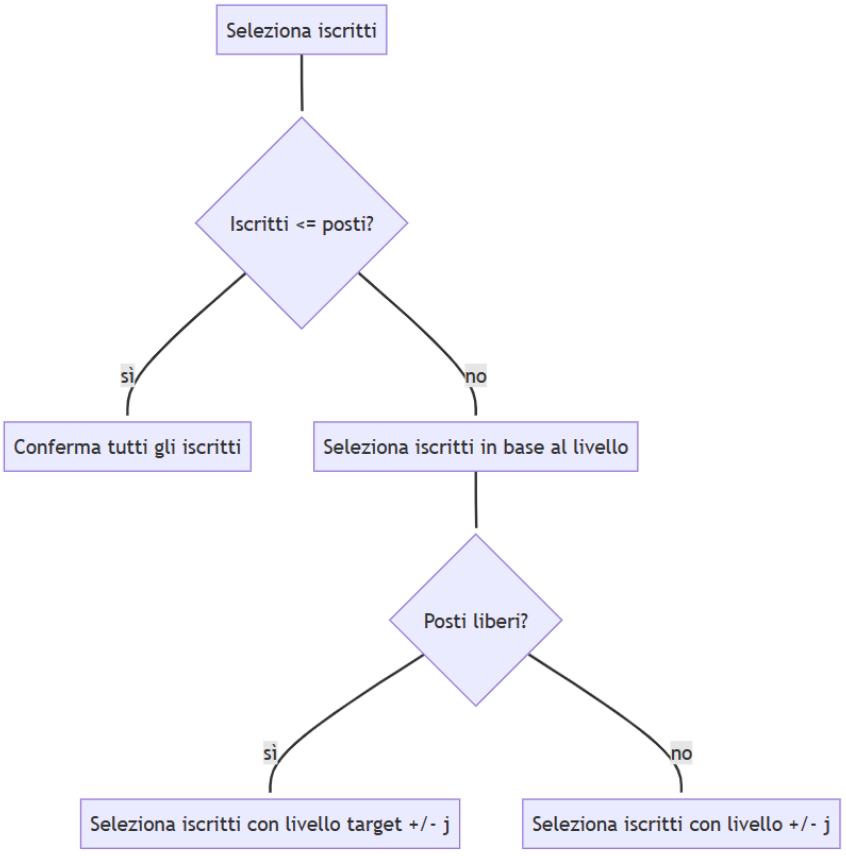
## Algoritmo: Strategia Greedy per la Selezione degli Iscritti

**Introduzione:**
La strategia greedy è un paradigma di risoluzione di problemi che fa scelte localmente ottimali in ogni passo con l'obiettivo di ottenere una soluzione globalmente ottimale. Nel contesto della selezione degli iscritti a un evento, la strategia greedy può essere applicata per massimizzare l'efficienza e soddisfare requisiti specifici come il numero massimo di partecipanti e la considerazione del livello di esperienza degli utenti.

**Flowchart e pseudocodice:**
L'algoritmo `selezionaIscritti` qui di seguito implementa una strategia greedy per la selezione degli iscritti a un evento, tenendo conto del numero massimo di partecipanti e del livello degli utenti. L'approccio si basa sull'idea di selezionare in modo iterativo gli utenti con il livello più adatto, iniziando dal livello dell'evento e ampliando la ricerca ai livelli adiacenti solo se necessario.



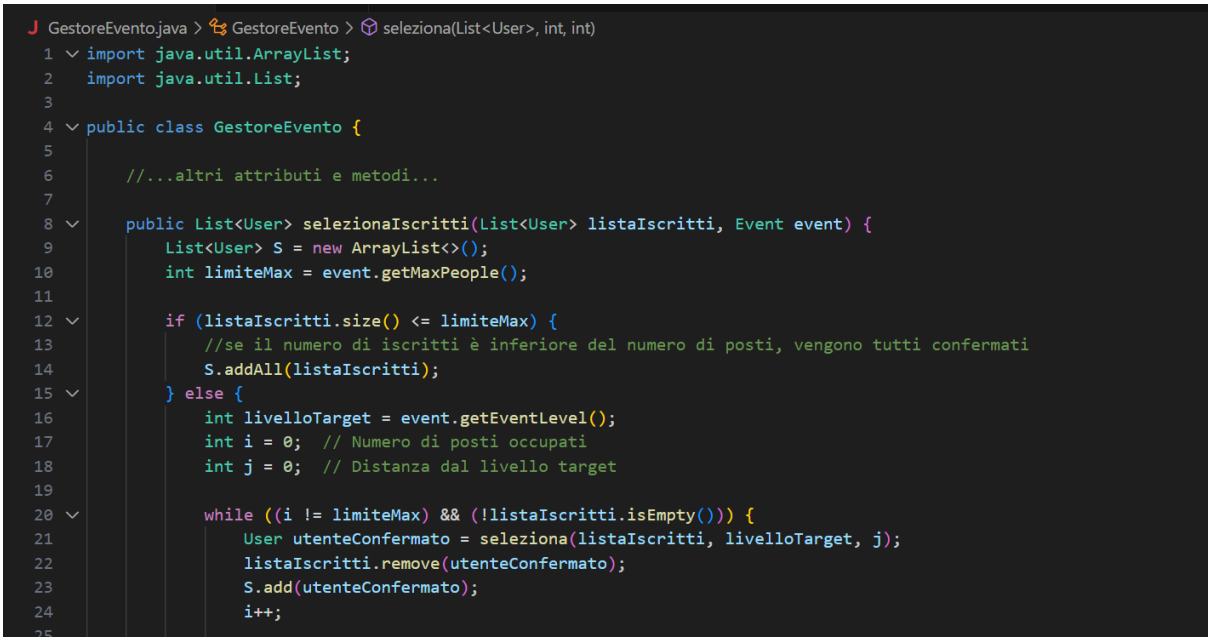
pseudocodice.txt

```
@metodo di EventManager

algoritmo selezionaIscritti(Lista listaIscritti)
    S <- {}
    limiteMax = Event.maxPeople

    if (listaIscritti.dim <= limiteMax) then S <- all(listaIscritti)
    else
        livello = Event.level
        int i = 0  //posti occupati
        int j = 0  //distanza dal livello evento

        while((i != limiteMax) and (listaIscritti != {})) do
            utenteConfermato = seleziona(listaIscritti, livello, j)
            if(not listaIscritti.contains(anotherUser con userLevel = livello +- j)) then j+1
            listaIscritti <- listaIscritti / {utenteConfermato}
            S <- S U {utenteConfermato}
            i + 1

    return S

//prima vengono considerati gli utenti con lo stesso livello dell'evento
//poi quelli ai livelli +-1, +-2 fino a esaurimento posti

funzione seleziona(listaIscritti, livello, distanza)
    return (User con userLevel = livello + distanza) or (User con userLevel - distanza)
```

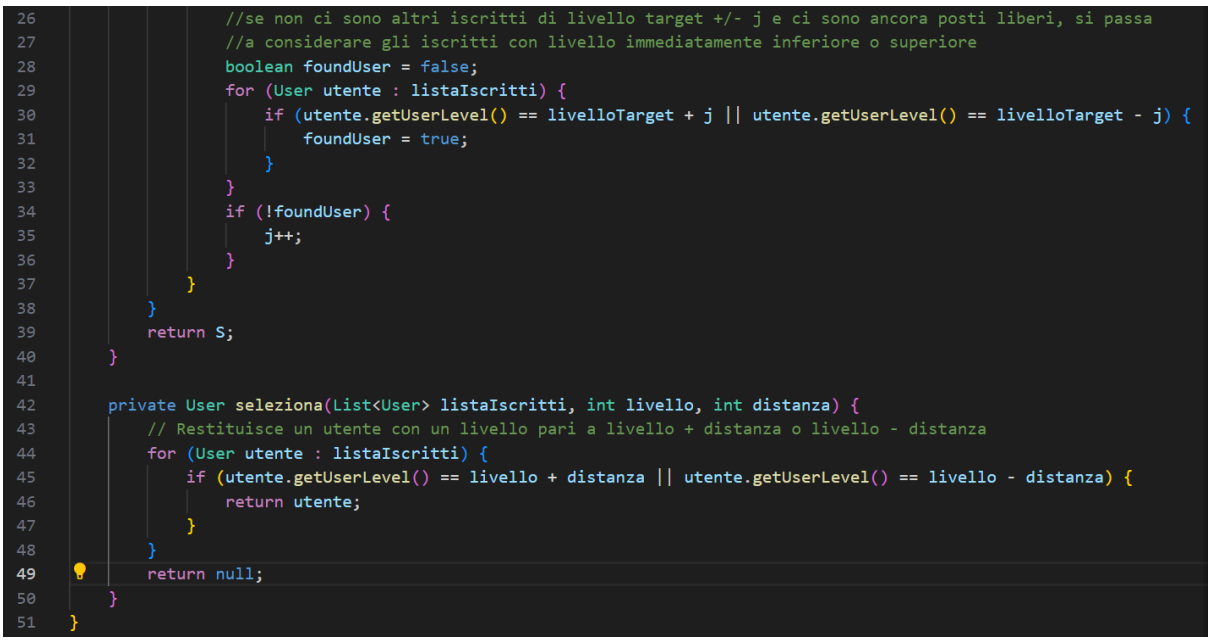
Questo algoritmo mira a massimizzare la partecipazione all'evento in modo efficiente, seguendo una logica greedy nell'assegnazione dei posti agli iscritti. La selezione degli utenti avviene in modo iterativo, considerando prima gli utenti con il livello esatto dell'evento e successivamente estendendo la ricerca ai livelli adiacenti solo se necessario.

**Implementazione in Java:**

Ecco un'implementazione in Java dell'algoritmo descritto nello pseudocodice. Questa implementazione utilizza una classe `User` e una classe `Event` per rappresentare gli utenti e gli eventi, rispettivamente. La funzione `selezionaIscritti` restituisce l'insieme di utenti selezionati per partecipare all'evento.



```java
GestoreEvento.java > GestoreEvento > seleziona(List<User>, int, int)
import java.util.ArrayList;
import java.util.List;

public class GestoreEvento {

    //...altri attributi e metodi...

    public List<User> selezionaIscritti(List<User> listaIscritti, Event event) {
        List<User> S = new ArrayList<>();
        int limiteMax = event.getMaxPeople();

        if (listaIscritti.size() <= limiteMax) {
            //se il numero di iscritti è inferiore del numero di posti, vengono tutti confermati
            S.addAll(listaIscritti);
        } else {
            int livelloTarget = event.getEventLevel();
            int i = 0;  // Numero di posti occupati
            int j = 0;  // Distanza dal livello target

            while ((i != limiteMax) && (!listaIscritti.isEmpty())) {
                User utenteConfermato = seleziona(listaIscritti, livelloTarget, j);
                listaIscritti.remove(utenteConfermato);
                S.add(utenteConfermato);
                i++;
```



```java
                //se non ci sono altri iscritti di livello target +/- j e ci sono ancora posti liberi, si passa
                //a considerare gli iscritti con livello immediatamente inferiore o superiore
                boolean foundUser = false;
                for (User utente : listaIscritti) {
                    if (utente.getUserLevel() == livelloTarget + j || utente.getUserLevel() == livelloTarget - j) {
                        foundUser = true;
                    }
                }
                if (!foundUser) {
                    j++;
                }
            }
        }
        return S;
    }

    private User seleziona(List<User> listaIscritti, int livello, int distanza) {
        // Restituisce un utente con un livello pari a livello + distanza o livello - distanza
        for (User utente : listaIscritti) {
            if (utente.getUserLevel() == livello + distanza || utente.getUserLevel() == livello - distanza) {
                return utente;
            }
        }
        return null;
    }
}
```

**Analisi di Complessità:**

Complessità Temporale:

1. Caso Migliore (Numero di iscritti <= Numero massimo di posti):
   Nel caso in cui il numero di iscritti sia inferiore o uguale al numero massimo di posti disponibili, l'algoritmo esegue una copia di tutti gli iscritti nella lista `S`.
   Complessità Temporale: O(n), dove n è la dimensione della lista degli iscritti.

1. Caso Peggiore (Numero di iscritti > Numero massimo di posti):

Nel caso in cui il numero di iscritti superi il numero massimo di posti, l'algoritmo utilizza due cicli while annidati. Il ciclo esterno viene eseguito fino a quando non vengono occupati tutti i posti desiderati o la lista degli iscritti è vuota, mentre il ciclo interno verifica la presenza di utenti con il livello desiderato nella lista degli iscritti. Nel peggiore dei casi, il numero di iterazioni del ciclo esterno è limitato dal numero massimo di posti e dalla dimensione della lista degli iscritti.
Complessità Temporale: O(limiteMax * n), dove n è la dimensione della lista degli iscritti.

Complessità Spaziale:

1. Spazio Ausiliario (Variabili e Strutture Dati):
   La lista `S` contiene gli iscritti selezionati. Nel caso peggiore, sarà di dimensione `limiteMax`.
   Altre variabili ausiliarie occupano uno spazio costante.
   Complessità Spaziale: O(limiteMax).

In generale, l'algoritmo ha una complessità temporale più significativa rispetto a quella spaziale, e la sua efficienza dipende dal rapporto tra il numero massimo di posti desiderati e la dimensione totale della lista degli iscritti.