**THE ITALIAN JOB**

****

**MEMBRI DEL GRUPPO**

* Bartolomeo Roberto Siciliano
* Giancarlo Pandolfi
* Giacomo Pagliara

**1.INTRODUZIONE**

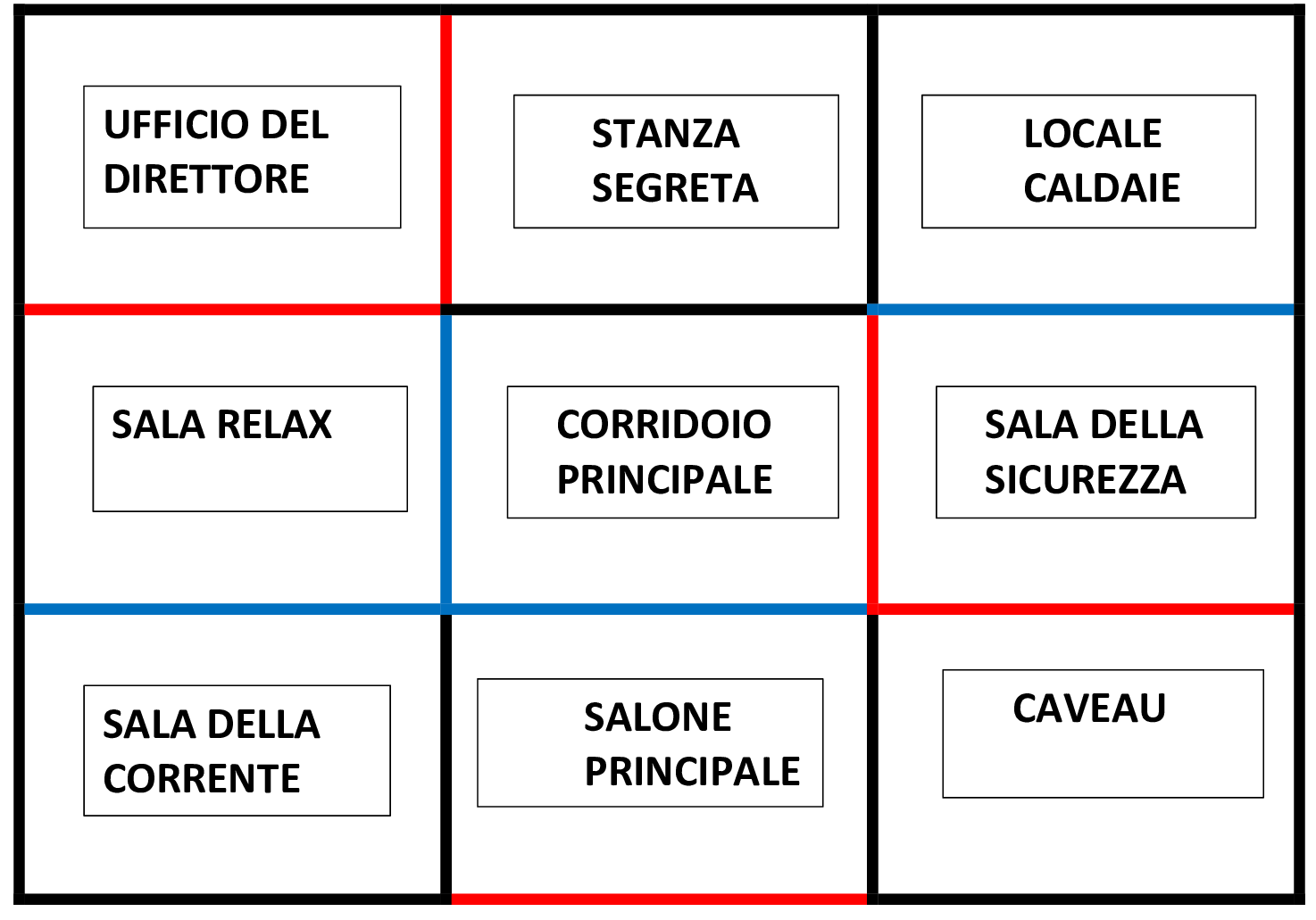
Questo documento è la relazione tecnica finale per il progetto che implementa il gioco **“The Italian Job”** del gruppo **PSP.** Il gioco si presenta come un’avventura testuale sviluppata interamente in linguaggio Java ed offre un’interfaccia utente con cui quest’ultimo può interagire.

L’interfaccia utente è stata implementata sia in versione grafica (Graphical User Interface, GUI) sia a riga di comando (Command Line Interface, CLI).

La storia del gioco realizzato ruota intorno a una rapina in banca, infatti il protagonista è un tecnico elettrico che viene chiamato dalla banca per risolvere un guasto all’impianto elettrico, ma il protagonista ha in mente un secondo scopo ovvero quello di rubare il tesoro contenuto nel caveau della banca.

Per poter compiere azioni nel gioco, l’utente dovrà inserire il comando e l’eventuale oggetto o personaggio con il quale si vuole interagire; in seguito alla pressione del tasto “Invio” o del JButton corrispondente, il comando viene processato in modo da essere riconosciuto al fine di attivare l’evento di gioco associato.

* 1. **MAPPA DI GIOCO**



* **LEGENDA**

|  |  |
| --- | --- |
| **COLORE LINEA** | **DESCRIZIONE** |
|  | MURO |
|  | PORTA APERTA |
|  | PORTA CHIUSA |

La struttura del gioco è basata sulla creazione e gestione di una mappa composta da varie stanze, le quali conterranno gli oggetti con i quali l’utente potrà interagire.

Per muoversi da una stanza all’altra l’utente dovrà inserire la direzione in cui intende muoversi specificando uno dei quattro punti cardinali (Nord, Sud, Est, Ovest). In caso di presenza di elementi che non consentono l’avanzamento nella direzione inserita (porte chiuse, nemici, etc.) , per proseguire l’utente dovrà prima liberare il passaggio compiendo determinate azioni in base al tipo di impedimento presente.

In particolare:

* per sbloccare la sala della sicurezza è necessario indossare il travestimento da guardia recuperabile attraverso l’uccisione di una guardia dopo aver premuto un interruttore nella sala della corrente che genera un blackout;
* per sbloccare l’ufficio del direttore è necessario premere l’interruttore nel locale caldaie;
* per sbloccare la stanza segreta è necessario spostare il tappeto nell’ufficio del direttore, alzare la botola e tirare la leva;
* Per sbloccare il caveau è necessario raccogliere il pass del direttore nell’ufficio del direttore e la password nella stanza segreta.

Il gioco inizia con il protagonista che si trova all’interno della banca, precisamente nel salone principale.

**2. REQUISITI SPECIFICI**

**2.1 REQUISITI FUNZIONALI**

I requisiti funzionali descrivono le funzionalità del software in termini di:

* *Servizi che il software deve fornire*
* *Risultati che il software deve produrre in risposta a specifici input*

Questo software fornisce le seguenti funzionalità:

|  |  |
| --- | --- |
| REQUISITO | DESCRIZIONE |
| Spostarsi a nord | Al comando “Nord” o vai a ‘nord’ l’applicazione deve muovere il protagonista verso la stanza adiacente a Nord. |
| Spostarsi a sud | Al comando “Sud” l’applicazione deve muovere il protagonista verso la stanza adiacente a Sud. |
| Spostarsi a est | Al comando “Est” l’applicazione deve muovere il protagonista verso la stanza adiacente a Est. |
| Spostarsi a ovest | Al comando “Ovest” l’applicazione deve muovere il protagonista verso la stanza adiacente a Ovest. |
| Mostrare l’elenco dei comandi | Al comando “Help” l’applicazione deve mostrare la lista dei comandi utilizzabili dall’utente. |
| Mostrare la descrizione di una stanza | Al comando “Osserva” l’applicazione deve mostrare la descrizione della stanza in cui si trova il giocatore e gli eventuali oggetti presenti in essa. |
| Mostrare la descrizione di un oggetto | Al comando “Esamina <oggetto>” l’applicazione deve mostrare la descrizione dell’oggetto inserito dopo il comando. |
| Mostrare l’inventario | Al comando “Inventario” l’applicazione deve mostrare la lista degli oggetti presenti nell’inventario. |
| Prendere un oggetto | Al comando “Prendi <oggetto>” l’applicazione deve prendere l’oggetto inserito dopo il comando, porlo nell’inventario e rimuoverlo dalla stanza. |
| Lasciare un oggetto | Al comando “Lascia <oggetto>” l’applicazione deve rimuovere dall’inventario l’oggetto inserito dopo il comando e lasciarlo nella stanza corrente. |
| Aprire un oggetto contenitore | Al comando “Apri <oggetto>” l’applicazione deve aprire l’oggetto contenitore inserito dopo il comando, lasciando gli oggetti contenuti dal contenitore, nella stanza corrente o nell’inventario a seconda se il contenitore si trova nella stanza o nell’inventario. |
| Chiudere un oggetto contenitore | Al comando “Chiudi <oggetto>” l’applicazione deve chiudere l’oggetto contenitore inserito dopo il comando, riponendo all’interno del contenitore gli oggetti che vi erano presenti prima di aprirlo, se quest’ultimi si trovano in prossimità dell’oggetto contenitore. |
| Premere un oggetto | Al comando “Premi <oggetto>” l’applicazione deve premere l’oggetto inserito dopo il comando. |
| Usare un oggetto | Al comando “Usa <oggetto>” l’applicazione deve consentire all’utente di utilizzare l’oggetto inserito dopo il comando per compiere una determinata azione. |
| Salvare la partita | Al comando “Salva” l’applicazione deve salvare su file la partita corrente. |
| Caricare la partita | Al comando “Carica” l’applicazione deve caricare da file la partita precedentemente salvata. |
| Uscire dalla partita | Al comando “Esci” l’applicazione deve consentire all’utente di uscire dalla partita corrente. |

**2.2 REQUISITI NON FUNZIONALI**

I requisiti non funzionali descrivono le caratteristiche qualitative del software sviluppato.

* *Robustezza:* il software è in grado di gestire le situazioni in cui si manifestano errori o eccezioni dovute a comandi non validi inseriti dall’utente;
* *Estetica dell’interfaccia utente:* il software implementa una GUI in cui i comandi principali sono rappresentati da dei JButtons;
* *Modularità:* il software è realizzato rispettando i principi del linguaggio OO Java;
* *Riusabilità:* il software è progettato in modo da poter essere utilizzato per l’implementazione di altri giochi simili, purché questi implementino i metodi astratti della classe ‘*GameDescription*’;
* *Portabilità:* il software può essere utilizzato su diversi sistemi operativi disponibili, grazie alle funzionalità offerte dallo sviluppo di un progetto Maven;
* *Estendibilità*: il software è predisposto ad accettare cambiamenti o aggiunte di nuove features come nuovi comandi, oggetti, stanze o personaggi;
* *Manutenibilità*: La manutenibilità del software è garantita grazie ai principi dell’Information Hiding applicati, alla modularità delle componenti e alla presentazione separata della componenti.

**3.SCELTE DI PROGETTO**

Lo sviluppo del progetto si basa sulla tassonomia *‘Entity – Control - Boundary’* che consiste nel partizionare il sistema in tre tipi di classi, in particolare, le classi di tipo *Boundary* hanno la responsabilità di gestire la comunicazione con l’utente, le classi di tipo *Control* hanno la responsabilità di gestire il flusso dell’applicazione e rappresentano la logica del sistema, le classi di tipo *Entity* rappresentano l’astrazione del concetto del modello di dominio e si occupano della persistenza dei dati infatti svolgono prevalentemente operazioni di tipo ‘*get’* e ‘*set’* degli attributi.

**3.1 GESTIONE INPUT – OUTPUT**

L’applicazione ha due interfacce per interagire con l’utente, una grafica e una a riga di comando, entrambe gestite da classi di tipo *Boundary* che si occupano di prendere il comando inserito dall’utente e stampare una risposta a seguito dell’elaborazione del comando attraverso classi di tipo Control.

Entrambe possiedono un menu iniziale attraverso il quale l’utente può scegliere se iniziare una nuova partita, caricare una partita precedentemente salvata o terminare l’applicazione.

* L’interfaccia grafica (GUI) viene rappresentata dalla classe ‘*StartMenu*’ che, una volta specificato il tipo di azione tra le tre possibili sopra descritte, passa il controllo di gestione di input-output alla classe ‘*GameGUI*’ che si occupa di leggere da una *JTextField* una frase inserita dall’utente e stampare una risposta su un *JTextPane*, inoltre permette di visualizzare il contenuto dell’inventario e inserire i comandi più comuni (spostamenti e osservare una stanza) attraverso dei *JButton.*
* L’interfaccia a riga di comando (CLI) si comporta come comporta come quella grafica, in particolare ha un menu iniziale rappresentato dalla classe ‘*StartMenuCLI*’ che rappresenta il punto di ingresso del programma da riga di comando che in seguito alla scelta di iniziare una partita cede il controllo ad un proprio metodo ‘execute()’ che gestisce l’input-output con l’utente attraverso la console da riga di comando.

Una volta iniziata una partita viene creata un’istanza della classe Engine che si occupa della creazione di un Parser e di un’istanza della classe GameDescription per inizializzare il gioco.

**3.2 PARSER**

La classe Parser si occupa di ricevere la frase inserita dall’utente e restituire un oggetto di tipo ParserOutput che conterrà un comando valido seguito da due ‘entita’ se presenti.

Il parser accetta frasi del tipo <comando> <entità1> <entità2> dove per entità si intende un oggetto presente nella stanza corrente, un oggetto dell’inventario o un npc, l’ordine in cui vengono inserite le entità non è importante.

Il parser non è key sensitive, infatti è possibile inserire frasi dove non è importante l’uso delle maiuscole o minuscole, possono essere inseriti segni di punteggiatura e simboli speciali e non è sensibile agli articoli, preposizioni o aggettivi poiché ‘elimina’ le parole contenute in un insieme di ‘stop word’ presente su file, inoltre i comandi e le entità possono essere inseriti attraverso l’utilizzo di diversi sinonimi inseriti al momento di inizializzazione del gioco.

Il parser genera un’eccezione nel caso in cui nella frase analizzata non è presente un comando o ne è presente più di uno poiché il gioco accetta un comando per ‘mossa’.

**3.3 GESTIONE DELLE MOSSE**

Una volta analizzata la frase le classi di tipo *Boundary* (*StartMenuCLI* e *GameGUI*) richiamano il metodo *‘nextMove’* della classe ‘*TheItalianJob’* la quale rappresenta il gioco vero e proprio, questo metodo si serve della classe ‘*CommandManager*’ di tipo Control per gestire la logica del gioco. La classe *CommandManager* analizza la mossa passata in input e si occupa di eseguire il comando, se è corretto, restituire una risposta e modificare il contenuto degli attributi di gioco (stanze, oggetti, inventario, npc), dove possibile si è cercato di implementare i controlli attraverso la programmazione funzionale attraverso espressioni lambda e pipeline.

**3.4 LA CLASSE GAME DESCRIPTION**

La classe astratta *GameDescription* rappresenta la classe di un generico gioco, attraverso essa è possibile implementare diversi giochi purché questi implementino i metodi astratti *init()* per inizializzare una partita (creazione comandi, mappa, oggetti e personaggi), *nextMove()* per la gestione delle mosse, *save()* e *load()* per il salvataggio e caricamento di una partita.

La classe è composta dalle classi di tipo *Entity* che compongono il gioco:

un insieme di stanze, dove ogni stanza possiede al suo interno una lista di oggetti o npc, la stanza corrente, un insieme di oggetti (inventario), un insieme di npc e il punteggio.

**3.5 PUNTEGGIO**

Il punteggio è rappresentato dalla classe Score la quale calcola il punteggio in base al numero di mosse effettuate e agli oggetti ‘speciali’ presi o utilizzati, cioè oggetti non obbligatori al fine del completamento del gioco (gioielli nella stanza segreta, disattivare le telecamere o chiamare la polizia). In particolare, si parte da un punteggio massimo di 60 al quale si sommano 10 punti per ogni oggetto speciale, inoltre il punteggio inizia a scendere dal momento in cui si supera il numero di mosse minimo per finire il gioco, in particolare per ogni mossa in più verrà scalato un punto.

**3.6 SALVATAGGIO E CARICAMENTO DELLA PARTITA**

Durante il gioco è stata inserita la possibilità di salvare la partita e caricarla successivamente in modo da non dover ricominciare da fissare dei checkpoint se l’utente lo ritiene necessario. Per poter salvare la partita l’utente dovrà inserire il comando di tipo “Save”, invece per caricare una partita precedentemente salvata si dovrà inserire il comando di tipo “Load”. Quest’ultimi sfruttando la serializzazione in byte degli oggetti che compongono il gioco, della classe *GameDescription,* salvano e caricano da file gli oggetti di interesse.

**3.7 DATABASE**

I dati relativi alle stanze, agli oggetti e agli npc (nomi, descrizioni, ecc.) sono contenuti su DataBase attraverso la classe DBManager, questa si occupa della creazione, del popolamento e delle interrogazioni delle tabelle.

Tutte i dati da inserire nelle tabelle e le query, sono contenuti nel pacchetto entityDescription divise in classi per stanze, oggetti e npc. In questo modo è possibile cambiare le descrizioni semplicemente modificando le stringhe presenti come costanti.

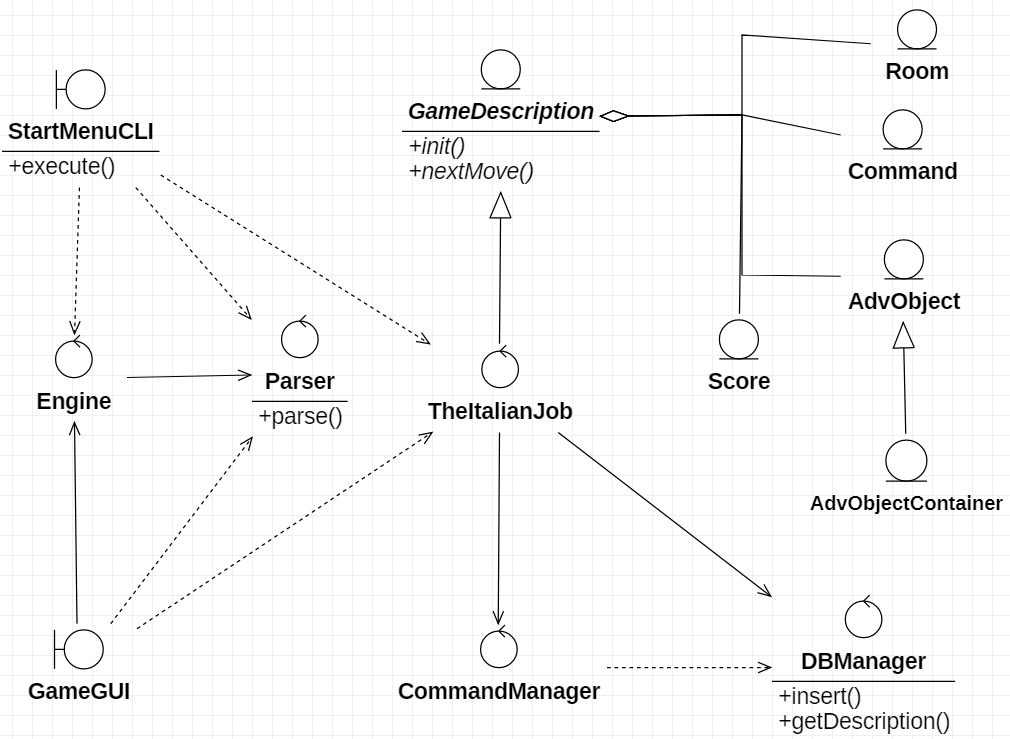
Per la connessione al DB sono stati impostati nome utente e password per renderlo più sicuro e per rendere il programma indipendente dal tipo di DB in uso, si è utilizzata una stringa di connessione presente su file, in questo modo è possibile cambiare DB semplicemente cambiando la stringa di connessione da file.

Per le interrogazioni delle tabelle (popolamento e recupero dei dati) sono state utilizzate query preimpostate per rendere il programma indipendente dalla sintassi del DB utilizzato.

Inoltre l’uso di database offre diversi vantaggi tra cui:

* *Indipendenza logica e fisica dei dati, ossia la modifica della struttura logica e/o fisica dei dati non implica dover apportare modifiche alle applicazioni utilizzatrici;*
  + - *Sicurezza (riservatezza ed integrità dei dati);*
    - *Condivisione dei dati;*
    - *Indipendenza dalla piattaforma hardware/software;*
    - *Semplificazione nello sviluppo delle applicazioni.*

**4. DIAGRAMMA DELLE CLASSI**

****

**5.SPECIFICA ALGEBRICA**

Abbiamo deciso di riportare la specifica algebrica

relativa alla struttura dati “lista” , la quale è stata

utilizzata nel progetto in svariate situazioni.

Una lista è una sequenza finita di elementi dello

stesso tipo e si può accedere direttamente solo al

primo elemento; la lista è una struttura dati

dinamica(a dimensione variabile) , infatti è

possibile inserire e cancellare elementi.

**SPECIFICA SINTATTICA**

**Tipi:**

Lista , posizione , tipoelem , boolean.

**Operatori:**

* crealista: () -> lista
* *crea una nuova lista*
* listavuota: (lista) -> boolean
* *stabilisce se la lista è vuota o meno*
* leggilista: (posizione , lista) -> tipoelem
* *restituisce l’elemento di una lista*
* scrivilista: (tipoelem , posizione , lista) -> lista
* *sovrascrive un elemento della lista*
* primolista: (lista) -> posizione
* *restituisce la posizione del primo elemento della lista*
* finelista: (posizione , lista) -> boolean
* stabilisce se la posizione inserita coincide con la fine della lista
* succlista: (posizione , lista) -> posizione
* *restituisce la posizione successiva rispetto a quella inserita*
* predlista: (posizione , lista) -> posizione
* *restituisce la posizione precedente rispetto a quella inserita*
* inslista: (tipoelem , posizione , lista) -> lista
* *inserisce un elemento nella lista*
* canclista: (posizione , lista) -> lista
* *cancella un elemento dalla lista*

**SPECIFICA SEMANTICA**

**Tipi:**

* Lista: insieme delle sequenze l = <a1,a2,…,an> , n > 0 , di elementi di tipo “tipoelem” dove l’elemento i-esimo ha valore a(i) e posizione pos(i)
* boolean: insieme dei valori di verità {true , false}
* posizione: pos(i) identifica la posizione dell’i-esimo elemento nella lista
* tipoelem: tipo degli elementi ospitati nella lista

**Operatori:**

* crealista() = l’
* POST: l’ = <> (lista vuota)
* listavuota(l) = b
* POST: b = true se l=<> , altrimenti b = false
* leggilista(p,l) = a
* PRE: p = pos(i) con 1 <= i <= n
* POST: a = a(i)
* scrivilista(a,p,l) = l’
* PRE: p = pos(i) con 1 <= i <= n
* POST: l’ = <a1,a2,…,a(i-1),a,a(i+1),…,an>
* primolista(l) = p
* PRE: listavuota(l) = false
* POST: p = pos(1)
* finelista(p,l) = b
* PRE: p = pos(i) con 1 <= i <= n+1
* POST: b = true se p = pos(n+1) , altrimenti b = false
* succlista(p,l) = p’
* PRE: p = pos(i) con 1 <= i <= n
* POST: p’ = pos(i+1)
* predlista(p,l) = p’
* PRE: p = pos(i) con 2 <= i <= n
* POST: p’ = pos(i-1)
* inslista(a,p,l) = l’
* PRE: p = pos(i) con 1 <= i <= n+1
* POST: l’ = <a1,a2,…,a(i-1),a,a(i),a(i+1),…,an> , se 1 <= i <= n

l’ = <a1,a2,…,an,a> , se i = n+1

* canclista(p,l) = l’
* PRE: p = pos(i) con 1 <= i <= n
* POST: l’ = <a1,a2,…,a(i-1),a(i+1),…,an>

**OPERAZIONI**

**COSTRUTTORI DI l’**

|  |  |  |
| --- | --- | --- |
|  | crealista() | inslista(a,p,l) |
| listavuota(l’) | true | false |
| leggilista(p’,l’) | error | If (p=p’) then a , else leggilista(p’,l) |
| canclista(p’,l’) | error | If (p=p’) then l ,  else inslista(a,p,canclista(p’,l)) |
| finelista(p’,l’) | true | If (p’=primolista(l’)) then false ,  else finelista(p’,l) |
| scrivilista(a’,p’,l’) | error | inslista(a’,p’,canclista(p’,l’)) |
| primolista(l’) | error | If (listavuota(canclista(p,l))) then p ,  else primolista(l) |
| succlista(p’,l’) | error | If (finelista(p’,l’)) then error ,  else succlista(p’,l’) |
| predlista(p’,l’) | error | If (p’=primolista(l’)) then error ,  else predlista(p’,l’) |

**6. ORGANIZZAZIONE DEL LAVORO**

Il gruppo ha utilizzato la piattaforma Microsoft Teams per organizzare riunioni nelle quali venivano divisi i compiti e si discuteva delle scelte di progetto adottate; per risolvere problemi più complessi venivano organizzate delle riunioni extra per risolvere insieme il problema.

Per condividere il lavoro fra i vari membri del gruppo è stato utilizzato il software GitHub con la creazione di un apposito repository; questo software è stato fondamentale per condividere al meglio il lavoro svolto da remoto data l’impossibilità di incontrarci fisicamente.

Tutti i membri del gruppo hanno utilizzato come ambiente di sviluppo Apache NetBeans.