UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Corso di Laurea in
Informatica e Comunicazione Digitale
sede di Taranto

CORSO DI
Algoritmi e Strutture Dati

Anno Accademico 2019/2020

Docente:

Prof. Stefano Ferilli

Studente:

Matteo Luceri

Matr.: 677663

Sommario

1. Traccia

Si richiede di modificare il progetto base (base*1911* - Rosa Chiarappa) in modo che vengano aggiunte le funzionalità presenti in quello di Mirco Sternativo, nello specifico, l'inserimento di uno zaino ed una valigia. Inoltre, è previsto che la realizzazione di almeno due strutture di dati vengano rese intercambiabili.

2. Analisi

2.1 Analisi preliminare

"Adventure with Simunav" è un gioco d'avventura testuale in cui il giocatore controlla il comandante dell'astronave "Neutronia", il quale dopo essersi svegliato di soprassalto nella sua cabina di pilotaggio, a causa di una elevata temperatura, deve muoversi all'interno dell'astronave risolvendo problemi/enigmi per salvare sé stesso ed il suo equipaggio.

Lo scopo del giocatore è raggiungere la soluzione del gioco risolvendo una serie di enigmi ed evitando vari ostacoli, in quanto il gioco stesso immerge il giocatore in una trama che si genererà passo dopo passo in base alle sue scelte.

Il tempo a disposizione per la riuscita dell'impresa è limitato, ma saranno presenti diverse ricompense a seguito di determinate azioni all'interno del gioco, che permetteranno di ricevere una quantità di secondi extra.

Il protagonista potrà spostarsi all'interno del gioco digitando dei comandi: Nord(N), Sud(S), Ovest(W), Est(E), Sali(A), Scendi(A); di conseguenza potrà spostarsi all'interno dell'astronave con un navigatore per orientarsi con più facilità nell'intero gioco, che potrà essere richiamato in qualsiasi momento della partita.

2.2 Analisi integrativa

Le funzionalità da implementare nel progetto di base sono:

L'inserimento di oggetti, uno zaino o in una valigia, ossia dei contenitori che permettono all'utente di trasportare determinati oggetti (come casco, tuta, camice o manuale). È possibile, inoltre, decidere se portare questi oggetti con sé o metterli nella valigia o nello zaino.

Lo zaino e la valigia hanno dei limiti di peso trasportabile. La posizione di questi due contenitori sarà fissa: ad ogni nuova partita si troveranno sempre nello stesso luogo, con la possibilità di scegliere se utilizzarli o meno. Non sarà possibile trasportare entrambi gli oggetti contenitori, bisognerà scegliere solo uno dei due.

Nella valigia si possono inserire e prelevare quanti oggetti si vuole, rimanendo nei limiti del peso massimo consentito, invece nello zaino si potranno sì mettere quanti oggetti si vuole rimanendo nei limiti del peso, ma si potrà prendere solo il primo di questi. Ciò comporta che, nel caso in cui si voglia esplorare tutto lo zaino, bisognerà togliere tutti gli oggetti presenti. L'utilizzo dello zaino o della valigia è a discrezione dell'avventuriero che, in caso ce l'abbia, può decidere se trasportare con sé gli oggetti o metterli nel contenitore.

3. Progettazione

3.1 Zaino e Valigia

L'oggetto Valigia è stato pensato come una collezione di oggetti nella quale si ha una visione complessiva, mentre lo zaino è stato pensato come un contenitore di oggetti uno sopra l'altro, in cui è possibile prendere solo quello in cima.
Come già detto in fase di analisi, gli oggetti contenitori hanno dei limiti di peso trasportabile. La Valigia avrà dimensione n, mentre lo Zaino avrà dimensione m con n>m.
Se si possiede un oggetto contenitore, al momento della raccolta di un oggetto, il giocatore potrà decidere se portare con sé l'oggetto o metterlo nel contenitore, perciò verrà posta la seguente domanda:

"Se vuoi portare l'oggetto con te, premi (y);
se vuoi metterlo in valigia, premi (v)"

in caso si sia raccolta la valigia oppure:

"Se vuoi portare l'oggetto con te, premi (y);
se vuoi metterlo nello zaino, premi (z)"

in caso si sia raccolto lo zaino.
In base alla risposta, l'oggetto verrà messo o nell'inventario o nel contenitore.

L'oggetto Valigia verrà realizzato mediante la struttura di dati insieme, in quanto l'oggetto è stato pensato come una collezione di oggetti.
La struttura di dati *insieme* intende riprodurre la nozione matematica di insieme, ossia una collezione (o famiglia) di elementi (detti membri) tutti appartenenti ad uno stesso tipo base (detto dominio).
Ogni stanza d'insieme sarà un elemento dell'insieme delle parti del dominio.
È una struttura omogenea, non-lineare, dinamica, ad accesso diretto e in memoria centrale.
A differenza delle liste gli elementi non sono caratterizzati da una posizione, e dunque non possono apparire più di una volta.
Il numero di elementi di un insieme i (detto cardinalità e denotato da |i|) rappresenta la dimensione dell'insieme.

Solitamente rappresentati graficamente tramite diagrammi di Venn, in matematica gli insiemi possono essere definiti:

- estensionalmente , cioè elencandone tutti i membri,
- intenzionalmente cioè specificando la proprietà caratteristica in base alla quale stabilire se un qualunque dato elemento del dominio e membro dell'insieme o meno.

In informatica ci si riferisce al modo estensionale.

La relazione fondamentale è quella di appartenenza di un elemento x ad un insieme i (x∈i), in base alla quale e poi definita la relazione di inclusione di un insieme i' in un insieme i'' (i^'⊆i^'').

Le operazioni principali fra due insiemi, i' e i'' sono unione ($i' \cup i''$), intersezione ($i' \cap i''$) e differenza ($i' \setminus i''$), ben note in matematica. Un insieme che non ha elementi è detto vuoto e viene indicato con Ø.

Le specifiche: Insieme

- **Specifica sintattica:**

  Tipi: Insieme, Boolean, Tipoelem

  Operatori [Insieme i]:

  creainsieme : ( ) -> Insieme
  insiemevuoto : (Insieme) -> Boolean
  appartiene : (Tipoelem, Insieme) -> Boolean
  inserisci : (Tipoelem, Insieme) -> Insieme
  cancella : (Tipoelem, Insieme) -> Insieme
  unione : (Insieme, Insieme) -> Insieme
  intersezione : (Insieme, Insieme) -> Insieme
  differenza : (Insieme, Insieme) -> Insieme

- **Specifica semantica**:

  Tipi:

  - *Insieme* : famiglia di insiemi – a definizione estensionale – costituiti da elementi di tipo *Tipoelem*.
  - *Boolean:* insieme dei valori di verità

  Operatori [*Insieme i*]:

  creainsieme = i
  Pre: *** *non ha precondizioni perché può essere sempre generata*
  Post: $i = \emptyset, |i| = 0$
  insiemevuoto (i) = b
  Pre: *** *non ha precondizioni perché può essere sempre generata*
  Post: b = TRUE se $i = \emptyset$ altrimenti b =FALSE
  appartiene (x, i) = i'

Pre: *** *non ha precondizioni perché può essere sempre generata*
Post: b = TRUE se $x \in i$ altrimenti b =FALSE

inserisci (x, i) = i'
Pre: $x \notin i$ (OPPURE senza precondizione)
Post: $\mathrm{i}' = i \cup \{x\}$

cancella (x, i) = i'
Pre: $x \in i$ (OPPURE senza precondizione)
Post: $\mathrm{i}' = i \setminus \{x\}$

unione (i, i') = i''
Pre: *** *non ha precondizioni perché può essere sempre generata*
Post: $\mathrm{i}'' = i \cup i'$

intersezione (i, i') = i''
Pre: *** *non ha precondizioni perché può essere sempre generata*
Post: $\mathrm{i}'' = i \cap i'$

differenza (i, i') = i''
Pre: *** *non ha precondizioni perché può essere sempre generata*
Post: $\mathrm{i}'' = i \setminus i'$

L'oggetto Zaino verrà creato mediante la struttura di dati pila.

Una pila è una sequenza di elementi omogenei, dove è possibile aggiungere o cancellare elementi da un estremo (la testa).

È una struttura lineare e dinamica.

Può essere vista come un caso speciale di lista in cui l'ultimo elemento inserito è il primo ad essere rimosso (LIFO). Tale accesso è diretto.

Le specifiche: Pila

- **Specifica sintattica:**

  Tipi: Pila, Boolean, Tipoelem

  Operatori [Lista L]:

  creapila : ( ) -> Pila
  pilavuota : (Pila) -> Boolean
  leggilpila : (Pila) -> Tipoelem
  fuoripila : (Pila) -> Pila
  inpila : (Tipoelem, Pila) -> Pila

- **Specifica semantica**:

  Tipi:

  - *Pila* : l'insieme delle sequenze $P = < a_1, a_2, \dots, a_n >, n \geq 1$, di elementi di tipo *Tipoelem,* gestite con accesso LIFO, dove l'elemento i-esimo ha valore e_i
  - *Boolean:* insieme dei valori di verità

  Operatori [Pila $P = < a_1, a_2, \dots, a_n >, n \geq 1$]:

creapila = P ***EMPTY STACK***

Pre:

*** *non ha precondizioni perché può essere sempre generata*

Post: $P = \Lambda,\ \Lambda =< >$

pilavuota (P) = b ***EMPTY***

Pre: *** *non ha precondizioni perché può essere sempre generata*

Post: b = TRUE se L= Λ

leggilpila (P) = a ***TOP***

Pre: $P =< a_1, a_2, \dots, a_n >, n \geq 1$

Post: $a = a_1$

fuoripila (P) = P′ ***POP***

Pre: $P =< a_1, a_2, \dots, a_n >, n \geq 1$

Post: $P =< a_2, \dots, a_n > \ se\ n > 1, P = \Lambda\ se\ n = 0$

inpila (a,P) = P′ ***PUSH***

Pre: $P =< a_1, a_2, \dots, a_n >, n \geq 0$

Post: $P' =< \boldsymbol{a}, a_1, a_2, \dots, a_n >$

4. Implementazione

Per quanto riguarda gli oggetti Zaino e Valigia e per i frammenti verranno inseriti nuovi vocaboli, all'interno del vocabolario:

- `Inserisci("aiuto", 259, vocabolario)`
- `Inserisci("help", 259, vocabolario)`
- `Inserisci("valigia", 155, vocabolario)`
- `Inserisci("zaino", 156, vocabolario)`
- `Inserisci("raccogli", 258, vocabolario)`

E verranno inseriti i seguenti oggetti:

- `Inserisci(Oggetto("una valigia", 155, 5, 0), oggetti)`
- `Inserisci(Oggetto("uno zaino", 156, 6, 0), oggetti)`

Ogni oggetto sarà costituito dalla sua descrizione, codice, codice del luogo in cui è posto all'inizio del gioco e dal suo peso. Quest'ultimo attributo è stato aggiunto per gestire il peso massimo trasportabile nello zaino e nella valigia. Gli oggetti con peso uguale a zero sono gli oggetti il cui peso è trascurato nel gioco, mentre ad ogni oggetto trasportabile nello zaino o nella valigia è assegnato un peso maggiore di zero.

- L'oggetto Manuale avrà peso 1;
- L'oggetto Tuta avrà peso 2;
- L'oggetto Camice avrà peso 3;
- L'oggetto Casco avrà peso 3;

Tutti i pesi sono espressi in kg.

Verranno aggiunte le seguenti azioni:

- `Inserisci(80155, -2, azioni)` `//azione prendi valigia`
- `Inserisci(80156, -2, azioni)` `//azione prendi zaino`
- `Inserisci(90155, 3, azioni)` `//azione lascia valigia`
- `Inserisci(90156, 3, azioni)` `//azione lascia zaino`
- `Inserisci(100155, 140, azioni)` `//azione guarda valigia`
- `Inserisci(100156, 141, azioni)` `//azione guarda zaino`
- `Inserisci(1550000, 143, azioni)` `//azione inventario valigia`
- `Inserisci(1560000, 144, azioni)` `//azione inventario zaino`
- `Inserisci(2590000, 145, azioni)` `//azione aiuto/help`

E verranno modificate le seguenti azioni:

- `Inserisci(2050, -71, azioni)`
- `Inserisci(2051, -71, azioni)`
- `Inserisci(2052, -71, azioni)`

in:

- `Inserisci (200050, -142)`
- `Inserisci (200051, -142)`
- `Inserisci (200052, -142)`

Perciò ogni azione sarà costituita dal codice del comando e dal codice dell'azione da eseguire per quel comando.

Per rappresentare il codice del comando saranno necessarie 6 cifre del tipo LLVVOO. Le prime due indicheranno il codice del luogo (LL) in cui il comando è impartito, le due intermedie indicheranno il codice del verbo (VV) e le ultime due indicheranno il codice dell'oggetto (OO).

Azioni	LL	VV	OO	CODICE
Prendi valigia	00	08	15	-2
Prendi zaino	00	08	16	-2
Lascia valigia	00	09	15	3
Lascia zaino	00	09	16	3
Guarda valigia	00	10	15	69
Guarda zaino	00	10	16	70
Inventario valigia	00	15	00	72
Inventario zaino	00	16	00	73
Aiuto/help	00	29	00	74
Indossa/metti casco	00	20	50	-71
Indossa/metti tuta	00	20	51	-71
Indossa/metti camice	00	20	52	-71

Per quelle azioni in cui si prende un oggetto verrà usato il codice azione generale per "prendi oggetto", per quelle in cui si lascia un oggetto verrà usato il codice azione generale per "lascia oggetto".

I file del progetto base (Rosa Chiarappa) che hanno subito modifiche sono i seguenti:

- `Astro.h`
- `Astro.cpp`
- `Gioco.h`
- `Gioco.cpp`
- `Mappa.h`
- `Interfaccia.cpp`
- `Oggetti.h`
- `Oggetti.cpp`
- `Oggetto.h`
- `Oggetto.cpp`

Mentre sono stati aggiunti i seguenti file:

- `Zaino.h`
- `Valigia.h`

4. Realizzazione

4.1.1 Astro.h

Sono state aggiunte le seguenti funzioni:

```
// Modifiche_ML
bool indossa_specifiche();
bool indosso_specifiche();
```

4.1.2 Astro.cpp

Sono stati aggiunti i seguenti vocaboli:

```
vocabolario.inserisci("valigia", 155);
vocabolario.inserisci("zaino", 156);
vocabolario.inserisci("raccogli", 258);
vocabolario.inserisci("aiuto", 259);
vocabolario.inserisci("help", 259);
```

Sono stati aggiunte le seguenti azioni:

```
// Modifiche Matteo_Luceri(ex_Sternativo)

azioni.inserisci(80155, -2); //Azione prendi valigia
azioni.inserisci(80156, -2); //Azione prendi zaino
azioni.inserisci(90155, 3); //Azione lascia valigia
azioni.inserisci(90156, 3); //Azione lascia zaino
azioni.inserisci(100155, 140); //Azione guarda valigia
azioni.inserisci(100156, 141); //Azione guarda zaino
azioni.inserisci(1550000, 143); //Azione inventario valigia
azioni.inserisci(1560000, 144); //Azione inventario zaino
azioni.inserisci(2590000, 145); //Azione aiuto/help
```

Sono state modificate le seguenti azioni:

```
azioni.inserisci(200050, -142); //Modifica ML
azioni.inserisci(200051, -142); //Modifica ML
azioni.inserisci(200052, -142); //Modifica ML
```

In tutti gli oggetti è stato aggiunto un attributo "peso" e sono stati aggiunti:

```
//_Modifiche_Matteo_Luceri
oggetti.inserisci(Oggetto("una valigia", 155, 6, 0));
oggetti.inserisci(Oggetto("uno zaino", 156, 6, 0));
```

Sono state inizializzate le seguenti variabili:

```
//Modifiche ML
aperto=false;
n_oggettiZ = 0;
peso_MaxZ = 5;
n_oggettiZV = 0;
peso_MaxZV = 8;
el1 = 0;
el2 = 0;
el3 = 0;
el4 = 0;
el5 = 0;
el6 = 0;
//fine modifiche
```

Sono state modificate le seguenti funzioni:

Le funzioni preso_specifiche e prendi_specifiche sono state "spostate" (con le dovute modifiche elencate di seguito) in indosso_specifiche e indossa_specifiche e le prime due sono state modificate in questo modo:

```
if(og == 143 && oggetti.get_oggetto(144).get_luogo() == 0){
        // controllo che l'oggetto sia la valigia e che nell'inventario ci sia lo zaino
    interfaccia.scrivi("Lascia prima lo zaino.");
}
else if(og == 144 && oggetti.get_oggetto(143).get_luogo() == 0){
    // controllo che l'oggetto sia lo zaino e che nell'inventario ci sia la valigia
    interfaccia.scrivi("Lascia prima la valigia.");
}
```

```
bool Astro::preso_specifiche() {
    //Modifiche Mirco Sternativo
    bool avvisato = true;

    if ((og == 11 || og == 4 || og == 23 || og == 25) && (oggetti.get_oggetto(143).get_luogo() == 0)){
        if(oggetti.get_oggetto(og).get_peso() > peso_MaxZV && og != 25){
            // controllo che il peso dell'oggetto sia maggiore allo spazio disponibile
            // nella valigia e che l'oggetto sia diverso dal manuale
            oggetti.set_luogo(og,luogo_attuale);
            interfaccia.scrivi("La valigia e' troppo piena...");
            interfaccia.scrivi("(Suggerimento: togli qualcosa dalla valigia.)");
        } else if(oggetti.get_oggetto(og).get_peso() > peso_MaxZV && og == 25){
            oggetti.set_luogo(og,0);
            interfaccia.scrivi("La valigia e' troppo piena...");
            interfaccia.scrivi("Lo tengo in mano, e' troppo importante.");
            interfaccia.scrivi("(Suggerimento: controlla nell'inventario)");
        } else{
            interfaccia.scrivi("Ora e' in valigia.");
            ins.inserisci(oggetti.get_oggetto(og).get_codice());
            n_oggettiZV++;
            peso_MaxZV -= oggetti.get_oggetto(og).get_peso();}
    } else if((og == 11 || og == 4 || og == 23 || og == 25) &&
            (oggetti.get_oggetto(144).get_luogo() == 0)){
        if(oggetti.get_oggetto(og).get_peso() > peso_MaxZ && og != 25){
            // controllo che il peso dell'oggetto sia maggiore allo spazio
            // disponibile nella valigia e che l'oggetto sia diverso dal manuale
            oggetti.set_luogo(og,luogo_attuale);
            interfaccia.scrivi("Lo zaino e' troppo pieno...");
            interfaccia.scrivi("(Suggerimento: togli qualcosa dallo zaino.)");
        } else if(oggetti.get_oggetto(og).get_peso() > peso_MaxZ && og == 25){
            oggetti.set_luogo(og,0);
            interfaccia.scrivi("Lo zaino e' troppo pieno...");
            interfaccia.scrivi("Lo tengo in mano, e' troppo importante.");
            interfaccia.scrivi("(Suggerimento: controlla nell'inventario)");
        }else{
            interfaccia.scrivi("Ora e' nello zaino.");
            z.inpila(oggetti.get_oggetto(og).get_codice()); //*
            n_oggettiZ++;
            peso_MaxZ -= oggetti.get_oggetto(og).get_peso();}
    } else
        avvisato = false;
    if(og == 23){
        stringa_risposta = "l'hai preso.";      //Modifica PMF(storia)
        storia_gioco.insStoria(stringa_comando , stringa_risposta); //Modifica PMF(storia)
        bacheca.CancellaMessaggioGioco
        ("*stai assorbendo troppe radiazioni! non hai la giusta protezione!");
    }else{
        avvisato = false;}

    return avvisato;
    //Fine Modifiche
```

Sono state aggiunte le seguenti azioni:

```
void Astro::azione_140(){        //Azione guarda valigia
    interfaccia.scrivi("E' la valigia del secondo pilota.");
    interfaccia.scrivi("Sembra molto capiente.");
}
void Astro::azione_141(){        //Azione guarda zaino
    interfaccia.scrivi("E' il tuo zaino.");
    interfaccia.scrivi("Puo' esserti utile per trasportare oggetti.");
}


void Astro::azione_142(){        //Azione indossa/metti
    if (oggetti.get_oggetto(og).get_luogo() == 0)
      // controllo che l'oggetto si trovi nell'inventario
        interfaccia.scrivi("- Gia' fatto.");
    else if (oggetti.get_oggetto(og).get_luogo() < 0)
      // controllo che l'oggetto non sia indossabile
        interfaccia.scrivi("- Non e' possibile.");
    else if (!indossa_specifiche()) {
        if(oggetti.get_oggetto(144).get_luogo() == 0){
            if (oggetti.get_oggetto(4).get_luogo() == 20 ||
                oggetti.get_oggetto(11).get_luogo() == 20 ||
                oggetti.get_oggetto(22).get_luogo() == 20){
                if(og == 4 || og == 11 || og == 23){
                    int c;
                    c=z.leggipila();
                    if(og == oggetti.get_zaino2(c)){
                   // controllo che l'oggetto sia proprio il primo oggetto nello zaino
                        oggetti.set_luogo(og,0);
                        z.fuoripila();
                        n_oggettiZ--;
                        peso_MaxZ += oggetti.get_oggetto(og).get_peso();
                    }
                    else if(oggetti.get_oggetto(og).get_luogo() == luogo_attuale){
                   // controllo che l'oggetto si trovi nel luogo attuale
                        oggetti.set_luogo(og,0);
                    }
                    else{
                        interfaccia.scrivi("Devi prima lasciare: ");
                        oggetti.get_zaino(c);
                    }
                }else{
                    oggetti.set_luogo(og,0);
                    interfaccia.scrivi("Fatto.");
                }
            }else{
                oggetti.set_luogo(og,0);
            }
        }
```

```
        else if(oggetti.get_oggetto(143).get_luogo() == 0){
            if (oggetti.get_oggetto(4).get_luogo() == 20 ||
             oggetti.get_oggetto(11).get_luogo() == 20 ||
             oggetti.get_oggetto(23).get_luogo() == 20){
                if(og == 4 || og == 11 || og == 23){
                    int c;
                    c=og;
                    if(ins.Appartiene(oggetti.get_valigia(c))){
                    // controllo che l'oggetto sia presente nella valigia
                        oggetti.set_luogo(og,0);
                        ins.Cancella(oggetti.get_valigia(c));
                        n_oggettiZV--;
                        peso_MaxZV += oggetti.get_oggetto(og).get_peso();
                    }
                    else if(!ins.Appartiene(oggetti.get_valigia(c)))
                    //controllo che l'oggetto non sia presente nella valigia
                        interfaccia.scrivi("- Non ce l'hai in valigia.");
                    else if(oggetti.get_oggetto(og).get_luogo() == luogo_attuale){
                    //controllo che l'oggetto si trovi nel luogo attuale
                        oggetti.set_luogo(og,0);
                    }
                }
                else{
                    oggetti.set_luogo(og,0);
                    interfaccia.scrivi("Fatto.");
                }
            }
            else{
                oggetti.set_luogo(og,0);
                }
        }
        else{
            if(og != 0 && oggetti.get_oggetto(og).get_luogo() != 20){
             // controllo che l'oggetto non si trovi in un altro luogo e che l'oggetto
    non si trovi nello zaino o nella valigia
                oggetti.set_luogo(og,0);
            }
            else
                interfaccia.scrivi("- Sta nello zaino o nella valigia, e non li hai.");
        }
        if (!indosso_specifiche()){ // faccio alcuni controlli in indosso_specifiche
            if(oggetti.get_oggetto(og).get_luogo() == 0)
             //controllo che l'oggetto si trovi nell'inventario
                interfaccia.scrivi("Fatto.");
        }
    }
}
```

```
void Astro::azione_143(){        //Azione inventario valigia
    if(oggetti.get_oggetto(143).get_luogo() == 0){
        if (!ins.InsiemeVuoto()){
          interfaccia.scrivi("Inventario Valigia");
          interfaccia.scrivi("\nVedo: ");
          if(ins.Appartiene(50)) // controllo che il casco sia presente nella valigia
                oggetti.get_zaino(50);
          if(ins.Appartiene(51)) // controllo che la tua sia presente nella valigia
                oggetti.get_zaino(51);
          if(ins.Appartiene(52)) // controllo che il camice sia presente nella valigia
                oggetti.get_zaino(52);
          if(ins.Appartiene(55)) // controllo che il manuale sia presente nella valigia
                oggetti.get_zaino(55);
          cout << "\nTotale Oggetti nella Valigia: " << n_oggettiZV << endl;
          cout << "Spazio disponibile: "<< peso_MaxZV << " su 8 kg."<< endl;
        }
        else
          interfaccia.scrivi("E' vuota.");
    }
    else
        interfaccia.scrivi("Non ce l'hai.");
}
```

```
void Astro::azione_144(){        //Azione inventario zaino
    if(oggetti.get_oggetto(144).get_luogo() == 0){
      // controllo che l'oggetto zaino si trovi nell'inventario
        if (!z.pilavuota()) { // controllo che l'oggetto zaino non sia vuoto //*
            interfaccia.scrivi("Inventario Zaino");
            interfaccia.scrivi("\nVedo in cima: ");
            int c;
            c=z.leggipila(); //*
            oggetti.get_zaino(c);
            cout << "\nTotale Oggetti nello Zaino: " << n_oggettiZ << endl;
            cout << "Spazio disponibile: "<< peso_MaxZ << " su 5 kg."<< endl;
        }
        else
            interfaccia.scrivi("E' vuoto.");
    }
    else
        interfaccia.scrivi("Non lo hai.");
}
```

```
void Astro::azione_145(){        //Azione aiuto/help
    interfaccia.scrivi("\nCOMANDI DI GIOCO:");
    interfaccia.scrivi("\nDirezioni: ");
    interfaccia.scrivi("- n/nord: per muoverti in avanti;");
    interfaccia.scrivi("- s/sud: per muoverti indietro;");
    interfaccia.scrivi("- e/est: per muoverti a destra;");
    interfaccia.scrivi("- w/o/ovest: per muoverti a sinistra;");
    interfaccia.scrivi("- a/alto/sali: per salire ad un piano superiore;");
    interfaccia.scrivi("- b/basso/scendi: per scendere ad un piano inferiore;");
    interfaccia.scrivi("\nAzioni: ");
    interfaccia.scrivi("- prendi/raccogli: per trasportare un oggetto in mano
                        o con se(zaino/valigia);");
    interfaccia.scrivi("- indossa/metti: per indossare un oggetto(es. casco);");
    interfaccia.scrivi("- guarda: per guardare ed ottenere informazioni su un oggetto
                        (es.tuta);");
    interfaccia.scrivi("- lascia/togli/leva: per lasciare o togliersi gli oggetti
                        trasportati;");
    interfaccia.scrivi("- apri: per aprire un oggetto fisso(es. armadietto);");
    interfaccia.scrivi("- leggi: per leggere una scritta(es.cartello);");
    interfaccia.scrivi("- spingi/tira: per spingere o tirare un oggetto
                        fisso(es.leva);");
    interfaccia.scrivi("- premi/schiaccia: per premere un oggetto fisso(es.pulsante);");
    interfaccia.scrivi("- inventario/cosa: per accedere all'inventario degli oggetti
                        trasportati;");
    interfaccia.scrivi("- zaino: per accedere agli oggetti trasportati nello zaino;");
    interfaccia.scrivi("- valigia: per accedere agli oggetti trasportati nella
                        valigia;");
    interfaccia.scrivi("- save/load: per salvare o caricare la partita;");
    interfaccia.scrivi("- mappa/navigatore: per avviare il navigatore SIMUNAV;\n"); }
```

`bool Astro::esegui_specifiche(int a, Mappa &M)` -> sono stati aggiunti i seguenti case:

```
    case 140:
            azione_140();
            break;

    case 141:
            azione_141();
            break;

    case 142:
            azione_142();
            break;

    case 143:
            azione_143();
            break;

    case 144:
            azione_144();
            break;

    case 145:
            azione_145();
            break;
```

4.1.3 Gioco.h

È stata aggiunta l'include per permettere l'implementazione della valigia:

```
#include "Valigia.h"
```

Sono state aggiunte le seguenti variabili:

```
Pila<int> p;
Pila<int> z;
int n_oggettiZ;      //Numero di oggetti nello zaino
int peso_MaxZ;       //Peso massimo trasportabile nello zaino

Insieme<int> ins;
int n_oggettiZV;     //Numero di oggetti nella valigia
int peso_MaxZV;      //Peso massimo trasportabile nella valigia

//el1=primo elemento zaino; el2=secondo elemento zaino;
//el3 [..] el6 elementi valigia
int el1;
int el2;
int el3;
int el4;
int el5;
int el6;
bool salva;          //Boooleano usato nella save e nella load
string risposta1;    //risposta al frammento
bool controllo1;
bool controllo2;
Lista<int> L
```

4.1.4 Gioco.cpp

Sono state modificate le seguenti funzioni:

```
void Gioco::prendi()
{
    //modifica Gallone Gianmarco -  Riadattamento metodo prendi per tasca, zaino e portafoglio.
    bool fatto = false;
    bool cpz= true ;
    riferimento_tasca = tasca.primolista();
    riferimento_zaino = zaino_frigo.primolista();
    bool trovato = false;
    int check = 0;


    if(og == 11 || og == 4 || og == 23 || og == 25)
    //controllo che gli oggetti che possono andare in valigia o nello zaino siano
    // o il casco, o la tuta, o il camice o il manuale
    {
        if (oggetti.get_oggetto(og).get_luogo() == 0)
        //controllo che l'oggetto non sia stato già preso
        {
            interfaccia.scrivi("- Gia' fatto.");
            stringa_risposta = "non e' stato possibile perche' avevi gia' quell'oggetto.";
            //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);
            //Modifica PMF(storia)
        }
        else if(oggetti.get_oggetto(og).get_luogo() == 20)
        //controllo che l'oggetto non sia già nela valigia o nello zaino
        {
            interfaccia.scrivi("- Sta gia' nello zaino o nella valigia.");
        }
        else if (oggetti.get_oggetto(og).get_luogo() < 0)
        //controllo che l'oggetto si possa prendere del tutto
        {
            stringa_risposta = "non e' stato possibile.";
            //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);
            //Modifica PMF(storia)
            interfaccia.scrivi("- Non e' possibile.");
        }
        else if (!prendi_specifiche())
        {
            controllo2 = false;
            controllo1 = false;
```

```
if(oggetti.get_oggetto(143).get_luogo() == 0)
// controllo se la valigia si trova nell'inventario
{
    if (og == 21 || og == 24 || og==49)
    {
        oggetti.set_luogo(og,0);
    }
    else
    {
        do
        {
            interfaccia.scrivi(
            "Se vuoi portare l'oggetto con te, premi(y);se vuoi metterlo in valigia, premi(v)");


            cin >> rispostal;
            if(rispostal == "y" || rispostal == "Y")
            {
                oggetti.set_luogo(og,0);
                interfaccia.scrivi("Ora e' nell'inventario.");
                controllo1 = true;
                controllo2 = true;
            }
            else if(rispostal == "v" || rispostal == "V")
            {
                oggetti.set_luogo(og,20);
                controllo2 = true;
            }
            else
            {
                controllo2 = false;
            }
        }
        while(controllo2 == false);
    }
}
else if(oggetti.get_oggetto(144).get_luogo() == 0)
{
    if (og == 21 || og == 24 || og==49)
    {
        oggetti.set_luogo(og,0);
    }
    else
    {
```

```
            do
            {
                interfaccia.scrivi(
                "Se vuoi portare l'oggetto con te, premi(y); se vuoi metterlo nello zaino, premi(z)");
                cin >> rispostal;
                if(rispostal == "y" || rispostal == "Y")
                {
                    oggetti.set_luogo(og,0);
                    interfaccia.scrivi("Ora e' nell'inventario.");
                    controllo1 = true;
                    controllo2 = true;
                }
                else if(rispostal == "z" || rispostal == "Z")
                {

                    oggetti.set_luogo(og,20);
                    controllo2 = true;
                }
                else
                {
                    controllo2 = false;
                }
            }
            while(controllo2 == false);
        }
    }
    else
    {
        oggetti.set_luogo(og,0);
    }

    if(!controllo1 && !preso_specifiche())
    {
        interfaccia.scrivi("Fatto.");
        stringa_risposta = "l'hai preso.";
        //Modifica PMF(storia)
        storia_gioco.insStoria(stringa_comando, stringa_risposta);
        //Modifica PMF(storia)
    }

    bacheca.CancellaMessaggioGioco(
    "*non uscire fuori dall'astronave se non hai l'equipaggiamento da astronauta");
  }
}
```

```
else if(!prendi_specifiche())
{
    if (oggetti.get_oggetto(og).get_luogo() == 0)
    {
        interfaccia.scrivi("- Gia' fatto.");
        stringa_risposta = "non e' stato possibile perche' avevi gia' quell'oggetto.";
        //Modifica PMF(storia)
        storia_gioco.insStoria(stringa_comando, stringa_risposta);
    }
    else if (oggetti.get_oggetto(og).get_luogo() < 0)
    {
        stringa_risposta = "non e' stato possibile.";
        //Modifica PMF(storia)
        storia_gioco.insStoria(stringa_comando, stringa_risposta);
        //Modifica PMF(storia)
        interfaccia.scrivi("- Non e' possibile.");
    }
    else if ((og != 27) && (og != 36) && (og != 37) && (og != 38) && (og != 39) &&
            (og != 40) && (og != 41) && (og != 42) && (og != 43) && (og != 44))
    {
        if(!preso_specifiche())
        {
            oggetti.set_luogo(og,0);
            if (og == 26)
            {
                interfaccia.scrivi("Fatto : l'hai addosso ! ");
                stringa_risposta = "l'hai preso.";
                    //Modifica PMF(storia)
                storia_gioco.insStoria(stringa_comando, stringa_risposta);
                //Modifica PMF(storia
            }
            else
            {
                interfaccia.scrivi("Fatto.");
                stringa_risposta = "l'hai preso.";
                //Modifica PMF(storia)
                storia_gioco.insStoria(stringa_comando, stringa_risposta);
                check++;
            }
        }
    }//Gestione portafogli
    else if ( og == 27|| (og >=36 && og <= 44))
    {
```

```
    bool port = false;
    port = portafoglio.hai_Portafoglio(oggetti);
    if (port &&(og>=36 && og <=44) )
    {
        if(!fatto)
            portafoglio.Prendi_Banconota(oggetti,og,interfaccia);
        oggetti.set_luogo(og,0);
    }
    else if(!port && (og>=36 && og <=44))
        interfaccia.scrivi(
        " Non e' Possibile prendere denaro se non hai gia' preso il portafoglio !" );
    else if (!port &&(og==27))
        interfaccia.scrivi(
        "Non e' Possibile prendere la carta di credito  se non hai gia' preso il portafoglio !") ;
    else if (port &&(og==27))
    {
        oggetti.set_luogo(og,0);
        interfaccia.scrivi("Presa! L'hai nel portafoglio");
    }
}
else if (og == 49 || og == 118 || og == 73)
//non puo prendere: motorino
{
    interfaccia.scrivi ("Quest'oggetto non e' tuo. Per averlo dovresti rubarlo!");
    //  chiave auto
    interfaccia.a_capo();
    //  buono pasto giallo
    oggetti.set_rubato(og,false);
}
else if(og == 74)        // Modifica documento d'identità //
{
    oggetti.set_luogo(og, 0);
    tasca.inslista(og, riferimento_tasca);
}
else if (og >=77 && og <= 79)
{
    oggetti.set_luogo(og, 0);
    tasca.inslista(og,riferimento_tasca);
    interfaccia.scrivi("Inserito nella tasca!");
}
else if (og >= 67 && og <=76)
{
    if (tasca.listavuota())
    {
        interfaccia.scrivi("Non hai buoni pasto per acquistarlo!");
        cpz=false;
    }
```

```
            else
            {
                interfaccia.scrivi("Dovresti comprare le pietanze, non prenderle...");
                cpz=false;
            }
        }
        //INIZIO modifiche PALAGIANO MARCELLO
        else if(oggetti.get_oggetto(117).get_luogo() == 0)
        {
            interfaccia.scrivi("** Per prendere oggetti devi lasciare l'autobus! **");
            cpz=false;
        }
        else if(oggetti.get_oggetto(119).get_luogo() == 0)
        {
            interfaccia.scrivi("** Per prendere oggetti devi lasciare l'automobile! **");
            cpz=false;
        }
        //FINE modifiche PALAGIANO MARCELLO
        else
            oggetti.set_luogo(og,0);
        //Fine Modifica Gallone
        // MANCA NELLE VERSIONI INTERMEDIE
        if (!preso_specifiche() && (cpz) && check == 0)
        {
            oggetti.set_rubato(og,false); // modifica CHIARAPPA ROSA
            interfaccia.scrivi("Fatto.");
            interfaccia.a_capo();
        }
    }
    while (!tasca.finelista(riferimento_tasca))
    {
        if (oggetti.get_oggetto(og).get_codice() ==
            oggetti.get_oggetto(tasca.leggilista(riferimento_tasca)).get_codice())
            trovato = true;
        riferimento_tasca = tasca.succlista(riferimento_tasca);
    }


    while (!zaino_frigo.finelista(riferimento_zaino))
    {
        riferimento_zaino = zaino_frigo.succlista(riferimento_zaino);
        oggetti.set_rubato(og,false);
    }
}
```

```
void Gioco::lascia()
{
    riferimento_tasca = tasca.primolista();
    riferimento_zaino = zaino_frigo.primolista();
    bool trovato = false;
    if (og != 44 && (luogo_attuale != 21 && luogo_attuale != 22))
    {
        if (og == 0 || oggetti.get_oggetto(og).get_luogo() != 0)
            interfaccia.scrivi("- Non ce l'hai.");
        else if (!lascia_specifiche())
        {
            if (og >= 77 && og <= 79)
                while (!tasca.finelista(riferimento_tasca) && trovato == false)
                {
                    if(oggetti.get_oggetto(tasca.leggilista(riferimento_tasca)).get_codice() ==
                        oggetti.get_oggetto(og).get_codice())
                    {
                        tasca.canclista(riferimento_tasca);
                        trovato = true;
                    }
                    else
                        riferimento_tasca = tasca.succlista(riferimento_tasca);
                }
            if (og >= 67 && og <= 76)
                while (!zaino_frigo.finelista(riferimento_zaino) && trovato == false)
                {
                    if(oggetti.get_oggetto(zaino_frigo.leggilista(riferimento_zaino)).get_codice() ==
                        oggetti.get_oggetto(og).get_codice())
                    {
                        zaino_frigo.canclista(riferimento_zaino);
                        trovato = true;
                    }
                    else
                        riferimento_zaino = zaino_frigo.succlista(riferimento_zaino);
                }


            if (oggetti.get_oggetto(og).get_rubato()==true)   //modifica ROSA CHIARAPPA (setto a falso
                                                              // gli oggetti rubati lasciati)
            {
                oggetti.set_rubato(og,false);
            }



            oggetti.set_luogo(og,luogo_attuale);
            interfaccia.scrivi("Fatto.");
        }
    }
    else if (luogo_attuale != 12)    //Modifica PMF(ufficio)
    {
```

```
if(og == 0 && oggetti.get_oggetto(og).get_luogo() != 0 && oggetti.get_oggetto(og).get_luogo() != 20)
// controllo che l'oggetto si trovi in un altro luogo e che l'oggetto non si trovi nè nell'inventario,
// nè nella valigia, nè nello zaino
{
    interfaccia.scrivi("- Non ce l'hai e qui non c'e'.");
    stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";//Modifica PMF(storia)
    storia_gioco.insStoria(stringa_comando, stringa_risposta);  //Modifica PMF(storia)
}

else if (!lascia_specifiche())
{
    // faccio alcuni controlli in lascia specifiche
    if(oggetti.get_oggetto(144).get_luogo() == 0)
    {
        // controllo che lo zaino si trovi nell'inventario
        if(og==49) //se lascio il motorino
            salito=false;

        if (oggetti.get_oggetto(4).get_luogo() == 20 || oggetti.get_oggetto(11).get_luogo() == 20 ||
             oggetti.get_oggetto(23).get_luogo() == 20 || oggetti.get_oggetto(25).get_luogo() == 20)
        {
            // controllo che uno tra:tuta, casco, camice e manuale si trovi nello zaino
            if(og == 4 || og == 11 || og == 23 || og == 25)
            {
                // controllo che l'oggetto sia proprio uno tra: tuta, casco, camice e manuale
                if(oggetti.get_oggetto(og).get_luogo()== 0)
                {
                    // controllo che l'oggetto si trovi nell'inventario
                    oggetti.set_luogo(og,luogo_attuale);
                    if(og==49)      //se lascio il motorino
                        salito=false;
                    interfaccia.scrivi("Fatto.");
                    stringa_risposta = "hai lasciato l'oggetto.";              //Modifica PMF(storia)
                    storia_gioco.insStoria(stringa_comando, stringa_risposta);  //Modifica PMF(storia)
                }
                else
                {
                    // controllo che l'oggetto si trovi nello zaino
                    int c,d;
                    c=z.leggipila(); //*
                    d=og;
                    if(og == oggetti.get_zaino2(c))
                    // controllo che l'oggetto sia il primo visibile nello zaino
                    {
                        oggetti.set_luogo(og,luogo_attuale);
                        z.fuoripila(); //*
                        n_oggettiZ--;
                        peso_MaxZ += oggetti.get_oggetto(og).get_peso();
                        interfaccia.scrivi("Fatto.");
                    }
                    else if(ins.appartiene(oggetti.get_valigia(d)))
                    // controllo che l'oggetto si trovi nella valigia
                    {
```

```
                {
                    interfaccia.scrivi("- Non ce l'hai addosso e non sta nello zaino..");
                    interfaccia.scrivi(" (Suggerimento: controlla nella valigia)");
                    stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";
                    //Modifica PMF(storia)
                    storia_gioco.insStoria(stringa_comando, stringa_risposta);
                    //Modifica PMF(storia)
                }
                else if (!z.pilavuota())
                // controllo che lo zaino non sia vuoto
                {
                    interfaccia.scrivi("Devi prima lasciare: ");
                    oggetti.get_zaino(c);
                }
            }
        }
        else
        {

            oggetti.set_luogo(og,luogo_attuale);
            interfaccia.scrivi("Fatto.");
            stringa_risposta = "hai lasciato l'oggetto.";                //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);   //Modifica PMF(storia)
        }
    }
    else
    {

        if(oggetti.get_oggetto(og).get_luogo() != 0)
        // controllo che l'oggetto non si trovi nell'inventario
        {
            interfaccia.scrivi("- Non ce l'hai.");
            stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";
            //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);
            //Modifica PMF(storia)
        }
        else
        // controllo che l'oggetto si trovi nell'inventario
        {

            oggetti.set_luogo(og,luogo_attuale);
            interfaccia.scrivi("Fatto.");
            stringa_risposta = "hai lasciato l'oggetto.";                //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);   //Modifica PMF(storia)
        }
    }
}
else if(oggetti.get_oggetto(143).get_luogo() == 0)
// controllo che la valigia si trovi nell'inventario
{
    if(og==39)                                          //se lascio il motorino
        salito=false;
    if (oggetti.get_oggetto(4).get_luogo() == 20 || oggetti.get_oggetto(11).get_luogo() == 20 ||
        oggetti.get_oggetto(23).get_luogo() == 20 || oggetti.get_oggetto(25).get_luogo() == 20)
        // controllo che uno tra: tuta, casco, camice e manuale si trovi nella valigia
    {
```

```
{
    if(og == 4 || og == 11 || og == 23 || og == 25)
    // controllo che l'oggetto sia proprio uno tra: tuta, casco, camice e manuale
    {
        int c;
        c=og;
        if(oggetti.get_oggetto(og).get_luogo()== 0)
        //controllo che l'oggetto si trovi nell'inventario
        {
            oggetti.set_luogo(og,luogo_attuale);
            if(og==49)                //se lascio il motorino
                salito=false;
            interfaccia.scrivi("Fatto.");
            stringa_risposta = "hai lasciato l'oggetto.";                //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);   //Modifica PMF(storia)
        }
        else if(!ins.appartiene(oggetti.get_valigia(c)))
        // controllo che l'oggetto non si trovi nella valigia
        {
            if((oggetti.get_valigia(c))==(z.leggipila()))
            // controllo che l'oggetto sia il primo visibile dello zaino
            {
                interfaccia.scrivi("- Non ce l'hai in valigia.");
                stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";
                //Modifica PMF(storia)
                storia_gioco.insStoria(stringa_comando, stringa_risposta);
                //Modifica PMF(storia)
                interfaccia.scrivi(" (Suggerimento: controlla nello zaino)");
            }
            else if((oggetti.get_valigia(c))!=(z.leggipila()))
            // controllo che l'oggetto non sia il primo visibile nello zaino
                interfaccia.scrivi("- Non ce l'hai in valigia.");
                stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";
                //Modifica PMF(storia)
                storia_gioco.insStoria(stringa_comando, stringa_risposta);
                //Modifica PMF(storia)
        }
        else if(ins.appartiene(oggetti.get_valigia(c)))
        // controllo che l'oggetto si trovi nella valigia
        {
            oggetti.set_luogo(og,luogo_attuale);
            ins.cancella(oggetti.get_valigia(c));
            n_oggettiZV--;
            peso_MaxZV += oggetti.get_oggetto(og).get_peso();
            interfaccia.scrivi("Fatto.");
            stringa_risposta = "hai lasciato l'oggetto.";                //Modifica PMF(storia)
            storia_gioco.insStoria(stringa_comando, stringa_risposta);   //Modifica PMF(storia)
        }
        else
            interfaccia.scrivi("- Non ce l'hai in valigia.");
        stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";
        //Modifica PMF(storia)
        storia_gioco.insStoria(stringa_comando, stringa_risposta);   //Modifica PMF(storia)
    }
    else
    // dovrebbe riferirsi a chiave e secondo se stanno nell'inventario
    {
```

```
                {
                    oggetti.set_luogo(og,luogo_attuale);
                    interfaccia.scrivi("Fatto.");
                    stringa_risposta = "hai lasciato l'oggetto.";               //Modifica PMF(storia)
                    storia_gioco.insStoria(stringa_comando, stringa_risposta);  //Modifica PMF(storia)
                }
            }
            else
            {
                if(oggetti.get_oggetto(og).get_luogo() != 0)
                // controllo che l'oggetto non si trovi nell'inventario
                {
                    interfaccia.scrivi("- Non ce l'hai.");
                    stringa_risposta = "non e' stato possibile perche' non avevi quell'oggetto.";
                    //Modifica PMF(storia)
                    storia_gioco.insStoria(stringa_comando, stringa_risposta);
                    //Modifica PMF(storia)
                }
                else
                {
                    oggetti.set_luogo(og,luogo_attuale);
                    interfaccia.scrivi("Fatto.");
                    stringa_risposta = "hai lasciato l'oggetto.";               //Modifica PMF(storia)
                    storia_gioco.insStoria(stringa_comando, stringa_risposta);  //Modifica PMF(storia)
                }
            }
        }
        else
        {
            if(oggetti.get_oggetto(og).get_luogo() == 0)
            // controllo che l'oggetto si trovi nell'inventario
            {
                oggetti.set_luogo(og,luogo_attuale);
                if(og==39)                                      //se lascio il motorino
                    salito=false;
                interfaccia.scrivi("Fatto.");
                stringa_risposta = "hai lasciato l'oggetto.";               //Modifica PMF(storia)
                storia_gioco.insStoria(stringa_comando, stringa_risposta);  //Modifica PMF(storia)
            }
            /*else
                interfaccia.scrivi("- Non ce l'hai.");*/
        }
    }
    else if ((luogo_attuale == 7 || luogo_attuale >= 9) && (oggetti.get_oggetto(4).get_luogo() != 0 ||
            oggetti.get_oggetto(11).get_luogo() != 0))
    {
        // controllo che il luogo attuale sia o 7 o qualsiasi luogo pari o superiore a 9 e che la tuta o
        // il casco non si trovino nell'inventario
        interfaccia.scrivi("\nAaaagh!!!");
        morto();
    }
    if (og == 67)
    {
        interfaccia.scrivi("Non puoi compiere questa azione!");
    }

} //Fine modifica
```

```
    //Modifica PMF(ufficio)
    else
    {
        interfaccia.scrivi("Non puoi lasciarlo nell' ufficio.");
    }
    //Modifica PMF: fin qui.

}
```

```
void Gioco::save()
{
    int i;

    interfaccia.scrivi("Salvataggio partita...");
    ofstream file(fStringa.c_str(), ios::out);
    for (i = 1; i <= oggetti.get_n_oggettiZ(); i++)
    {
        file << oggetti.get_oggetto(i).get_luogo() << '\n';
    }
    file << luogo_attuale << '\n';
    file << tempo << '\n';
    file << passo_soluzione << '\n';

    //Modifiche ML
    file << n_oggettiZ << '\n';
    file << peso_MaxZ << '\n';

    if(!(z.pilavuota()))              // controllo che lo zaino non sia vuoto
    {
        el1 = z.leggipila();
        z.fuoripila();
        if(!(z.pilavuota()))         // controllo che lo zaino non sia vuoto
        {
            el2 = z.leggipila();
            z.fuoripila();
            z.inpila(el2);
        }
        z.inpila(el1);
    }
    file << el1 << '\n';
    file << el2 << '\n';

    file << n_oggettiZV << '\n';
    file << peso_MaxZV << '\n';
    if(!(ins.insiemevuoto()))        //controllo che la valigia non sia vuota
    {
        if(ins.appartiene(50))       // controllo che il casco sia presente nella valigia
        {
            el3 = 50;
        }
        file << el3 << '\n';
        if(ins.appartiene(51))       // controllo che la tuta sia presente nella valigia
        {
            el4 = 51;
        }
        file << el4 << '\n';
        if(ins.appartiene(52))       // controllo che il camice sia presente nella valigia
        {
            el5 = 52;
        }
        file << el5 << '\n';
        if(ins.appartiene(55))       // controllo che il manuale sia presente nella valigia
        {
            el6 = 55;
        }
```

```
        file << el6 << '\n';
    }
    //Fine modifiche
    save_specifiche(file);
    salvaStatiDialoghi(); //modifiche D'Andria Dresda, metodo per salvare gli stati su file
    bacheca.SalvaBacheca(file);
    salvaJukeBoxLuci(); //modifiche VAcca - Salva lo stato del Jukebox e delle luci
    file.close();
    //INIZIO MODIFICHE CICALA GIACOMO
    for(int k=1; k<=31; k++)
        if(slotmachine.appartiene(k))
            numEuro++;

    file << numEuro << '\n';

    for(int j=1; j<=31; j++)
        if(slotmachine.appartiene(j))
            file << j << '\n';
    //FINE MODIFICHE CICALA GIACOMO
}
```

```
void Gioco::load()
{
    int i;
    int valore;
    int slot; //CICALA GIACOMO
    svuotaInsieme(slotmachine); //CICALA GIACOMO

    ifstream file(fStringa.c_str(), ios::in);
    //modifiche effettuate da D'Andria Dresda sul controllo del caricamento da file
    ifstream astrostatidialoghi("Astrostatidialoghi.txt");
    bool astrostatidialoghiaperto=astrostatidialoghi.good();
    bool fileaperto=file.good();
    if(fileaperto || astrostatidialoghiaperto)
    {

        interfaccia.scrivi("Ripristino partita...");
        if(fileaperto)
        {
            for (i = 1; i <= oggetti.get_n_oggetti(); i++)
            {
                file >> valore;
                oggetti.set_luogo(i,valore);
            }
            file >> luogo_attuale;
            file >> tempo;
            file >> passo_soluzione;

            while(!(z.pilavuota()))                          //controllo che lo zaino non sia vuoto
                z.fuoripila();
            file >> n_oggettiZ;
            file >> peso_MaxZ;
            file >> el1;
            file >> el2;
            if(el2 !=0)
                z.inpila(el2);
            if(el1 !=0)
                z.inpila(el1);

            file >> n_oggettiZV;
            file >> peso_MaxZV;
            file >> el3;
            file >> el4;
            file >> el5;
            file >> el6;
            if((!(ins.appartiene(el3))) && el3 != 0)
                ins.inserisci(el3);
            if((!(ins.appartiene(el4))) && el4 != 0)
                ins.inserisci(el4);
            if((!(ins.appartiene(el5))) && el5 != 0)
                ins.inserisci(el5);
            if((!(ins.appartiene(el6))) && el6 != 0)
                ins.inserisci(el6);
            //Fine ML
```

```
            load_specifiche(file);
            bacheca.CaricaBacheca(file);
            file.close();
        }
        if(astrostatidialoghiaperto)
        {
            caricaStatiDialoghi(); //modifiche D'Andria Dresda
        }
        ifstream jukeLuci("JukeLuci.txt");
        if (jukeLuci.good() )
        {
            caricaJukeBoxLuci();
        }

    }
    else
    {
        interfaccia.scrivi("Non ci sono partite salvate!");
    }
    //INIZIO MODIFICHE CICALA GIACOMO
    file >> numEuro;

    for(int j=1; j<=numEuro; j++)
    {
        file >> slot;
        slotmachine.inserisci(slot);
    }
    //fine modifiche CICALA GIACOMO
}
```

4.1.5 Mappa.h

È stata aggiunta l'include per permettere l'implementazione dello zaino:

```
#include "Zaino.h"
```

4.1.6 Interfaccia.cpp

È stata modificata la seguente funzione:

```
void Interfaccia::elenca_oggetti(Oggetti oggetti, string premessa)
{
    //Modifica Mirco Sternativo
    int n_oggettiZ = oggetti.get_n_oggettiZ();

    if (n_oggettiZ > 0)
    {
        cout << premessa;
        for (int i = 1; i <= n_oggettiZ; i++)
        {
            cout << "\n- " << oggetti.get_oggetto(i).get_nome();

            if (i == n_oggettiZ)
            {
                cout << ".";
            }
            else
            {
                cout << ";";
            }
        }
        cout << endl;
    }
    //fine modifiche
}
```

4.1.7 Oggetti.h

Sono stati aggiunti i seguenti metodi:

```
int get_n_oggettiZ();
void get_zaino(int);
int get_zaino2(int);
int get_valigia(int);
```

4.1.8 Oggetti.cpp

Sono stati aggiunti i seguenti metodi:

```
int Oggetti::get_n_oggettiZ()
{
    return fo;
}


void Oggetti::get_zaino(int c)
{
    bool trovato = false;
    int i = 1;
    while (i <= fo && !trovato) {
      // scandisco tutti gli oggetti presenti nel vocabolario finché o finiscono
      gli oggetti o trovato viene impostato a true
        if (oggetti[i].get_codice() == c){
      // controllo che il codice dell'oggetto con indice "i" sia uguale al
      codice "c"
            cout << "- " <<oggetti[i].get_nome() << endl;
            trovato = true;
        }
        i++;
    }
}
```

```
int Oggetti::get_zaino2(int c)
{
    bool trovato = false;
    int i = 1;
    int i2 = 0;
    while (i <= fo && !trovato) {
      // scandisco tutti gli oggetti presenti nel vocabolario finché o finiscono
      gli oggetti o trovato viene impostato a true
      if (oggetti[i].get_codice() == c){
      // controllo che il codice dell'oggetto con indice "i" sia uguale al
codice "c"
            i2=i;
            trovato = true;
        }
        i++;
    }
    return(i2);
}


int Oggetti::get_valigia(int c)
{
    bool trovato = false;
    int i = 1;
    int i2 = 0;
    while (i <= fo && !trovato) {
      // scandisco tutti gli oggetti presenti nel vocabolario finché o finiscono
gli oggetti o trovato viene impostato a true
        if (i == c){//controllo che l'indice "i" sia uguale al codice "c"
            i2 = oggetti[i].get_codice();
            trovato = true;
        }
        i++;
    }
    return(i2);
}
```

Ed è stata modificata la seguente funzione:

```
int Oggetti::luogo_oggetto(int c2, int lu) {
    bool trovato = false;
    int i = 1;
    int og = 0;

    while (i <= fo && !trovato) {
        if (c2 == 30){
            if(oggetti[i].get_luogo() == lu){
                og = i;
            }
        }else if (oggetti[i].get_codice() == c2)
            if (abs(oggetti[i].get_luogo()) == lu || oggetti[i].get_luogo() == 0
|| oggetti[i].get_luogo() == 20) {
                og = i;
                trovato = true;
            }
        i++;
    }
    return og;
}
```

4.1.9 Oggetto.h

È stato creato un nuovo costruttore che integra l'attributo peso

```
Oggetto(string n, int c, int l, int w);
//Modifica Mirco Sternativo -- Peso oggetto
```

È stato modificato il costruttore avente l'attributo prezzo, aggiungendo il peso

```
Oggetto(string n, int c, int l, int w, float p );
//MODIFICA D-R(D'Orsi):Negozio + Banca
//Modifica Mirco Sternativo -- Peso oggetto
```

È stato modificato il metodo get_peso:

```
int get_peso();              //Modifica Mirco Sternativo
```

4.1.10 Oggetto.cpp

È stato aggiunto il seguente metodo

```
Oggetto::Oggetto(string n, int c, int l, int w)      //w: weight
{
    nome = n;
    codice = c;
    luogo = l;
    peso = w;
}
```

Sono stati modificati i seguenti metodi

```
Oggetto::Oggetto(string n, int c, int l, int w, float p)
{
  nome = n;
  codice = c;
  luogo = l;
  peso = w;
  prezzo = p;
}

int Oggetto::get_peso()
{
    return peso;
}
```

4.1.11 Zaino.h

È stato semplicemente aggiunto il file senza alcuna modifica particolare.

4.1.12 Valigia.h

È stato semplicemente aggiunto il file senza alcuna modifica particolare.

5 Accorgimenti e risoluzione di bug

È stata riscontrata, durante l'analisi, una serie di bug che non permettevano il corretto funzionamento del gioco nel progetto di base (Rosa Chiarappa). Nello specifico:

- Si è riscontrata un evidente disparita fra tutti i codici degli oggetti e delle azioni a loro correlate, fra i progetti. Sono state apportate modifiche ai codici:
  - degli oggetti (salvo quelli standard es *4 tuta*, *11 casco, 23 camince, ecc*);
  - delle azioni, infatti i codici "basati" sul modello LLVVOO non erano espressi nell'ordine delle decine ma delle migliaia. Quindi un ipotetico codice dello Sternativo, *guarda zaino* definita dal codice *10155*, diviene nel progetto base, *(00)100155*.
- Si è evidenziata un'evidente disparità rappresentativa dei codici degli oggetti nell'applicativo.
  In particolare, la disparità fra il codice di un vocabolo e il codice *logico* che lo stesso oggetto ricopre nel sistema. Per esempio, il vocabolo *zaino* è salvato nel vocabolario con il codice 155, ma il numero dell'oggetto , nell'elenco degli oggetti dell'applicativo, sarà 144.
  Per ovviare approssimativamente a ciò si è scelto di commentare il codice enumerando di 5 in 5 tutti gli oggetti nella fase di immissione (Astro.cpp) , per avere un , se pur poco significante, riferimento.

6 Strutture dati intercambiabili

Si è scelto di implementare la *Coda con puntatori* nel progetto base perché migliore e completa. Si sono applicati piccole migliorie al fine dell'implementazione.

Per quanto riguarda la seconda struttura dati si è scelto di implementare *una coda con priorità* realizzata mediante una lista ordinata con puntatori monodirezionale. .

Entrambe le realizzazioni sono state aggiunte nella cartella "Strutture intercambiabili" del progetto "base 1911" .

6.1 Coda con puntatori

```
/*
    Coda.h

      Created by: Mirco Sternativo
      Edited by:  Matteo Luceri

    Date: 16-Mar-2015 // 30-Oct-2019
*/

#ifndef _CODA_H
#define _CODA_H

#include "Cella.h"
#include <iostream>
#include <cstdlib>
using namespace std;

template<class tipoelem>
class Coda{

    public:


        //COSTRUTTORE E DISTRUTTORE ------------------------------------------------------
--------------
        Coda();                         //costruttore coda
        Coda(Coda&);                    //costruttore di copia
        ~Coda();                        //distruttore coda

        //OPERATORI ----------------------------------------------------------------------
--------------
        void creacoda();
        bool codavuota() const;
        tipoelem leggicoda() const;
        void incoda(tipoelem);
        void fuoricoda();
             //NUOVI METODI
        void inverti_coda();
        void svuota();

    private:
             //DEFINIZIONE DEI TIPI -----------------------------------------------------
-------------------
        typedef Cella<tipoelem>* posizione;
        //VARIABILI ----------------------------------------------------------------------
--------------
        posizione testa;
        posizione fondo;


};
```

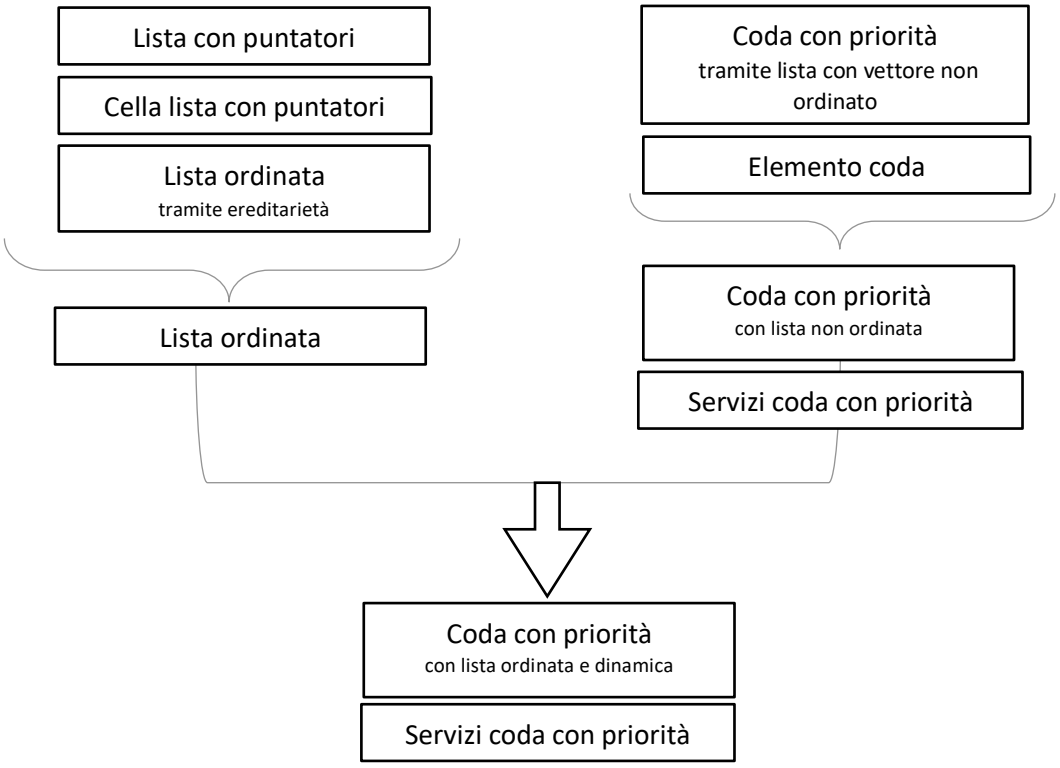
```
//IMPLEMENTAZIONI COSTRUTTORE E DISTRUTTORE ----------------------------------------
--------------
template<class tipoelem>
Coda<tipoelem>::Coda(){
    creacoda();
}


template<class tipoelem>
Coda<tipoelem>::Coda(Coda<tipoelem>& c){
    //N.B.: questo costruttore effettua una copia o clone di un oggetto
    creacoda();
    tipoelem temp;
    Coda<tipoelem> comodo;

    while (!c.codavuota()){
        comodo.incoda(c.leggicoda());
        //copio gli elementi di c in una coda d'appoggio
        c.fuoricoda();                          //distruzione della coda
    }

    while (!comodo.codavuota()){
        temp=comodo.leggicoda();
        comodo.fuoricoda();
        incoda(temp);
        //copia degli elementi della coda d'appoggio nella nuova coda
        c.incoda(temp);
        //e ripristino c
    }
}


template<class tipoelem>
Coda<tipoelem>::~Coda(){
    while (!codavuota())
    {
        fuoricoda();
        //eliminazione elementi coda
    }
    delete fondo;
      //eliminazione riferimenti per inizio e fine coda
    delete testa;
}



//IMPLEMENTAZIONI OPERATORI ---------------------------------------------------------
--------------
template<class tipoelem>
void Coda<tipoelem>::creacoda(){
    testa=nullptr;
    fondo=nullptr;
}
```

```
template<class tipoelem>
bool Coda<tipoelem>::codavuota() const{
    return ((testa==nullptr));
    //controllo esistenza coda tramite verifica puntatori inizio e fine
}


template<class tipoelem>
tipoelem Coda<tipoelem>::leggicoda() const {
    if (!codavuota())
        //precondizione coda non vuota
        return (testa->leggicella());
    //lettura elemento in testa
}


template<class tipoelem>
void Coda<tipoelem>::incoda(tipoelem a){
    posizione temp=new Cella<tipoelem>;
    //creazione nuovo elemento temp
    temp->scrivicella(a);
    //valorizzazione di temp
    temp->scrivisucc(nullptr);

    if (!codavuota())
        fondo->scrivisucc(temp);
             fondo=temp;
        //se la coda non è vuota, l'elemento successivo è temp
    else
        testa=temp;
             fondo=temp;
        //altrimenti sta alla testa

}


template<class tipoelem>
void Coda<tipoelem>::fuoricoda(){
    if (!codavuota()){
    //precondizione coda non vuota

             posizione temp=testa;
             //puntatore all'elemento da eliminare

             testa=testa->leggisucc();
             delete temp;
             //eliminazione elemento in testa
}

// NUOVI METODI-------------------------------------------------------------------------
-------------
template<class tipoelem>
void Coda<tipoelem>::inverti_coda(){
    tipoelem Elemento;
```

```
    if (!codavuota()){
        Elemento = leggicoda();
        fuoricoda();
        inverti_coda();
        incoda(Elemento);
    }
}


template<class C>
void Coda<C>::svuota(){
    while (!codavuota()){
        fuoricoda();
    }
};

#endif // CODA_H
```

6.2 Coda con priorità tramite lista ordinata

Si fornisce uno schema riassuntivo del processo di creazione della struttura:



6.2.1 Cella_Lista_Con_Puntatori_Monodirezionale.h

```
/*
Realizzazione:  Cella per Lista con Puntatori Monodirezionale
Modificato da:  Matteo Luceri
Autore       :  Sconosciuto
*/

#ifndef CELLA_LISTA_CON_PUNTATORI_MONODIREZIONALE_H_
#define CELLA_LISTA_CON_PUNTATORI_MONODIREZIONALE_H_

template<class T>
class Cella_LP_Mono
{
public:
        typedef T tipoelem;
    Cella_LP_Mono();
        ~Cella_LP_Mono();

    void scriviCella(tipoelem);
    tipoelem leggiCella() const;
    void scriviSucc(Cella_LP_Mono<T>*);
    Cella_LP_Mono<T>* leggiSucc() const;

private:
    tipoelem elemento;
    Cella_LP_Mono<T>* succ;
```

```
};

template<class T>
Cella_LP_Mono<T>::Cella_LP_Mono()
{
        succ = nullptr;
}

template<class T>
Cella_LP_Mono<T>::~Cella_LP_Mono()
{
    //dtor
}

template<class T>
void Cella_LP_Mono<T>::scriviCella(tipoelem label)
{
    elemento = label;
}

template<class T>
T Cella_LP_Mono<T>::leggiCella() const
{
    return elemento;
}


template<class T>
void Cella_LP_Mono<T>::scriviSucc(Cella_LP_Mono<T>* c)
{
    succ = c;
}

template<class T>
Cella_LP_Mono<T>* Cella_LP_Mono<T>::leggiSucc() const
{
    return succ;
}



#endif /* CELLA_LISTA_CON_PUNTATORI_MONODIREZIONALE_H_ */
```

6.2.2 Lista_Con_Puntatori_Monodirezionale.h

```
/*
Realizzazione: Lista con Puntatori Monodirezionale
Modificato da   Matteo Luceri
Autore  :       Sconosciuto
Nota Bene: Il costruttore di copia e' stato implementato in quanto
           NECESSARIO per via di alcuni file del progetto in cui
           vengono passate liste come parametri di funzioni e
           restituite (dalle stesse) non per riferimento.
*/

#ifndef LISTA_CON_PUNTATORI_MONODIREZIONALE_H_
#define LISTA_CON_PUNTATORI_MONODIREZIONALE_H_

#include "Cella_Lista_Con_Puntatori_Monodirezionale.h"

template<class L>
class Lista
{
public:
    typedef Cella_LP_Mono<L>* posizione;
    typedef L tipoelem;

    Lista();
```

```
        Lista(const Lista&);
    ~Lista();

    void crealista();
    bool listavuota() const;
    bool finelista(posizione) const;
    posizione primolista() const;
    posizione succlista(posizione) const;
    posizione preclista(posizione) const;
    tipoelem leggilista(posizione) const;
    void scrivilista(tipoelem, posizione);
    void inslista(tipoelem, posizione&);
    void canclista(posizione &);

private:
    posizione lista;
};


template <class L>
Lista<L>::Lista()
{
    crealista();
}

template <class L>
Lista<L>::Lista(const Lista& b)
{
    crealista();
    typename Lista<L>::posizione ind = primolista(),ind2 = b.primolista();
    while (!b.finelista(ind2))
    {
        inslista(b.leggilista(ind2),ind);
        ind = succlista(ind);
        ind2 = b.succlista(ind2);
    }
}

template <class L>
Lista<L>::~Lista()
{
    posizione p = primolista();
    while(p->leggiSucc() != nullptr)
    {
        posizione temp = p;
        p=p->leggiSucc();
        delete temp;
    }
}

template <class L>
void Lista<L>::crealista()
{
    lista = new Cella_LP_Mono<L>;
    lista->scriviSucc(nullptr);
}

template <class L>
bool Lista<L>::listavuota() const
{
    return (lista->leggiSucc() == nullptr);
}

template <class L>
bool Lista<L>::finelista(posizione pos) const
{
    return (pos->leggiSucc() == nullptr);
}
```

```
template <class L>
typename Lista<L>::posizione Lista<L>::primolista() const
{
    return lista;
}

template <class L>
typename Lista<L>::posizione Lista<L>::succlista(posizione pos) const
{
    if (lista->leggiSucc() != nullptr)
        return (pos->leggiSucc());
    else
        return pos;
}

template <class L>
typename Lista<L>::posizione Lista<L>::preclista(posizione pos) const
{
    typename Lista<L>::posizione temp = primolista();
    if(pos == temp)
        temp = nullptr;
    else
    {
        while(succlista(temp) != pos)
            temp = succlista(temp);
    }
    return temp;
}

template <class L>
L Lista<L>::leggilista(posizione pos) const
{
    return (pos->leggiCella());
}

template <class L>
void Lista<L>::scrivilista(tipoelem elem, posizione pos)
{
    pos->scriviCella(elem);
}

template <class L>
void Lista<L>::inslista(tipoelem elem, posizione &pos)
{
    typename Lista<L>::posizione temp;
    temp = new Cella_LP_Mono<L>;
    temp->scriviCella(elem);
    temp->scriviSucc(pos);

    if (pos == primolista())
        lista = temp;
    else
        preclista(pos)->scriviSucc(temp);
    pos = temp;
}

template <class L>
void Lista<L>::canclista(posizione & pos)
{
    typename Lista<L>::posizione temp = pos;

    if(pos != lista)
        preclista(pos)->scriviSucc(pos->leggiSucc());
    else
        lista = pos->leggiSucc();

    pos = pos->leggiSucc();
    delete temp;
}
```

```
#endif /* LISTA_CON_PUNTATORI_MONODIREZIONALE_H_ */
```

6.2.3 Lista_Ordinata_Ereditata.h

```
/*
 Definizione della struttura dati "Lista Ordinata".
 Definizione di Lista Ordinata per la realizzazione tramite ereditarietà da Lista.
 Autore:        Andrea Esposito.
 Modificata da: Matteo Luceri
*/

#ifndef LISTA_ORDINATA_EREDITATA_H_
#define LISTA_ORDINATA_EREDITATA_H_

#include "Lista_Con_Puntatori_Monodirezionale.h"

template <class T>
class ListaOrdinata : private Lista<T>
{
public:
    typedef typename Lista<T>::posizione posizione;

    ListaOrdinata();
    ListaOrdinata(const ListaOrdinata&);
    ~ListaOrdinata();

    void crealista();
    bool listavuota() const;
    bool finelista(posizione) const;
    posizione primolista() const;
    posizione succlista(posizione) const;
    posizione preclista(posizione) const;
    T leggilista(posizione) const;
    void inslista(T);
    void canclista(posizione&);
};

template <class T>
ListaOrdinata<T>::ListaOrdinata() :
    Lista<T>() // call parent constructor
{

}

template <class T>
ListaOrdinata<T>::ListaOrdinata(const ListaOrdinata<T>& l):
    Lista<T>(l) // call parent constructor
{

}

template <class T>
ListaOrdinata<T>::~ListaOrdinata()
{
    Lista<T>::~Lista();
}

template <class T>
void ListaOrdinata<T>::crealista()
{
    Lista<T>::crealista();
}

template <class T>
bool ListaOrdinata<T>::listavuota() const
{
    return Lista<T>::listavuota();
}
```

```
template <class T>
typename ListaOrdinata<T>::posizione ListaOrdinata<T>::primolista() const
{
    return Lista<T>::primolista();
}

template <class T>
bool ListaOrdinata<T>::finelista(posizione p) const
{
    return Lista<T>::finelista(p);
}

template <class T>
typename ListaOrdinata<T>::posizione ListaOrdinata<T>::succlista(posizione p) const
{
    return Lista<T>::succlista(p);
}

template <class T>
typename ListaOrdinata<T>::posizione ListaOrdinata<T>::preclista(posizione p) const
{
    return Lista<T>::preclista(p);
}

template <class T>
void ListaOrdinata<T>::inslista(T el)
{
    posizione p = primolista();
    if(!listavuota())
        while(leggilista(p) < el && !finelista(p))
            p = succlista(p);

    Lista<T>::inslista(el, p);
}

template <class T>
void ListaOrdinata<T>::canclista(posizione &p)
{
    Lista<T>::canclista(p);
}

template <class T>
T ListaOrdinata<T>::leggilista(posizione p) const
{
    return Lista<T>::leggilista(p);
}

#endif /* LISTA_ORDINATA_EREDITATA_H_ */
```

6.2.4 Elemento_Coda_Con_Priorita.h

```
/*
Realizzazione:  Tipo di elemento della coda con priorità
Modificato da:  Matteo Luceri
Autore :Sconosciuto
*/

#ifndef ELEMENTO_CODA_CON_PRIORITA_H_
#define ELEMENTO_CODA_CON_PRIORITA_H_

#include <iostream>
#include <cstdlib>

using namespace std;


template<class X> class Priorielem
```

```
{
public:

    //dichiarazione di tipo
    typedef float priorita; //la priorità è di tipo numerico (valore minore=priorità più alta)
    typedef X tipoelem;

    //costruttori
    Priorielem();
    Priorielem(const Priorielem&);
    Priorielem(priorita,tipoelem);

    //distruttore di default

    //setter e getter
    void scrivipriorita(priorita);
    priorita leggipriorita() const;
    void scrivielem(tipoelem);
    tipoelem leggielem() const;

    //sovraccarichi
    void operator=(Priorielem);
    bool operator==(Priorielem);
    bool operator<(Priorielem);
    bool operator>(Priorielem);

private:
    priorita prior; //priorità
    tipoelem elem; //informazione

};

template <class X> Priorielem<X>::Priorielem() //costruttore generico
{
    prior=9999;
}

template <class X> Priorielem<X>::Priorielem(const Priorielem& p) //costruttore di copia
{
    prior=p.leggipriorita();
    elem=p.leggielem();
}

template <class X> Priorielem<X>::Priorielem(priorita p,tipoelem e) //costruttore specifico
{
    prior=p;
    elem=e;
}

template <class X> void Priorielem<X>::scrivipriorita(priorita p)
{
    prior=p;
}

template <class X> void Priorielem<X>::scrivielem(tipoelem e)
{
    elem=e;
}

template <class X> float Priorielem<X>::leggipriorita() const
{
    return(prior);
}

template <class X> X Priorielem<X>::leggielem() const
{
    return(elem);
}
```

```
//sovraccarichi

template <class X> void Priorielem<X>::operator=(Priorielem<X> p) //assegnamento
{
    prior=p.leggipriorita();
    elem=p.leggielem();
}

template <class X> bool Priorielem<X>::operator==(Priorielem<X> p) //uguaglianza
{
    return (elem==p.leggielem());
}

template <class X> bool Priorielem<X>::operator<(Priorielem<X> p) //maggioranza (solo sulla priorità)
{
    return (prior<p.leggipriorita());
}

template <class X> bool Priorielem<X>::operator>(Priorielem<X> p) //minoranza (solo sulla priorità)
{
    return (prior>p.leggipriorita());
}

//sovraccarico output

template<class X> ostream& operator<<(ostream& os, const Priorielem<X>& p)
{
    os<<"("<<p.leggipriorita()<<"|"<<p.leggielem()<<")";
    return(os);
}



#endif /* ELEMENTO_CODA_CON_PRIORITA_H_ */
```

6.2.5 Coda_Con_Priorita.h

```
/*
Realizzazione:  Coda con priorità tramite una lista ordinata
Note:           Valore di priorità minore indica priorità maggiore
Modificato da:  Matteo Luceri
Autore      :   Sconosciuto
*/

#ifndef CODA_CON_PRIORITA_H_
#define CODA_CON_PRIORITA_H_

#include "Lista_Ordinata_Ereditata.h"
#include <iostream>
#include <cstdlib>
#include "Elemento_Coda_Con_Priorita.h"

using namespace std;

//realizzazione di una
template<class P> class Prioricoda
{
public:

    //definizione di tipo
    typedef Priorielem<P> tipoelem;

    //costruttori
```

```
    Prioricoda();
    Prioricoda(const Prioricoda<P>&);

    //distruttore di default

    //operatori di specifica
    void creaprioricoda();
    void inserisci(tipoelem);
    tipoelem min() const;
    void cancellamin();

    //operatori ereditati dell'insieme
    bool insiemevuoto() const;
    bool appartiene(tipoelem) const;

private:
    ListaOrdinata<tipoelem> prioricoda;  //la coda con priorità di fatto è una lista

    friend ostream& operator<< (ostream& o, const Prioricoda<P>& p) //sovraccarico output
    {
        o<<p.prioricoda;
        return o;
    }

};


template<class P> Prioricoda<P>::Prioricoda() //costruttore generico
{
    creaprioricoda();
}

template <class P> Prioricoda<P>::Prioricoda(const Prioricoda<P>& p) //costruttore di copia
{
    creaprioricoda();
    prioricoda=p.prioricoda;
}

template<class P> void Prioricoda<P>::creaprioricoda() //crea la coda con priorità
{
    prioricoda.crealista();
}

template<class P> void Prioricoda<P>::inserisci(tipoelem a) //inserimento
{
    prioricoda.inslista(a);
}

template<class P> Priorielem<P> Prioricoda<P>::min() const //restituisce il minimo della coda
{
        tipoelem m;
    if (!prioricoda.listavuota())  //precondizione coda non vuota
    {
         typename ListaOrdinata<tipoelem>::posizione indice=prioricoda.primolista();
         m= prioricoda.leggilista(indice);
    }

    return m;
}

template<class P> void Prioricoda<P>::cancellamin() //elimina il minimo dalla coda
{
    if (!prioricoda.listavuota())  //precondizione coda non vuota
    {
        typename ListaOrdinata<tipoelem>::posizione indice=prioricoda.primolista();
        prioricoda.canclista(indice);
    }
}
```

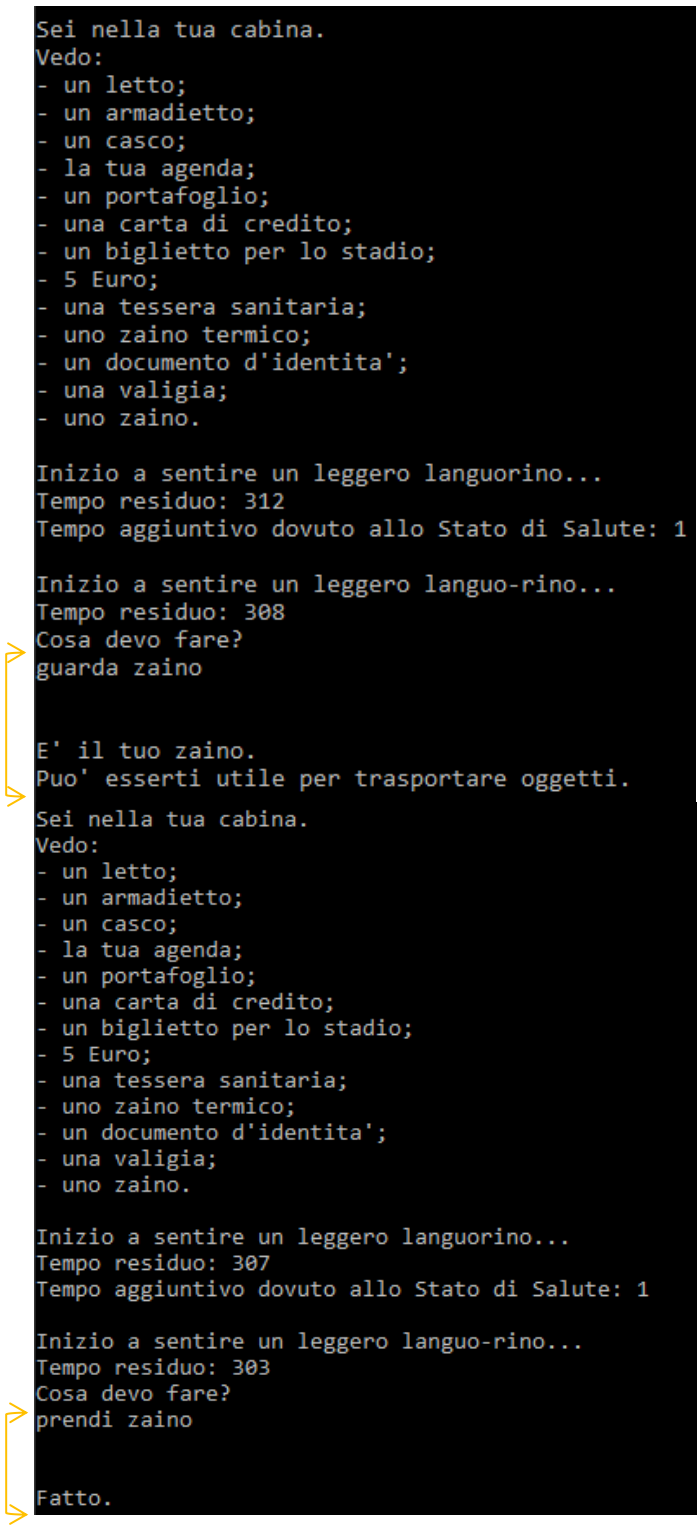
```
template<class P> bool Prioricoda<P>::insiemevuoto() const //verifica se la coda è vuota
{
    return (prioricoda.listavuota());
}

template<class P> bool Prioricoda<P>::appartiene(tipoelem a) const //verifica se l'elemento appartiene alla
coda
{
    bool trovato=false;
    if (!prioricoda.listavuota())
    {
        typename ListaOrdinata<tipoelem>::posizione indice=prioricoda.primolista(); //ricerca
        while (!prioricoda.finelista(indice) && !trovato)
        {
            if (prioricoda.leggilista(indice)==a) //se l'elemento corrente è il cercato
                trovato=true;
            else indice=prioricoda.succlista(indice);
        }
    }
    return(trovato);
}



#endif /* CODA_CON_PRIORITA_H_ */
```

6.2.6 Servizi_Coda_Con_Priorita.h

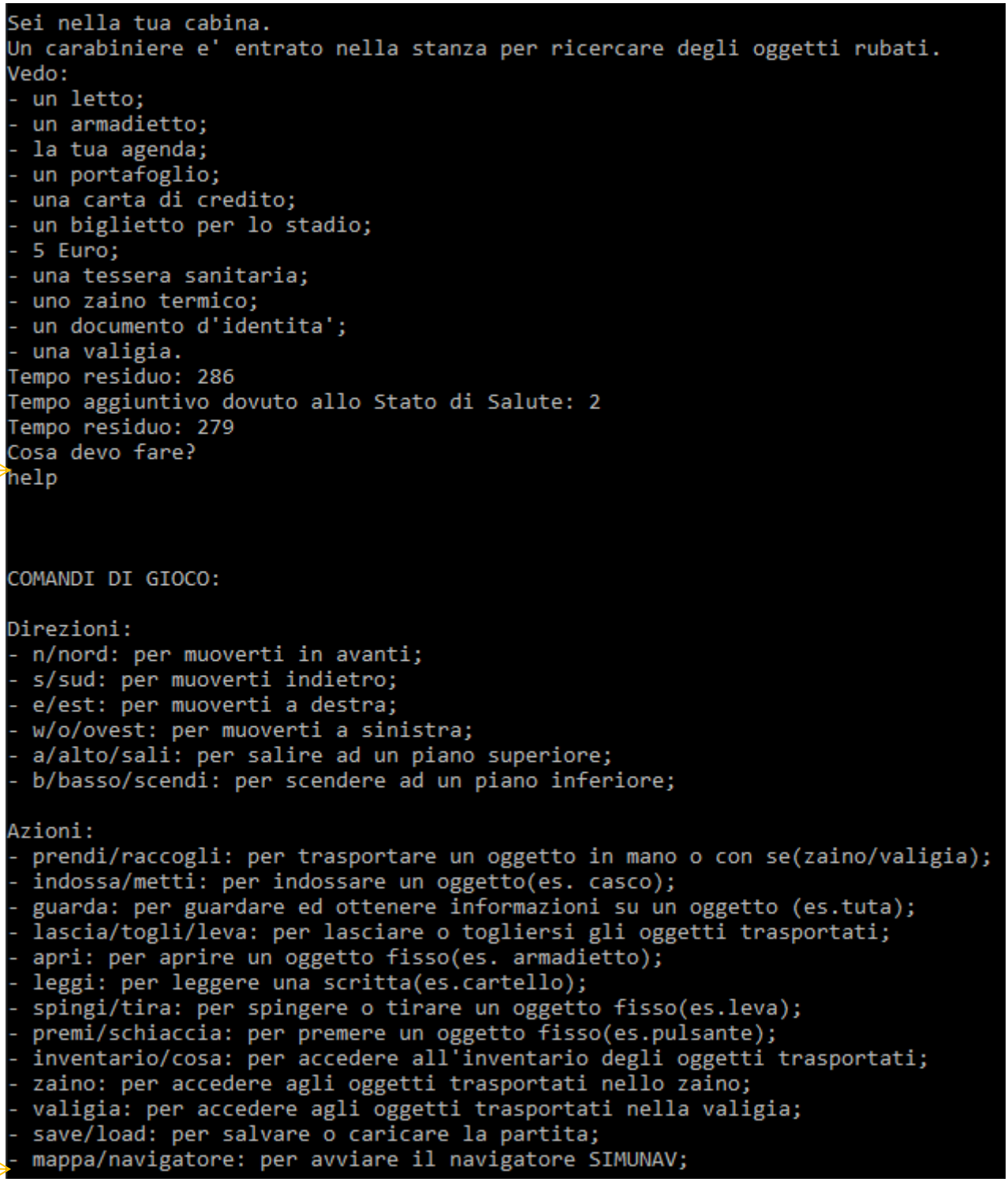
```
/*
 Definizione dei servizi per la struttura Coda Con Priorità
 Autore:        Sconosciuto
 Modificata da: Matteo Luceri
*/


#ifndef SERVIZI_CODA_CON_PRIORITA_H_
#define SERVIZI_CODA_CON_PRIORITA_H_

#include <string>
#include <iostream>
#include <fstream>
#include <exception>

#include "Coda_Con_Priorita.h"

template <class T>
void stampaPrioriCoda(Prioricoda<T>&);

template <class T>
void inputPrioriCodaDaFile(Prioricoda<T>&, std::ifstream&);

template <class T>
void outputPrioriCodaSuFile(Prioricoda<T>&, std::ofstream&);

// Implementazione

template <class T>
void stampaPrioriCoda(Prioricoda<T>& p)
{
        if(!p.insiemevuoto())
        {
                typename Prioricoda<T>::tipoelem el = p.min();
                std::cout << el.leggielem() << " - con priorita: " << el.leggipriorita() << std::endl;
                p.cancellamin();

                stampaPrioriCoda(p);

                p.inserisci(el);
```

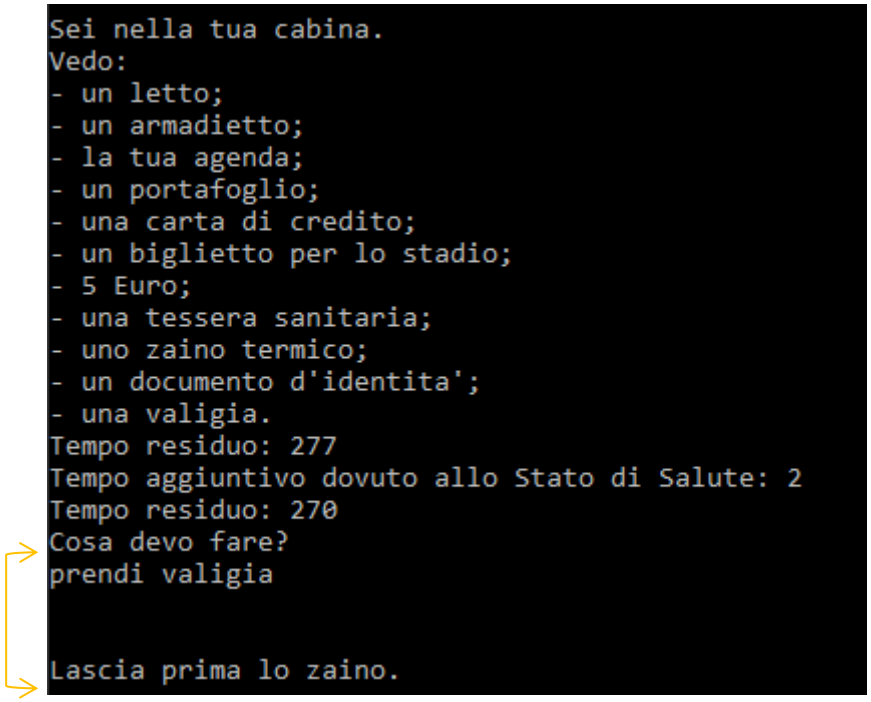
```
        }
}

template <class T>
void inputPrioriCodaDaFile(Prioricoda<T>& p, std::ifstream& file)
{
        if(!file.fail())
        {
                if(file.peek() != ifstream::traits_type::eof())
                        while(!file.eof())
                        {
                                T content;
                                string s;
                                getline(file, s,',');
                                file >> content;
                                typename Prioricoda<T>::tipoelem::priorita priori = (typename
Prioricoda<T>::tipoelem::priorita) atof(s.c_str());
                                p.inserisci(typename Prioricoda<T>::tipoelem(priori, content));
                        }
        }
        else
                throw std::runtime_error("Errore di apertura del file");
}

template <class T>
void outputPrioriCodaSuFile(Prioricoda<T>& p, std::ofstream& file)
{
        if(!file.fail())
        {
                if(!p.insiemevuoto())
                {
                        typename Prioricoda<T>::tipoelem el = p.min();
                        file << el.leggipriorita() << "," << el.leggielem() << std::endl;
                        p.cancellamin();

                        outputPrioriCodaSuFile(p, file);

                        p.inserisci(el);
                }
        }
        else
                throw std::runtime_error("Errore di apertura del file");
}

#endif /* SERVIZI_CODA_CON_PRIORITA_H_ */
```

7. Test


Sei nella tua cabina.
Vedo:
- un letto;
- un armadietto;
- un casco;
- la tua agenda;
- un portafoglio;
- una carta di credito;
- un biglietto per lo stadio;
- 5 Euro;
- una tessera sanitaria;
- uno zaino termico;
- un documento d'identita';
- una valigia;
- uno zaino.
Inizio a sentire un leggero languorino...
Tempo residuo: 312
Tempo aggiuntivo dovuto allo Stato di Salute: 1
Inizio a sentire un leggero languo-rino...
Tempo residuo: 308
Cosa devo fare?
guarda zaino
E' il tuo zaino.
Puo' esserti utile per trasportare oggetti.
Sei nella tua cabina.
Vedo:
- un letto;
- un armadietto;
- un casco;
- la tua agenda;
- un portafoglio;
- una carta di credito;
- un biglietto per lo stadio;
- 5 Euro;
- una tessera sanitaria;
- uno zaino termico;
- un documento d'identita';
- una valigia;
- uno zaino.
Inizio a sentire un leggero languorino...
Tempo residuo: 307
Tempo aggiuntivo dovuto allo Stato di Salute: 1
Inizio a sentire un leggero languo-rino...
Tempo residuo: 303
Cosa devo fare?
prendi zaino
Fatto.

```
Sei nella tua cabina.
Vedo:
- un letto;
- un armadietto;
- un casco;
- la tua agenda;
- un portafoglio;
- una carta di credito;
- un biglietto per lo stadio;
- 5 Euro;
- una tessera sanitaria;
- uno zaino termico;
- un documento d'identita';
- una valigia.

Inizio a sentire un leggero languorino...
Tempo residuo: 302
Tempo aggiuntivo dovuto allo Stato di Salute: 1
Tempo residuo: 297
Cosa devo fare?
prendi casco


Se vuoi portare l'oggetto con te, premi(y); se vuoi metterlo nello zaino, premi(z)
z
Ora e' nello zaino.
Fatto.

Sei nella tua cabina.
Vedo:
- un letto;
- un armadietto;
- la tua agenda;
- un portafoglio;
- una carta di credito;
- un biglietto per lo stadio;
- 5 Euro;
- una tessera sanitaria;
- uno zaino termico;
- un documento d'identita';
- una valigia.
Tempo residuo: 295
Tempo aggiuntivo dovuto allo Stato di Salute: 2
Tempo residuo: 288
Il carabiniere ti ha perquisito e non ha trovato nessun oggetto rubato in tuo possesso.

Cosa devo fare?
zaino


Inventario Zaino

Vedo in cima:
- un casco

Totale Oggetti nello Zaino: 1
Spazio disponibile: 2 su 5 kg.
```


Sei nella tua cabina.
Un carabiniere e' entrato nella stanza per ricercare degli oggetti rubati.
Vedo:
- un letto;
- un armadietto;
- la tua agenda;
- un portafoglio;
- una carta di credito;
- un biglietto per lo stadio;
- 5 Euro;
- una tessera sanitaria;
- uno zaino termico;
- un documento d'identita';
- una valigia.
Tempo residuo: 286
Tempo aggiuntivo dovuto allo Stato di Salute: 2
Tempo residuo: 279
Cosa devo fare?
help
COMANDI DI GIOCO:
Direzioni:
- n/nord: per muoverti in avanti;
- s/sud: per muoverti indietro;
- e/est: per muoverti a destra;
- w/o/ovest: per muoverti a sinistra;
- a/alto/sali: per salire ad un piano superiore;
- b/basso/scendi: per scendere ad un piano inferiore;
Azioni:
- prendi/raccogli: per trasportare un oggetto in mano o con se(zaino/valigia);
- indossa/metti: per indossare un oggetto(es. casco);
- guarda: per guardare ed ottenere informazioni su un oggetto (es.tuta);
- lascia/togli/leva: per lasciare o togliersi gli oggetti trasportati;
- apri: per aprire un oggetto fisso(es. armadietto);
- leggi: per leggere una scritta(es.cartello);
- spingi/tira: per spingere o tirare un oggetto fisso(es.leva);
- premi/schiaccia: per premere un oggetto fisso(es.pulsante);
- inventario/cosa: per accedere all'inventario degli oggetti trasportati;
- zaino: per accedere agli oggetti trasportati nello zaino;
- valigia: per accedere agli oggetti trasportati nella valigia;
- save/load: per salvare o caricare la partita;
- mappa/navigatore: per avviare il navigatore SIMUNAV;


Sei nella tua cabina.
Vedo:
- un letto;
- un armadietto;
- la tua agenda;
- un portafoglio;
- una carta di credito;
- un biglietto per lo stadio;
- 5 Euro;
- una tessera sanitaria;
- uno zaino termico;
- un documento d'identita';
- una valigia.
Tempo residuo: 277
Tempo aggiuntivo dovuto allo Stato di Salute: 2
Tempo residuo: 270
Cosa devo fare?
prendi valigia
Lascia prima lo zaino.