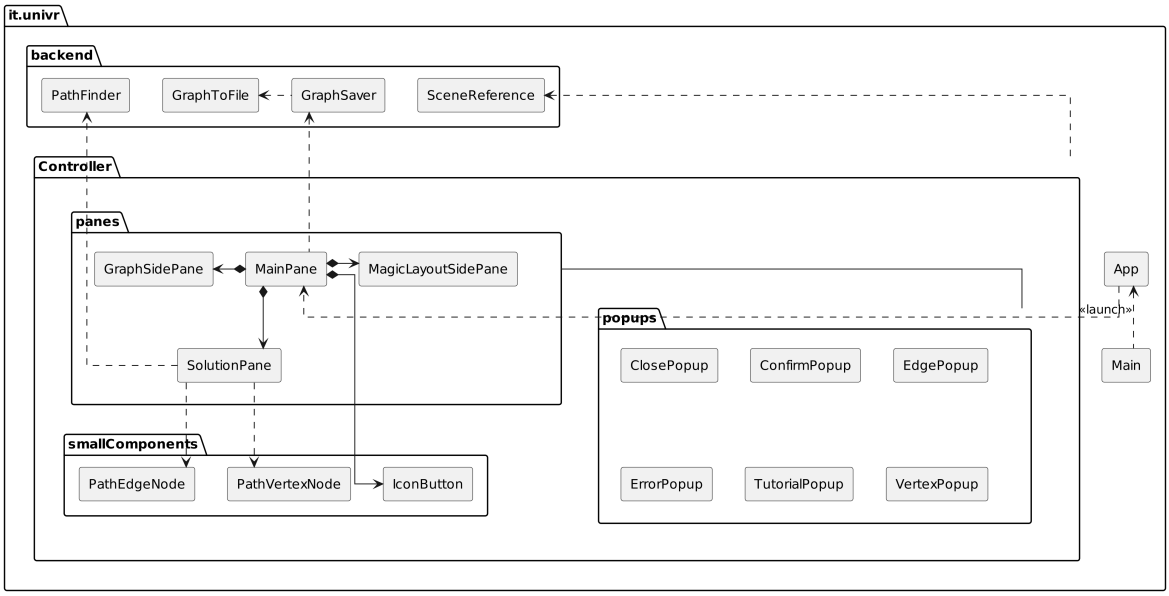


Figura 2.5: Diagramma delle classi generale del software

2.4 Diagrammi di sequenza

Nelle seguenti pagine riportiamo i diagrammi di sequenza delle componenti che abbiamo ritenuto più importanti. Sono presenti alcune semplificazioni necessarie principalmente per favorire la leggibilità degli schemi e per questioni di spazio.

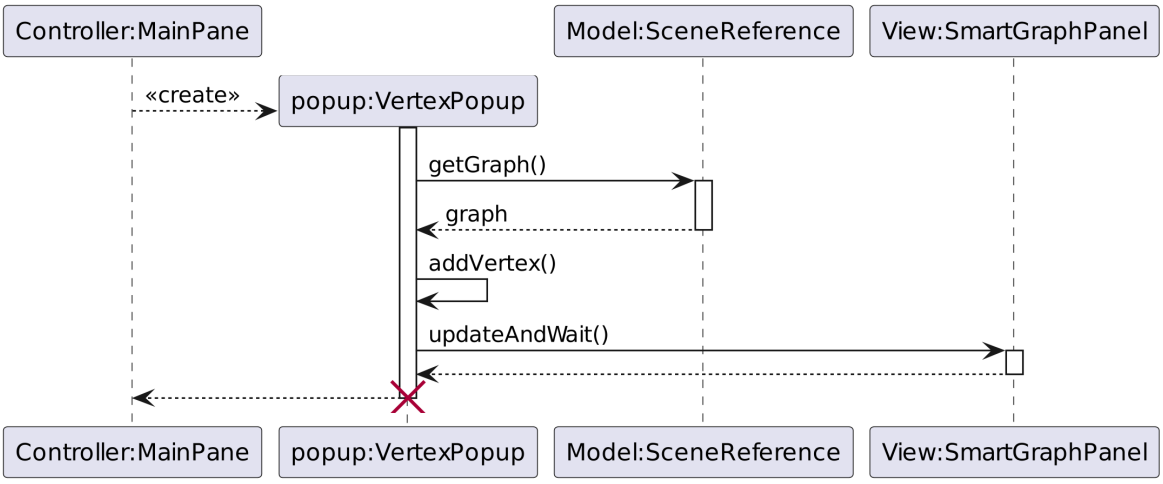


Figura 2.6: Diagramma di sequenza dell'aggiunta di un nodo all'automa

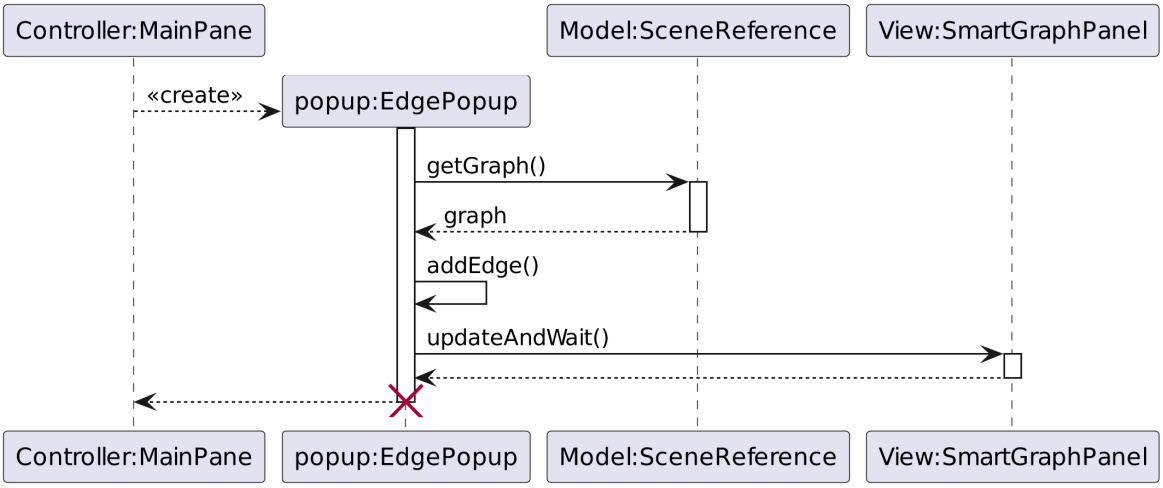


Figura 2.7: Diagramma di sequenza dell'aggiunta di una transizione all'automa

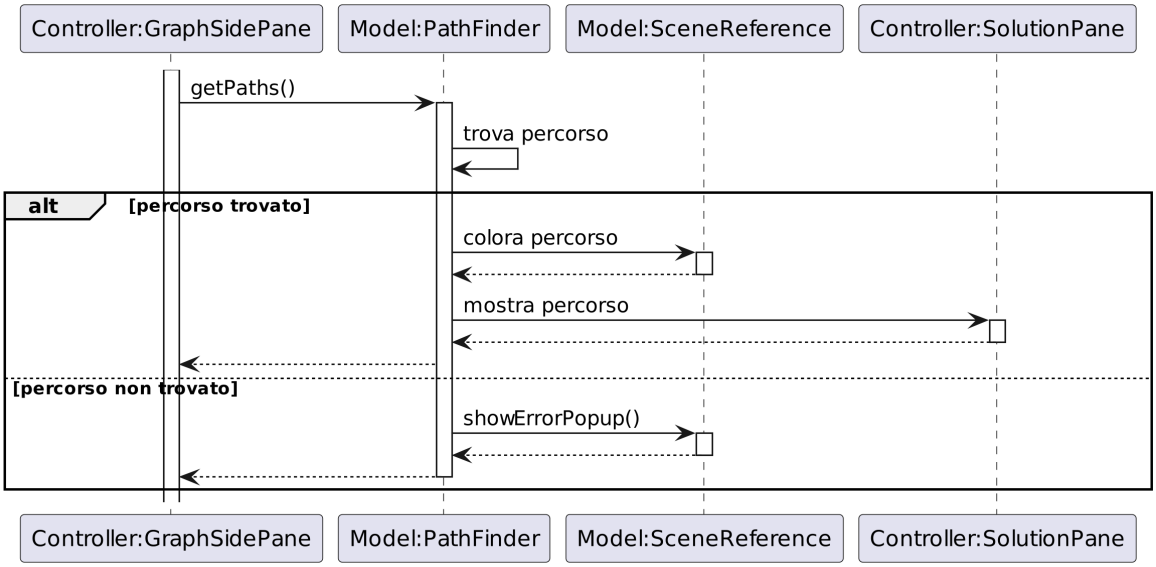


Figura 2.8: Diagramma di sequenza della ricerca di un percorso nell'automa

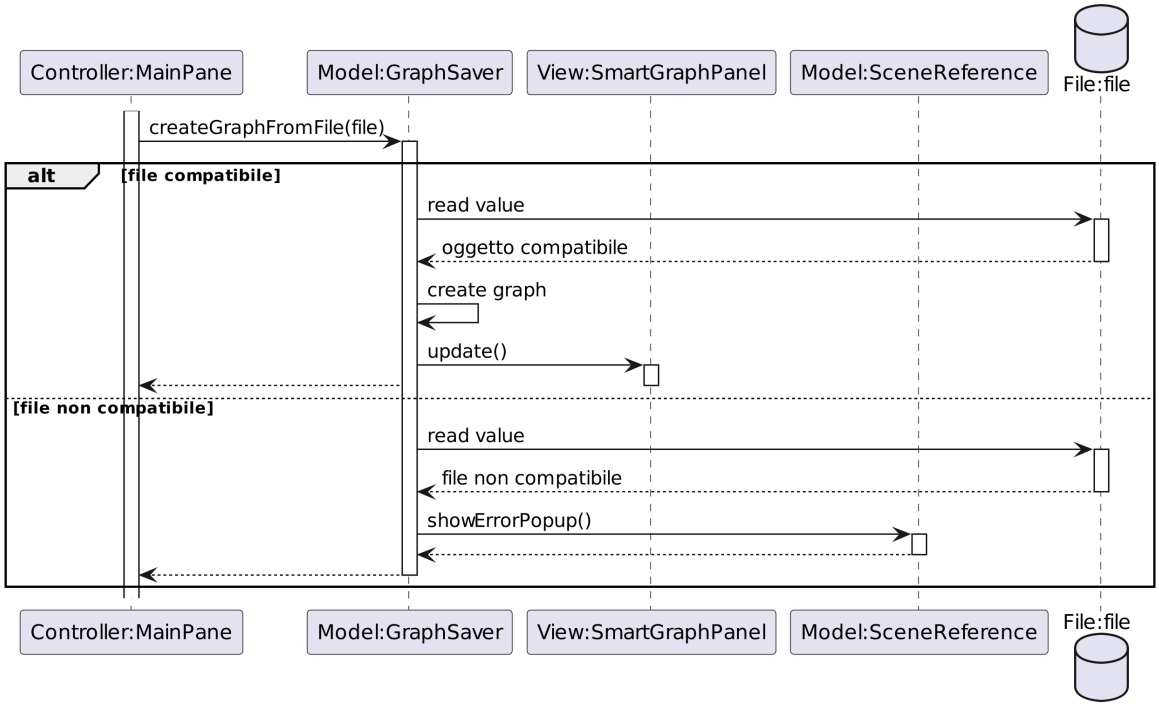


Figura 2.9: Diagramma di sequenza dell'importazione di un automa

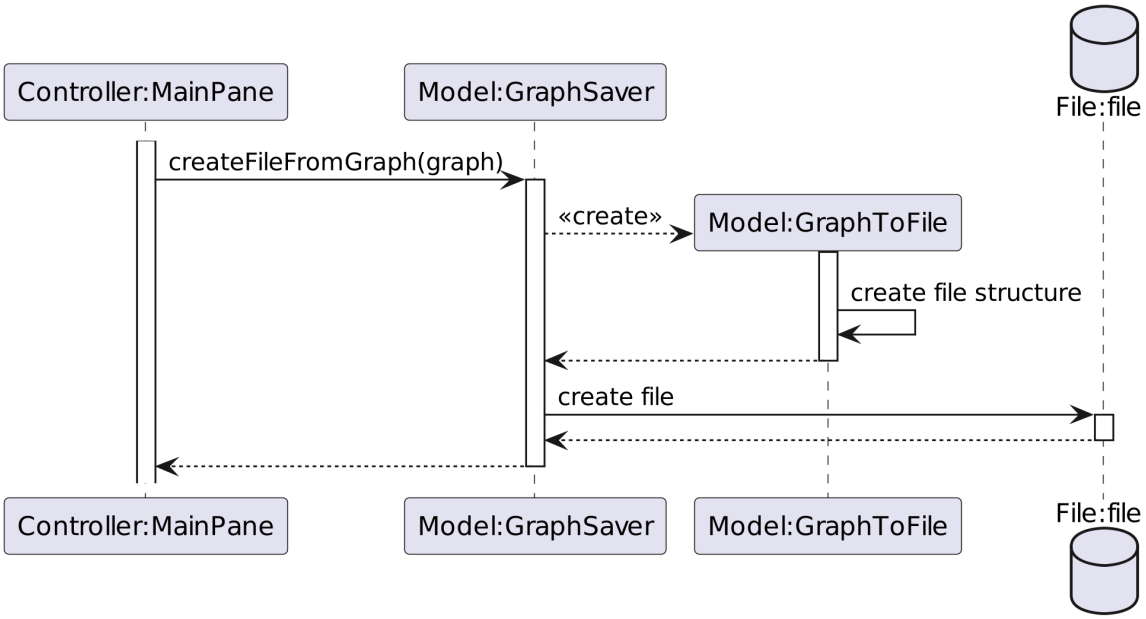


Figura 2.10: Diagramma di sequenza dell'esportazione dell'automa

3 Attività di test e validazione

Come precedentemente evidenziato, il progetto è stato approcciato in maniera test-driven, effettuando i test necessari immediatamente dopo l'implementazione della nuova funzionalità per assicurarne il corretto funzionamento. Questa modalità di sviluppo ha permesso una notevole semplificazione della fase di debugging fornendo la garanzia di lavorare con componenti funzionanti.

3.1 Testing Esterno

In questa fase di testing è stato fornito il software a possibili utenti finali con diversi livelli di competenze informatiche, ai quali è stato chiesto di eseguire diverse operazioni quali:

- aprire la finestra del tutorial
- creazione di stati
- creazione di transizioni tra gli stati precedentemente creati
- effettuare il test su almeno un percorso
- esportare l'automa e importarne uno fornito

L'obiettivo principale di questi test era di ricevere un feedback sulla facilità e comodità di utilizzo del software.

livello informatica	**Voto UI**	**Voto UX**
basso	9,5	7
alto	7	7,5
medio	8	10
alto	9,5	7
medio	10	8,5
basso	9	7,5

3.2 Testing interno

Durante la fase di sviluppo sono stati effettuati dal nostro team dei test su ogni unità e/o componente, segue la struttura di questi.
Il test inizia con il controllo della correttezza sintattica del codice, si procede poi con la verifica della correttezza semantica, una volta superati questi controlli il codice viene eseguito e si controlla il corretto funzionamento della nuova funzionalità implementata, se anche questo va a buon fine si procede con il verificare che l'implementazione della nuova funzionalità non abbia arrecato danni al resto del programma; nel momento in cui anche un singolo test fallisca, si identifica la causa del fallimento e una volta applicate le necessarie modifiche al codice si ripetono nell'ordine i test precedentemente elencati.

Seguono elencate le principali funzionalità testate ad ogni release

Aggiunta di uno stato	• Inserimento di un nome valido • Inserimento di un nome vuoto • Inserimento di un nome già presente
Aggiunta di una transizione	• Inserimento di un nome valido • Inserimento di un nome vuoto • Inserimento di un nome non valido
Modifica di uno stato	• Inserimento di un nome valido • Inserimento di un nome vuoto • Inserimento di un nome già esistente • Eliminazione di uno stato
Modifica di una transizione	• Inserimento di un nome valido • Inserimento di un nome vuoto • Inserimento di un nome non valido • Eliminazione di una transizione
Calcolo di un percorso	• Calcolo di un percorso esistente • Calcolo di un percorso non esistente(per qualsiasi motivo possibile) • Tentativo di calcolo di un percorso senza nodo iniziale impostato • Tentativo di calcolo di un percorso senza almeno un nodo finale • Tentativo di calcolo di un percorso vuoto
Salvataggio di un automa	• Salvataggio di un automa qualsiasi
Caricamento di un automa	• Caricamento di un file contenente un automa valido • Tentativo di caricamento di un file non contenente un automa valido