Esercitazione N°5

Luca Zepponi
(Sara Berdini)

24 febbraio 2023

1. DESCRIZIONE DEL PROGRAMMA

Il programma consiste in un'applicazione che mira a simulare un gioco di esplorazione cooperativa di un territorio ignoto. Tale applicazione è stata strutturata per essere giocata o in gruppo o in solitario. Ogni giocatore parte da una posizione iniziale, detta `startPath` e nel primo turno può muoversi in tutte e quattro le direzioni. Ogni giocatore ha una vitalità iniziale impostata a `3`. Tale vita, durante il gioco, può diminuire se si incontra una *bestia* o aumentare se si trovano *misteriose pozioni* fino ad un massimo di 3.

All'interno del Labirinto è nascosta l'uscita. I giocatori dovranno trovarla prima che i minotauri trovino loro se vorranno avere salva la vita. Trovare l'uscita è l'unico modo che si ha per uscire dal Labirinto ed evitare di diventare uno sfizioso spuntino.

Durante la fase di organizzazione, è stato optato di scrivere il programma avvalendoci della Programmazione ad Oggetti. Abbiamo quindi creato due classi `Player` e `Path`, più le relative sottoclassi

`PathWithBeast`, `PathWithPotion` e `PathVictory`.

1.1. DESCRIZIONE DELLE SINGOLE FUNZIONI

Nel programma sono state caricate le seguenti librerie:

- `<iostream>`: per la gestione dei flussi di output e di input;
- `<string>`: per poter utilizzare l'omonimo tipo di dato.

A seguire si trovano le costanti utili per il corretto funzionamento del programma e due `enum`:

`Pathtype` e `Direction`.

Il primo ha cinque possibili valori: `Start`, `Beast`, `Potion`, `Victory` e `Normal`. Questi vengono utilizzati per indicare il tipo di path in cui il giocatore si trova. Ogni tipo di path ha un effetto diverso sul personaggio, quindi è indispensabile poterli distinguere. Il `Pathtype::Victory` viene usato attivamente nel programma per poter far raggiungere la vittoria a tutti i personaggi (vedi sottosezione 1.1.2 a pagina 4).

L'`enum Direction` contiene al suo interno cinque possibili valori, rispettivamente `N`, `E`, `S`, `W` e `none`, che rappresentano i quattro punti cardinali più `none`. Questo `enum` viene utilizzato per rappresentare le direzioni in cui il personaggio del gioco può muoversi e la direzione `none` è stata usata nella costruzione del percorso iniziale, dove i giocatori nascono.

Seguono le funzioni implementate da me:

- `class Player`;
- `class PathVictory`;
- `Path* Player::goToward(direction d)`;
- `Path* Path::extractNewPath(direction d)`;
- `int numberPlayers()`;
- `Player* createPlayers(int num)`.

1.1.1 Classe del giocatore

```
1  // Classe giocatore
2  class Player {
3    bool win;
4    int  vitality ;
5    Path* position;
6    string  playerName;
7    string  listaMosse;
8
9    public:
10     // Costruttore
11     Player(string  pn = "Name") :
12          playerName(pn), vitality(MAX_VITALITY), win(false) {}
13
14     bool getWin() { return win; }
15     void setWin() { win = true; }
```

```
16
17      // get per la vitalità
18      int getVitality() { return vitality; }
19      // Aumenta vitality
20      int increaseVitality();
21      // Diminuisci vitality
22      int decreaseVitality() { if (vitality) --vitality; return vitality; }
23      // Il giocatore è morto?
24      bool isDeath() { return (!vitality); }
25
26      // Impostare il path iniziale
27      Path* setPath(Path* p) { return (position = p); }
28      // Ottenere la posizione
29      Path* getPath() { return position; }
30      // Muovere il giocatore
31      Path* goToward(direction d);
32
33      // get per il nome
34      string getName() { return playerName; }
35      // set per il nome
36      void setName(string newName) { playerName = newName; }
37
38      void aggiornaMosse(direction d);
39      string getMosseFatte() { return listaMosse; }
40  };
```

La classe `class Player` contiene i tre *attributi* che il giocatore deve avere, questi sono:

- `string playerName`: per memorizzare il nome del giocatore;
- `int vitality`: per memorizzare la vita del giocatore;
- `Path* position`: per memorizzare la posizione del giocatore.

Tali attributi hanno visibilità *privata* e verranno modificati con appositi *metodi*. Tutti i metodi e il *costruttore* sono pubblici.

Il costruttore

```
1  Player(string pn = "Name") :
2      playerName(pn), vitality(MAX_VITALITY) {}
```

prende in input una stringa `string pn` che verrà utilizzata per impostare il nome del giocatore. Utilizzando la sintassi

`string pn = "Name"`

si passa in input il nome come *argomento di default*. Il costruttore si occupa anche di inizializzare la vitalità con la costante `MAX_VITALITY` pari a `3`.

Il metodo `getWin()` restituisce il valore dell'attributo `win`:

- `win = true` se il giocatore ha raggiunto l'uscita;
- `win = false` altrimenti.

Il metodo `setWin()` imposta l'attributo booleano `win` a `true` in modo da indicare che il giocatore ha raggiunto l'uscita.

Il metodo `getVitality()` si occupa solamente di restituire la vitalità del giocatore all'esterno della classe perché è un attributo con visibilità privata.

Il metodo `increaseVitality()` per prima cosa controlla se la vitalità del giocatore sia strettamente minore della vitalità massima e in tal caso la aumenta di uno, altrimenti stampa un messaggio e restituisce la vitalità senza aumentarla.

```
1 // Incrementa di 1 la  vitalit à
2 int Player:: increaseVitality () {
3   if ( vitality  < MAX_VITALITY) return (++vitality);
4   else {
5     cout << "Vitalità massima." << endl;
6     return vitality;
7   }
8 }
```

Il metodo `decreaseVitality()` fa esattamente l'opposto del precedente: se la vitalità non è nulla, la diminuisce di uno.

Il metodo `bool isDeath() { return (!vitality); }` restituisce un valore booleano che controlla che il giocatore sia morto o meno.

Per iniziare il gioco tutti i giocatori devono essere posizionati nel percorso iniziale e questo compito spetta al metodo `setPath(Path* p)`, che prenderà in input il path `startPath` che verrà creato nel `main`.

Dato che l'attributo `position` di `Player` è privato, per poterlo leggere anche all'esterno della classe è stato necessario costruire il metodo `getPath()`, il cui unico compito è quello di restituire il `Path` di dove si trova il giocatore.

1.1.2 Spostamento del giocatore

Il metodo `goToward(direction d)` viene utilizzato per far muovere il giocatore lungo un percorso in una determinata direzione `d` passata in input.

Per iniziare, il metodo imposta la destinazione del giocatore tramite il metodo `getPath()`. Se la destinazione è una nuova posizione, ovvero una posizione che nessun giocatore ha mai esplorato prima, il metodo controlla se la posizione sia disponibile per essere raggiunta tramite il metodo `isAvailable()`.

Se la posizione è disponibile, viene creato un nuovo percorso che connette la posizione corrente del giocatore alla nuova posizione tramite il metodo `extractNewPath()`. Viene poi eseguita un'azione sul giocatore tramite il metodo `actionOnPlayer()`, il cui effetto sul giocatore dipende dal tipo di path che si scopre. Infine, viene aggiornata la mossa del giocatore tramite il metodo `aggiornaMosse()` per avere uno storico del cammino intrapreso.

Se la destinazione risulta essere stata già esplorata in precedenza, il metodo controlla se la destinazione sia una posizione di vittoria tramite il metodo `getPathType()`. In caso affermativo, viene eseguita l'azione sul giocatore tramite il metodo `actionOnPlayer()` (opportune spiegazioni nella sottosezione 1.1.4) (in poche parole, il giocatore vince e smette di giocare). Se non avessimo inserito questo controllo, ogni giocatore che prova a raggiungere il percorso con la vittoria dopo che questo viene scoperto non riuscirebbe a vincere perché non verrebbe eseguita l'azione sul giocatore. Se la destinazione è stata già esplorata e non è l'uscita dal labirinto, viene aggiornata solo la posizione del giocatore.

La funzione restituisce infine un puntatore all'oggetto `destination`.

```
1  // sposta il giocatore nella nuova posizione
2  Path* Player::goToward(direction d) {
3    // Imposto la destinazione
4    auto destination = position->getPath(d);
5    // Se la posizione è nuova
6    if (position->isNew(d)) {
7      if(position->isAvailable()){
8        // creo un nuovo path
9        position = position->extractNewPath(d);
10       // eventuale azione sul giocatore
11       position->actionOnPlayer(this);
12
13       // Aggiorno mossa
14       aggiornaMosse(d);
15
16     }
17     else {
18         cout << "Direzione non disponibile." << endl;
19     }
```

```
20    }
21    else if (destination->getPathType() == Pathtype::victory)
22      destination->actionOnPlayer(this);
23    // Altrimenti aggiorno solo la posizione
24    else {
25      position = destination;
26      cout << "Direzione già esplorata" << endl;
27
28      // Aggiorno mossa
29      aggiornaMosse(d);
30    }
31    return destination;
32  }
```

Per leggere e modificare il nome del giocatore sono stati implementati i metodi `getName()` e `setName()` rispettivamente.

Per concludere la lista, si hanno anche i metodi

`aggiornaMosse(direction d)` e `getMosseFatte()`.

Il secondo consente di leggere la stringa `listaMosse` che ha visibilità privata. La prima, invece, prende in input la direzione in cui il giocatore si sta spostando e la aggiunge in coda alla lista già creata. All'inizio del gioco tale stringa sarà vuota e si andrà ad aggiornare ad ogni spostamento. Abbiamo deciso di implementare questa funzionalità per agevolare i giocatori a ritrovare la strada già percorsa per raggiungere tutti quanti la vittoria.

```
1   // Crea stringa con mosse fatte
2   void Player::aggiornaMosse(direction d) {
3     // Aggiorno mossa
4     if (d == direction::N)
5       listaMosse = listaMosse.append("N ");
6     else if (d == direction::E)
7       listaMosse = listaMosse.append("E ");
8     else if (d == direction::S)
9       listaMosse = listaMosse.append("S ");
10    else if (d == direction::W)
11      listaMosse = listaMosse.append("W ");
12  }
```

1.1.3 Creazione di un nuovo percorso

Il metodo `Path* extractNewPath(Direction d)` prende in input una direzione e, usando i metodi booleani `isBeast`, `isPotion` e `isVictory` stampa

alcuni messaggi utili all'utente per capire in quale percorso è capitato e gli effetti che questo ha e crea un nuovo path del corrispondente tipo.

1.1.4 Sottoclasse con vittoria

```
class PathVictory : public Path {
  public:
  // Costruttore
  PathVictory(Path* from, direction d) :
    Path("Victory", "Victory is Here", from, d, Pathtype::victory) {}

  // Azione sul giocatore : il giocaotre ha vinto
  void actionOnPlayer(Player* p) {
    while (!p->isDeath()) {
      p->decreaseVitality();
    }
    // Assegno all'attributo "win" del giocatore il valore "true"
    p->setWin();

    // Diminuisco giocatori attivi
    playerAlive -= 1;
    cout << "Complimenti!" << endl;
    cout << "Sei arrivato all'uscita." << endl;
    cout << "I minotauri non ti hanno preso." << endl;
  }
};
```

La classe `PathVictory` è una sottoclasse di `Path`. Il suo costruttore richiede due argomenti: un puntatore a un oggetto `Path`, chiamato `from`, e un oggetto di tipo `Direction`, chiamato `d`. Questo nuovo costruttore richiama il costruttore della sovraclasse `Path` con quattro argomenti: la stringa `Victory` come nome del percorso, la stringa `Victory is Here` come descrizione, il puntatore all'oggetto `from`, la direzione `d` e il tipo di path `Pathtype::Victory` che indica che questo è un path di tipo vittoria.

La classe contiene anche una metodo chiamato `actionOnPlayer` che prende in input un puntatore ad un oggetto di classe `Player` chiamato `p`. Questo metodo decrementa la vitalità del giocatore fino a farla arrivare a zero, dopodiché viene richiamato il metodo `setWin()` della classe `Player` per impostare l'attributo `win` del giocatore su `true`. Per finire, diminuisce il numero di giocatori in vita di uno e stampa un messaggio di congratulazioni sulla console.

1.1.5 Inserire il numero di giocatori

La funzione `numberPlayers()` si occupa solo di chiedere con quanti personaggi si vuole giocare. Una volta che l'utente inserisce la propria scelta, il programma controlla che tale numero sia minore del numero massimo di giocatori consentito. Se tale valore viene sforato, si richiede un nuovo input.

Questa funzione non chiede in input nulla e restituisce un intero.

```
// Inserimento numero giocatori
int numberPlayers() {
  int num;
  // Controllo sul numero di giocatori inseriti
  do {
    cout << "Attenzione! Massimo numero di giocatori consentiti: ";
    cout << MAX_PLAYER << endl;
    cout << "Inserisci il numero di giocatori: ";
    cin >> num;
  } while (num > MAX_PLAYER);

  return num;
}
```

1.1.6 Creare i giocatori

La funzione `createPlayers()` chiede in input il numero di personaggi con cui si vuole giocare e crea un array di tipo `Player` che contiene il numero di giocatori passato in input.

All'interno del ciclo for, viene richiesto all'utente di inserire il nome del giocatore i-esimo e viene salvato input nella variabile `pName` di tipo stringa. Successivamente, il nome viene assegnato al giocatore i-esimo utilizzando il metodo `setName`.

Infine, la funzione restituisce l'array di giocatori creato, che può essere utilizzato in altre parti del programma.

È importante notare che l'array creato con l'operatore `new` deve essere eliminato alla fine del programma tramite l'operatore `delete` liberare la memoria allocata.

```
// Per creare l'array di giocatori
Player* createPlayers(int num) {
  // Creo array di giocatori
  Player* players = new Player[num];

  // Riassegno il nome ad ogni giocatore
```

```
7   for (int i = 0; i < num; i++) {
8     // Scelta nome al giocatore i-esimo
9     string pName;
10    cout << "Inserisci il nome del giocatore " << i + 1 << ": ";
11    cin >> pName;
12
13    // Assegno al giocatore i-esimo il nome
14    players[i].setName(pName);
15  }
16
17  // Restituisco l'array di giocatori creato
18  return players;
19 }
```

1.1.7 Main

Il `main` inizia con una descrizione ironica del programma. Seguono le dichiarazioni delle variabili che verranno utilizzate in seguito.

L'utente, come prima decisione, deve impostare il numero di personaggi con cui vuole giocare. Questo viene fatto con la funzione `numberPlayers()`. In seguito, con la funzione `createPlayers()`, si crea un array di tipo `Player` in cui ogni suo elemento è un oggetto `Player`.

Per poter giocare c'è bisogno che esista almeno un giocatore in vita (o che non abbia raggiunto ancora l'uscita). Per questo motivo è stata introdotta la variabile `playerAlive`, a cui viene assegnato lo stesso numero di giocatori creati.

Si prosegue creando l'oggetto `startPath`, che rappresenta il `Path` dove tutti i giocatori nascono e dal quale inizieranno a muoversi. Per informare tutti i giocatori di quale è l'origine è stato inserito un ciclo `for` al cui interno è presente il metodo `setPath` e come suo argomento l'oggetto `startPath`.

Una volta creati tutti i giocatori e il path iniziale e aver settato la posizione iniziale ad ogni giocatore è il momento di iniziare a giocare.

Il ciclo `do-while` si occupa di mandare avanti i turni finché esiste almeno un giocatore in vita e che non abbia raggiunto l'uscita.

All'inizio di ogni turno viene mandato in stampa il corrispettivo numero che viene incrementato di volta in volta.

Per far giocare ogni personaggio è stato inserito il seguente ciclo `for`. All'interno del ciclo, ogni comando viene ripetuto per ogni giocatore in vita.

Dato che quando un giocatore muore o raggiunge l'uscita non viene eliminato dall'array è stato necessario introdurre anche un controllo sulla vitalità

dell'i-esimo giocatore per evitare che continuino a giocare anche i cadaveri dei personaggi morti o chi ha raggiunto l'uscita.

Il programma continua stampando dei messaggi informativi, quali:

- il nome del giocatore a cui tocca svolgere il turno;
- la vitalità che gli è rimasta;
- le mosse che ha fatto fino a qual momento.

In particolare l'ultima informazione è di vitale importanza per consentire ad ogni giocatore di poter ritornare sui propri passi per raggiungere l'uscita una volta che questa viene scoperta da un compagno.

Il gioco prosegue impostando `controlMove = false` e viene chiesto al giocatore di turno di inserire la direzione in cui vuole andare. Se il giocatore inserisce una delle quattro lettere maiuscole proposte, allora si esce dal ciclo `while`, altrimenti viene mandato in output un messaggio di errore e si richiede di inserire una nuova lettera. Abbiamo scelto di impostare `controlMove = false` subito dopo il `do` per evitare di ripetere tale comando dopo ogni assegnazione di `dir`. Tale variabile è necessaria per "convertire" il `char` che inserisce l'utente in un tipo `Direction`. Questa fase è molto importante perché i metodi che verranno utilizzati a seguire non accettano `char`, ma tipi `Direction`.

A questo punto il giocatore si deve spostare e questo viene fatto attraverso il metodo `goToward`, di cui ho già discusso nella sottosezione 1.1.2 a pagina 4.

L'ultima parte del `main` si occupa di capire se un personaggio è morto nel suo turno oppure no. In caso il giocatore sia ancora vivo non succede nulla, ma in caso di morte viene stampato un messaggio che informa dell'accaduto e il numero di giocatori viene diminuito di uno.

Come ricordato nella sottosezione 1.1.6 a pagina 8, si utilizza `delete` per eliminare l'array di giocatori.

1.1.8 Metodi implementati da Sara

- `class Path`: Questa classe contiene tutti i metodi e gli attributi che servono per descrivere correttamente il territorio. Ogni path ha un nome, un testo e un tipo `Pathtype` che servono a differenziare i vari path e quattro puntatori per creare i collegamenti fra i vari percorsi. Tutti questi attributi sono privati.
- È stato poi implementato il costruttore della classe che assegna ad ogni percorso un nome, un testo e il suo tipo. Assumiamo di spostarci verso nord. Si assegna al puntatore `south` del nuovo path l'indirizzo di

memoria del path da cui si proviene e al puntatore `north` del path di provenienza viene assegnato l'indirizzo di memoria del nuovo path.

- `getName`, `getText` e `getPathType` servono unicamente per poter leggere il nome, il testo e il tipo del percorso anche all'esterno della classe.

- i tre metodi `isBeast`, `isPotion` e `isVictory` entrano in gioco quando il personaggio esplora un nuovo percorso e servono a dire se il nuovo path è di tipo bestia, pozione, vittoria o normale.

- `Path* getPath(Direction d)` prende in input una direzione e restituisce il puntatore a quella direzione, se tale puntatore non esiste restituisce `NULL`.

- `Path* isAvailable()` restituisce un puntatore se la direzione è disponibile, nulla altrimenti.

- Il metodo `bool isNew(Direction d)` restituisce `false` se il territorio è stato già esplorato, `true` altrimenti.

- L'ultimo metodo della classe `Path` è

  `virtual void actionOnPlayer(Player* p)`.

  Questo metodo ha un comportamento differente a seconda del tipo del path in cui si trova. Se il path è di tipo `normal`, esso non ha alcun effetto, negli altri casi il suo comportamento è stato definito nelle sottoclassi di `Path`.

- le due sottoclassi

  `class PathWithBeast` e `class PathWithPotion`

  sono sottoclassi di `Path`. In queste viene definito il metodo `actionOnPlayer` con l'azione che deve fare:

  - se c'è una bestia, deve decrementare di 1 la vita del giocatore;
  - se c'è una pozione deve aumentare di 1 la vita.

2. CODICE SORGENTE

Il seguente codice è stato leggermente modificato solo per consentire al testo di essere interamente visibile.

```
#include <iostream>
#include <string>

using namespace std;

// Costanti
#define MAX_PLAYER 6
#define MAX_VITALITY 3
#define PROB_BEAST 10
#define PERC_DIRECTION 50
#define PROB_POTION 10
#define PROB_VICTORY 5

#define DPRINT(VAR) cout << #VAR << " = " << (VAR) << endl;

// Definizione di Pathtype
using Pathtype = enum { Start, Beast, Potion, Victory, Normal };
// Definizione di Direction
using Direction = enum { N, E, S, W, none };

// Variabile che conta il numero di giocatori in vita
int playerAlive;

// Dichiaro la classe altrimenti il metodo "actionOnPlayer" della classe
// "Path" non funziona
class Player;

// SARA
// Classe percorso
class Path {
  string pathName;
  string text;
  Pathtype type;
  Path* north = NULL;
  Path* south = NULL;
  Path* east = NULL;
  Path* west = NULL;

```

```
39    public:
40      // Costruttore
41      Path(const string &name = "Name", const string &t = "Text",
42           Path*from = NULL, Direction d = Direction::none,
43           Pathtype p = Pathtype::Start);
44
45      // Per ottenere il "name"
46      string getName() const { return pathName; }
47      // Per ottenere il "text"
48      string getText() const { return text; }
49      // Per ottenere il puntatore
50      Path* getPath(Direction d);
51      // Il percorso è aperto?
52      Path* isAvailable();
53      // Cosa c'è sul territorio?
54      Path* extractNewPath(Direction d);
55      // C'è una bestia?
56      bool isBeast() { return ((rand() % 100) < PROB_BEAST); }
57      // C'è una pozione?
58      bool isPotion() { return ((rand() % 100) < PROB_POTION); }
59      // C'è la vittoria?
60      bool isVictory() { return ((rand() % 100) < PROB_VICTORY); }
61      // Il territorio è stato scoperto?
62      bool isNew(Direction d);
63      // Azione sul giocatore : creata nelle sottoclassi
64      virtual void actionOnPlayer(Player* p) { ; }
65      Pathtype getPathType() { return type; }
66  };
67
68  // LUCA
69  // Classe giocatore
70  class Player {
71    bool win;
72    int vitality;
73    Path* position;
74    string playerName;
75    string moveList;
76
77    public:
78      // Costruttore
79      Player(string pn = "Name") :
80        playerName(pn), vitality(MAX_VITALITY), win(false) {}
81
```

```
82      // get vittoria
83      bool getWin() { return win; }
84      // Il giocatore ha vinto
85      void setWin() { win = true; }
86
87      // get per la vitalità
88      int getVitality() { return vitality; }
89      // Aumenta vitality
90      int increaseVitality();
91      // Diminuisci vitality
92      int decreaseVitality() { if (vitality) --vitality; return vitality; }
93      // Il giocatore è morto?
94      bool isDeath() { return (!vitality); }
95
96      // Impostare il path iniziale
97      Path* setPath(Path* p) { return (position = p); }
98      // Ottenere la posizione
99      Path* getPath() { return position; }
100     // Muovere il giocatore
101     Path* goToward(Direction d);
102
103     // get per il nome
104     string getName() { return playerName; }
105     // set per il nome
106     void setName(string newName) { playerName = newName; }
107
108     // Aggiornare moveList
109     void aggiornaMosse(Direction d);
110     // Ottenere moveList
111     string getMosseFatte() { return moveList; }
112 };
113
114 // SARA
115 class PathWithBeast : public Path {
116   public:
117     // Costruttore
118     PathWithBeast(Path* from, Direction d) :
119       Path("Beast", "Beast is Here", from, d, Pathtype::Beast) {}
120     // Azione sul giocatore : diminuisce vitality
121     void actionOnPlayer(Player* p) { p->decreaseVitality(); }
122 };
123
124 // SARA
```

```
125 class PathWithPotion : public Path {
126   public:
127     // Costruttore
128     PathWithPotion(Path* from, Direction d) :
129       Path("Potion", "Potion is Here", from, d, Pathtype::Potion) {}
130     // Azione sul giocatore : aumenta vitality
131     void actionOnPlayer(Player* p) { p->increaseVitality(); }
132 };
133
134 // LUCA
135 // Sottoclasse vittoria
136 class PathVictory : public Path {
137   public:
138     // Costruttore
139     PathVictory(Path* from, Direction d) :
140       Path("Victory", "Victory is Here", from, d, Pathtype::Victory) {}
141
142     // Azione sul giocatore : il giocatore ha vinto
143     void actionOnPlayer(Player* p) {
144       while (!p->isDeath()) {
145         p->decreaseVitality();
146       }
147       // Assegno all'attributo "win" del giocatore il valore "true"
148       p->setWin();
149
150       // Diminuisco giocatori attivi
151       playerAlive -= 1;
152       cout << "Complimenti!" << endl;
153       cout << "Sei arrivato all'uscita." << endl;
154       cout << "I minotauri non ti hanno preso." << endl;
155     }
156 };
157
158 // Funzioni
159 int numberPlayers();
160 Player* createPlayers(int num);
161
162 // SARA + LUCA
163 int main() {
164   cout <<"\n\n\n";
165
166   cout << "\u2620\uFE0F  \u2620\uFE0F  \u2620\uFE0F  ";
167   cout << "BENVENUTI AVVENTURIERI! ";
```

```
168   cout << "\u2620\uFE0F␣␣\u2620\uFE0F␣␣\u2620\uFE0F" << endl;
169
170   cout << "Il␣sole␣splende,␣gli␣ uccellini ␣cantano␣e␣";
171   cout << "i␣minotauri␣vi␣rincorrono!␣";
172   cout << "Vi␣starete␣chiedendo␣perché vi abbiamo riuniti qui oggi.\n"<<endl;
173
174   cout << "Prima␣di␣spiegarvelo␣è debita una presentazione. ";
175   cout << "Io␣sono␣il␣Giuallare␣Messaggero␣e␣porto␣le␣veci␣del␣";
176   cout << "grande␣Mefisto!";
177   cout << "␣Ma␣bando␣alle␣ciance␣e␣ciancio␣alle␣bande.\n" << endl;
178
179   cout << "Vi␣ritrovate␣nel␣Grande␣Labirinto␣di␣Baalzebul,␣";
180   cout << "Signore␣delle␣Mosche.␣";
181   cout << "Un␣luogo␣pieno␣di␣pericolose␣creature␣mistiche,␣";
182   cout "pozioni␣dagli␣ effetti ␣straordinari ,␣";
183   cout << "scale␣incantate␣e␣tesori␣perduti.\n"  << endl;
184   cout << "Per␣riuscire␣a␣fuggire␣avete␣solo␣un␣modo:␣";
185   cout << "trovare␣l'uscita␣segreta␣situata␣nella␣sala␣del␣tesoro,␣";
186
187   cout << "Ricordate␣però: 'l'uinone fa  la  forza '.  ";
188   cout << "State␣attenti␣a␣non␣lasciare␣i␣vostri␣compagni␣indietro␣";
189   cout << "o␣le␣conseguenze␣per␣tutti␣voi␣saranno␣drastiche.";
190   cout << "Un␣ultimo␣avvertimento:␣'attenzione␣a␣dove␣mettete␣i␣piedi,␣";
191   cout << "alle␣scale␣piace␣cambiare'.\n" << endl;
192
193   cout << "Ma␣perché dovreste fare tutto questo? ";
194   cout << "Ovviamente␣per␣il␣diletto␣del␣nostro␣Padrone␣e␣";
195   cout << "Signore␣Assoluto!" << endl;
196
197   for (int t = 0; t < 30; t++) {
198     cout << "\u2620\uFE0F␣␣";
199   }
200   cout << "\n\n\n";
201
202
203
204   string newPlayerName;
205   char move;
206   Direction dir;
207   int turnNumber = 1;
208   bool controlMove;
209
210   // Imposto numero di giocatori
```

```
211   int num_players = numberPlayers();
212   // Creo array di giocatori
213   Player* players = createPlayers(num_players);
214   // Assegno il numero di giocatori in vita
215   playerAlive = num_players;
216
217   // Creo l'origine
218   Path* startPath = new Path("Start Path", "Start Path", NULL,
219                              Direction::none, Pathtype::Start);
220   // Posiziono tutti i giocatori sull'origine
221   for (int j = 0; j < num_players; j++) {
222     // Impostiamo la posizione iniziale del giocatore
223     players[j].setPath(startPath);
224   }
225
226   // Svolgimento dei vari turni
227   // Finché c'é almeno un giocatore attivo
228   do {
229     cout << "\n%-~-~-~-~-~-~-~-~-~-~-~-~-~-%\n\n\n" << endl;
230     cout << "Turno " << turnNumber << "." << endl;
231
232     // Per far giocare tutti i giocatori
233     for (int i = 0; i < num_players; i++) {
234       // Controllo se il giocatore è morto (o ha vinto)
235       if (players[i].getVitality() > 0) {
236         cout << "\n";
237         cout << "Tocca al giocatore '" << players[i].getName();
238         cout << "' (vitalità: " << players[i].getVitality() << ")." << endl;
239         cout << "Mosse fatte: " << players[i].getMosseFatte() << endl;
240
241         do {
242           controlMove = false;
243           // Il giocatre inserisce la mossa da tastiera
244           cout << "Inserisci 'N' per Nord, 'S' per Sud, ";
245           cout << "'E' per Est E 'W' per Ovest: ";
246           cin >> move;
247           // Assegniamo la direzione scelta
248           if (move == 'N')
249             dir = Direction::N;
250           else if (move == 'E')
251             dir = Direction::E;
252           else if (move == 'S')
253             dir = Direction::S;
```

```
254             else if (move == 'W')
255               dir = Direction::W;
256             else {
257               cout << "Mossa non disponibile" << endl;
258               controlMove = true;
259             }
260           } while (controlMove);
261
262           // Il giocatore si sposta
263           players[i].goToward(dir);
264
265           // Controllo che il giocatore sia morto nel turno corrente
266           // Se il giocatore ha vinto, non gioca (e quindi non può morire)
267           if ((players[i].isDeath()) && (!players[i].getWin())) {
268             cout << "Il giocatore " << players[i].getName();
269             cout << " è morto :(" << endl;
270             // In tal caso diminuisco il numero di giocatori attivi
271             playerAlive -= 1;
272           }
273         }
274       }
275
276       // Contatore turni
277       turnNumber += 1;
278     } while (playerAlive != 0);
279
280     // Alla fine del programma elimino l'array di giocatori
281     delete[] players;
282     cout << "\n";
283   }
284
285   // SARA
286   // Costruttore classe Path
287   Path::Path(const string &name, const string &t, Path* from, Direction d, Pathtype p) {
288     pathName = name;
289     text = t;
290     type = p;
291     switch (d) {
292       case Direction::N:
293         // Se la direzione è north, provengo da south.
294         // From punta a nord perché nella prossima mossa io sarò "a north"
295         south = from;
296         from->north = this;
```

```
297         break;
298       case Direction::E:
299         west = from;
300         from->east = this;
301         break;
302       case Direction::S:
303         north = from;
304         from->south = this;
305         break;
306       case Direction::W:
307         east  = from;
308         from->west = this;
309         break;
310       case Direction::none:
311         break;
312     }
313 }
314
315 // SARA
316 // Restituisce un puntatore se la direzione è disponibile, nulla altrimenti
317 Path* Path::isAvailable() {
318   Path pathAvailable;
319   // assegno la posizione di memoria di pathAvailable in una costante
320   // puntatore "AVAILABLE"
321   Path* const AVAILABLE = &pathAvailable;
322   return (((rand() % 100) < PERC_DIRECTION) ? AVAILABLE : NULL);
323 }
324
325 // SARA
326 Path* Path::getPath(Direction d) {
327   // Se d == <punto_cardinale>,
328   //  restituisce  il  puntatore a quel <punto_cardinale>
329   if (d == Direction::N) return north;
330   else if (d == Direction::E) return east;
331   else if (d == Direction::S) return south;
332   else if (d == Direction::W) return west;
333   else return NULL;
334 }
335
336 // LUCA
337 // sposta il giocatore nella nuova posizione
338 Path* Player::goToward(Direction d) {
339   // Imposto la destinazione
```

```
340   auto destination = position->getPath(d);
341   // Se la posizione è nuova
342   if (position->isNew(d)) {
343     if(position->isAvailable()){
344       // creo un nuovo path
345       position = position->extractNewPath(d);
346       // eventuale azione sul giocatore
347       position->actionOnPlayer(this);
348       //DPRINT(position);
349
350       // Aggiorno mossa
351       aggiornaMosse(d);
352
353     }
354     else {
355         cout << "Direzione non disponibile." << endl;
356         //DPRINT(position);
357     }
358   }
359   else if (destination->getPathType() == Pathtype::Victory)
360     destination->actionOnPlayer(this);
361   // Altrimenti aggiorno solo la posizione
362   else {
363     position = destination;
364     cout << "Direzione già esplorata" << endl;
365
366     // Aggiorno mossa
367     aggiornaMosse(d);
368   }
369   return destination;
370 }
371
372 // LUCA
373 // Incrementa di 1 la vitalità
374 int Player:: increaseVitality () {
375   if ( vitality < MAX_VITALITY) return (++vitality);
376   else {
377     cout << "Vitalità massima." << endl;
378     return vitality;
379   }
380 }
381
382 // LUCA
```

```
383 // Crea stringa con mosse fatte
384 void Player::aggiornaMosse(Direction d) {
385   // Aggiorno mossa
386   if (d == Direction::N)
387     moveList = moveList.append("N ");
388   else if (d == Direction::E)
389     moveList = moveList.append("E ");
390   else if (d == Direction::S)
391     moveList = moveList.append("S ");
392   else if (d == Direction::W)
393     moveList = moveList.append("W ");
394 }
395
396 // SARA
397 // Restituisce "false" se il territorio è stato già esplorato,
398 // "true" altrimenti
399 bool Path::isNew(Direction d) {
400   if (d == Direction::N) {
401     // Se il puntatore è nullo, la posizione è nuova
402     if (north == NULL) return true;
403     // Se il puntatore non è vuoto, la posizione è stata già esplorata
404     else return false;
405   }
406   else if (d == Direction::E) {
407     if (east == NULL) return true;
408     else return false;
409   }
410   else if (d == Direction::S) {
411     if (south == NULL) return true;
412     else return false;
413   }
414   else if (d == Direction::W) {
415     if (west == NULL) return true;
416     else return false;
417   }
418   else return false;
419 }
420
421 // LUCA
422 Path* Path::extractNewPath(Direction d) {
423   // Creo un nuovo path
424
425   // Se isBeast() = true
```

```
426   if (isBeast()) {
427     // creo un nuovo Path con bestia
428     cout << "Scoperto percorso con \U0001f4a3 minotauro ";
429     cout << "\U0001f4a3." << endl;
430     cout << "Oh no!! Sei stato attaccato dal minotauro!";
431     cout << "La tua vitalità è diminuita di 1." << endl;
432
433     // Creo Path con bestia
434     return new PathWithBeast(this, d);
435   }
436   // Analogamente sotto
437   else if (isPotion()) {
438     cout << "Scoperto percorso con \U0001f31f pozione \U0001f31f."<<endl;
439     cout << "Che fortuna! Hai trovato una pozione!";
440     cout << "La tua vitalità è aumentata di 1." << endl;
441
442     return new PathWithPotion(this, d);
443   }
444   else if (isVictory()) {
445
446     cout << "Scoperto percorso con \U0001f308 vittoria \U0001f308."<<endl;
447
448     return new PathVictory(this, d);
449   }
450   else {
451     cout << "Scoperto percorso normale." << endl;
452     cout << "Ti è andata bene... per ora..." << endl;
453
454     return new Path("Walk", "walk freely", this, d, Pathtype::Normal);
455   }
456 }
457
458 // LUCA
459 // Inserimento numero giocatori
460 int numberPlayers() {
461   int num;
462   // Controllo sul numero di giocatori inseriti
463   do {
464     cout << "Attenzione! Massimo numero di giocatori consentiti: ";
465     cout << MAX_PLAYER << endl;
466     cout << "Inserisci il numero di giocatori: ";
467     cin >> num;
468   } while (num > MAX_PLAYER);
```

```
469
470   return num;
471 }
472
473 // LUCA
474 // Per creare l'array di giocatori
475 Player* createPlayers(int num) {
476   // Creo array di giocatori
477   Player* players = new Player[num];
478
479   // Riassegno il nome ad ogni giocatore
480   for (int i = 0; i < num; i++) {
481     // Scelta nome al giocatore i-esimo
482     string pName;
483     cout << "Inserisci il nome del giocatore " << i + 1 << ": ";
484     cin >> pName;
485
486     // Assegno al giocatore i-esimo il nome
487     players[i].setName(pName);
488   }
489
490   // Restituisco l'array di giocatori creato
491   return players;
492 }
```

3. RISULTATO

Nelle pagine seguenti vengono riportate le foto del terminale.

4. OSSERVAZIONI E CONCLUSIONI

- La prima miglioria che salta subito all'occhio riguarda il metodo `isAvailable`. Quando inseriamo una direzione, ma questa non risulta disponibile, il giocatore perde il suo turno senza muoversi, ma se si insiste a voler andare in quella precisa direzione, prima o poi questa risulterà accessibile.

- Un altra osservazione importante sta nella plancia di gioco. Come mostrato nelle Figure 2-4, nonostante il giocatore `B` sia comunque finito

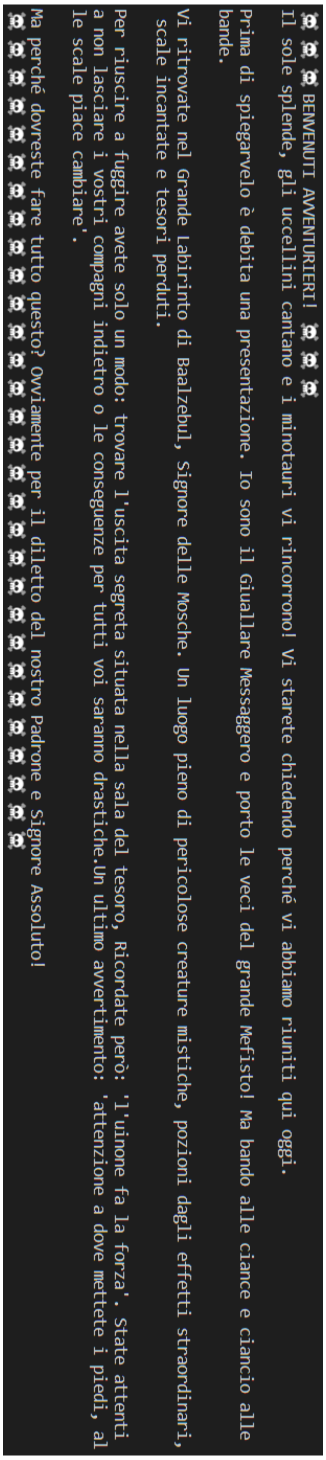

Figura 1: Descrizione del gioco.

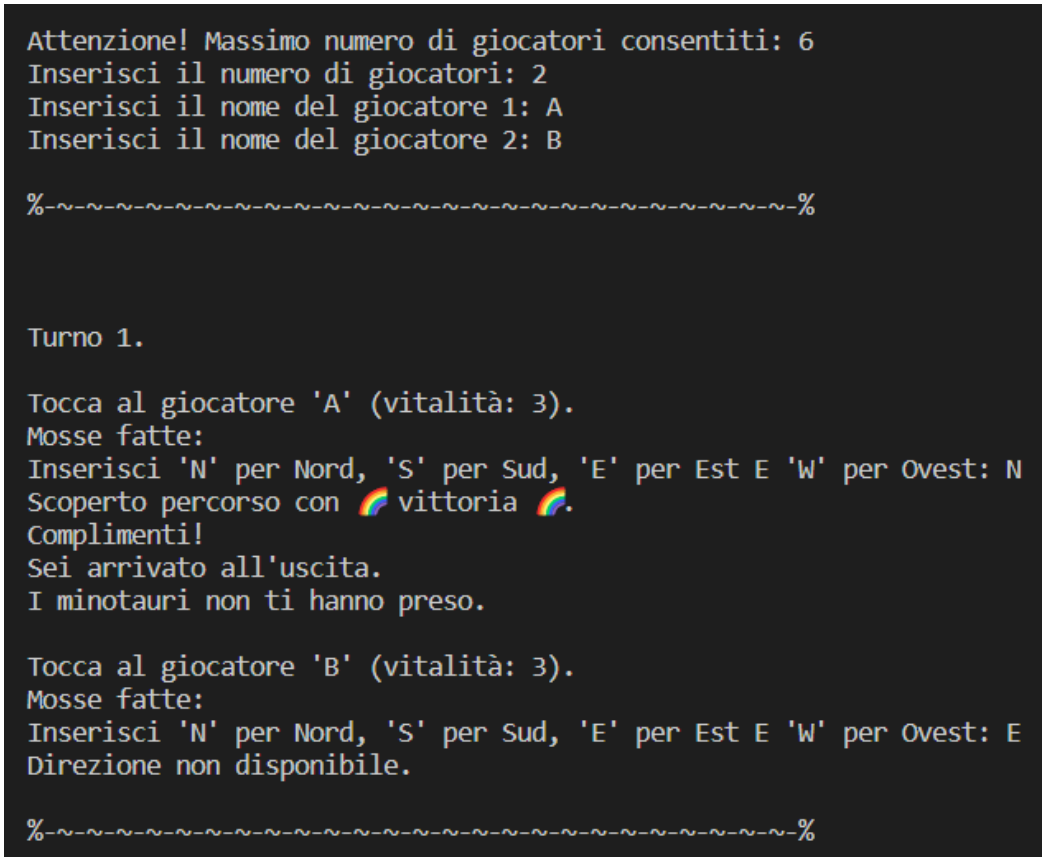

Figura 2: Path con vittoria.

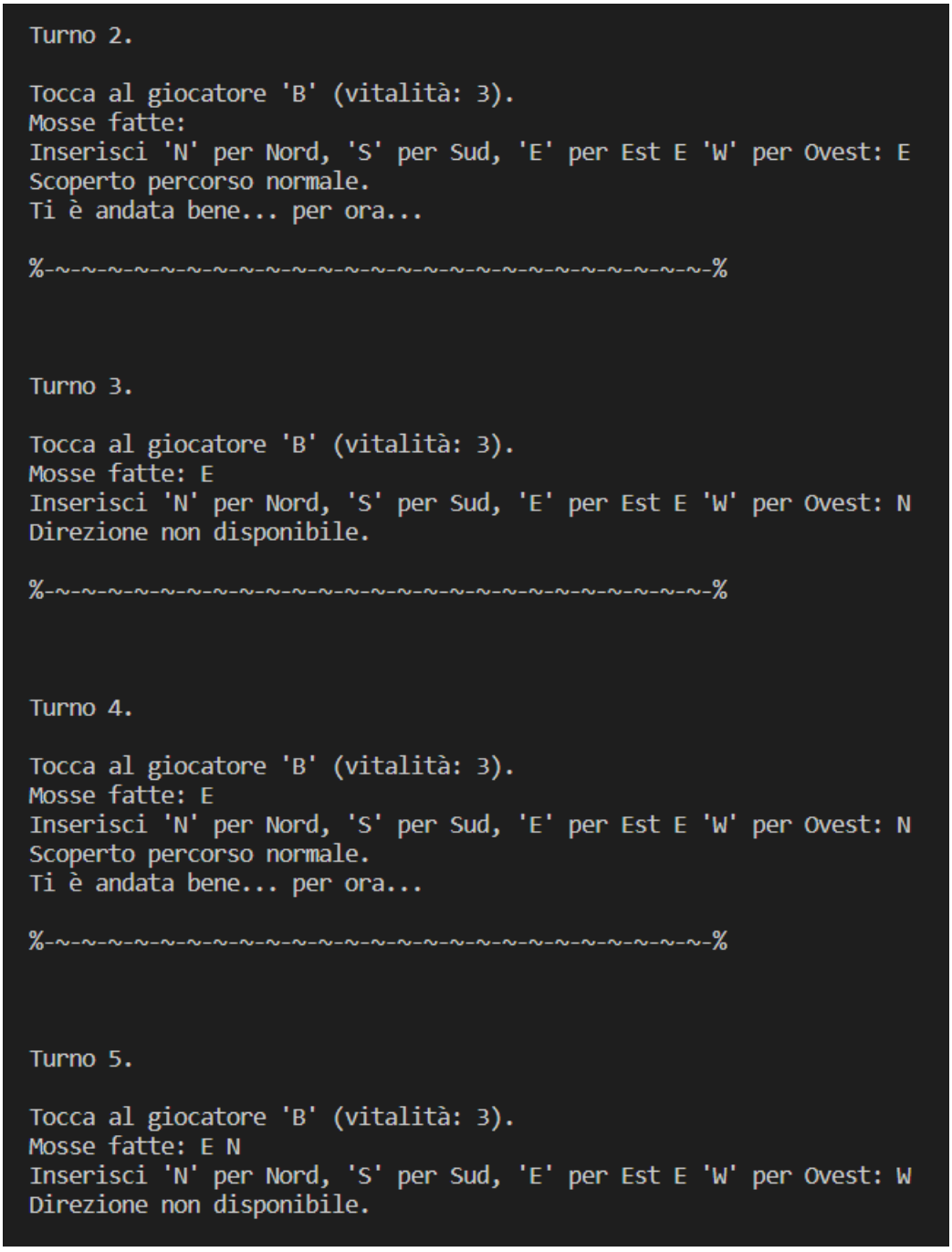

Figura 3: Svolgimento del gioco.

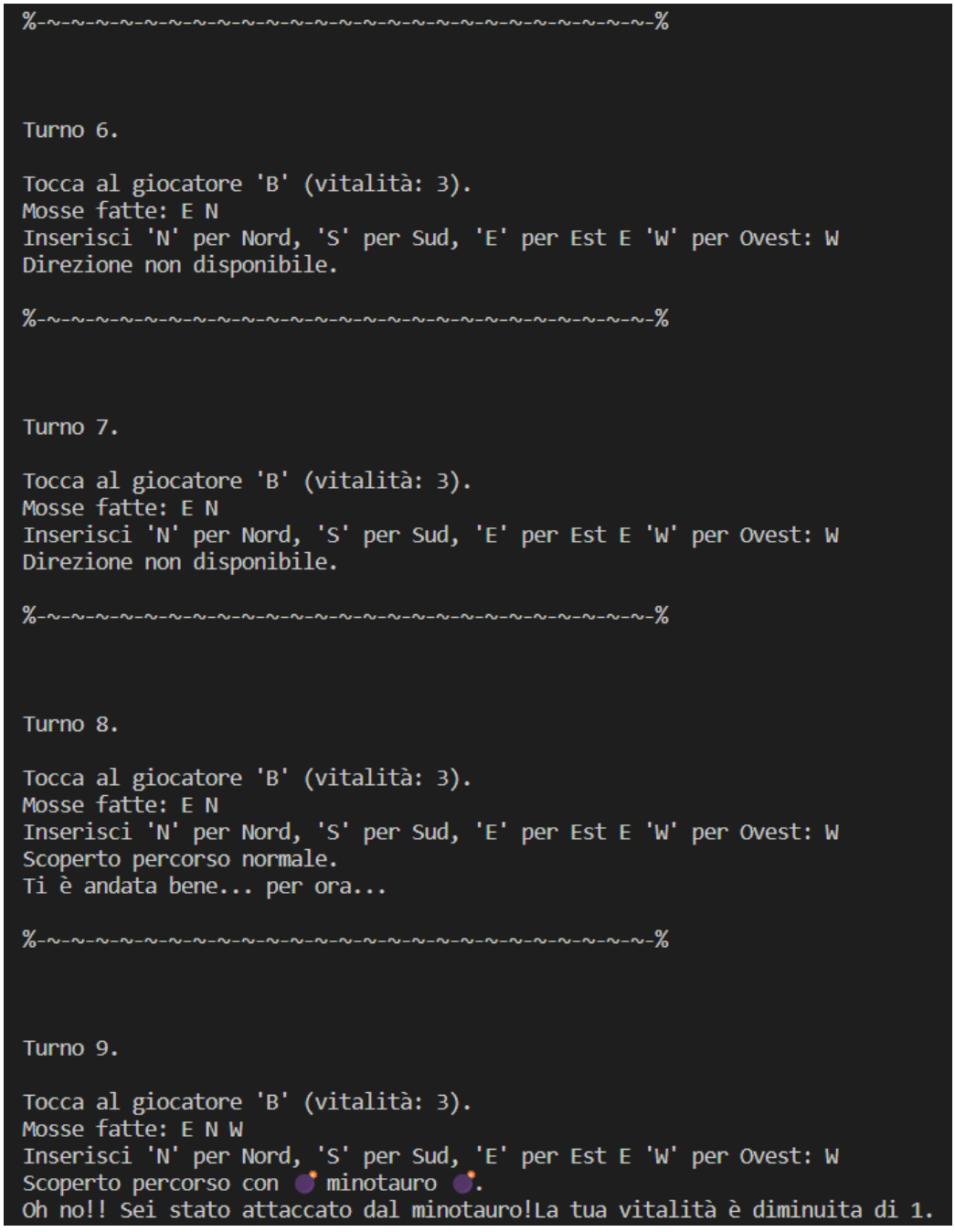

Figura 4: Path con minotauro.

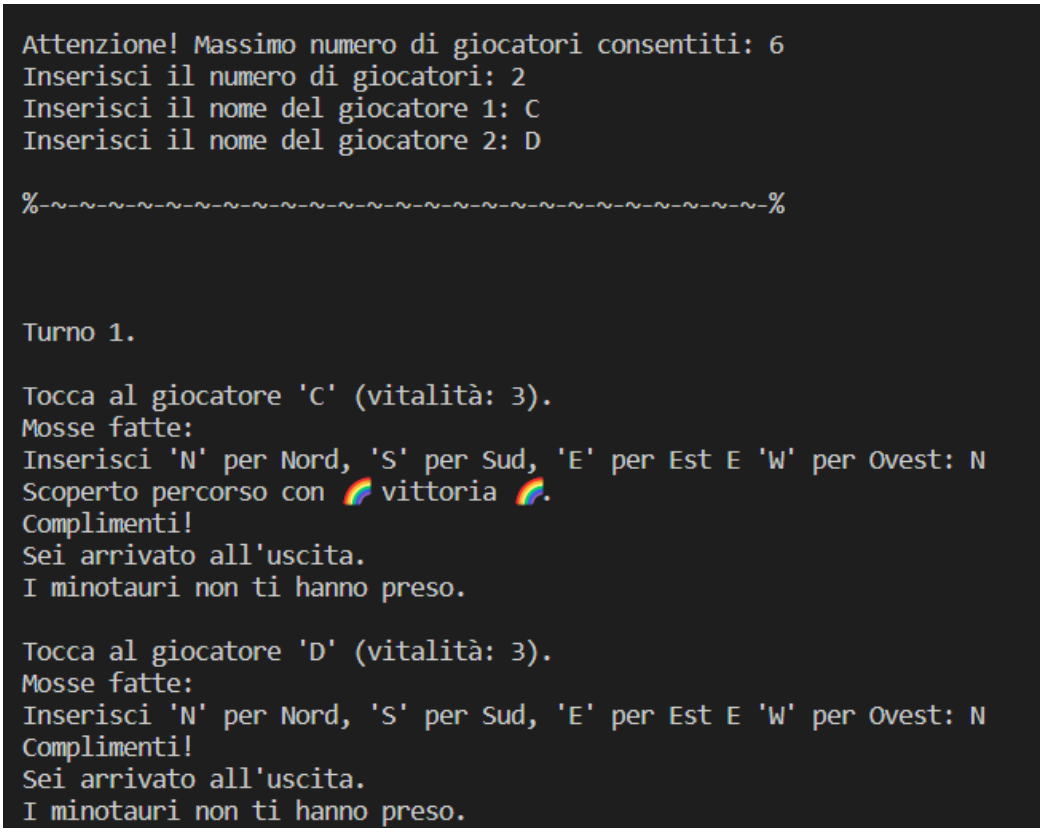

Figura 5: Vittoria di entrambi i giocatori.

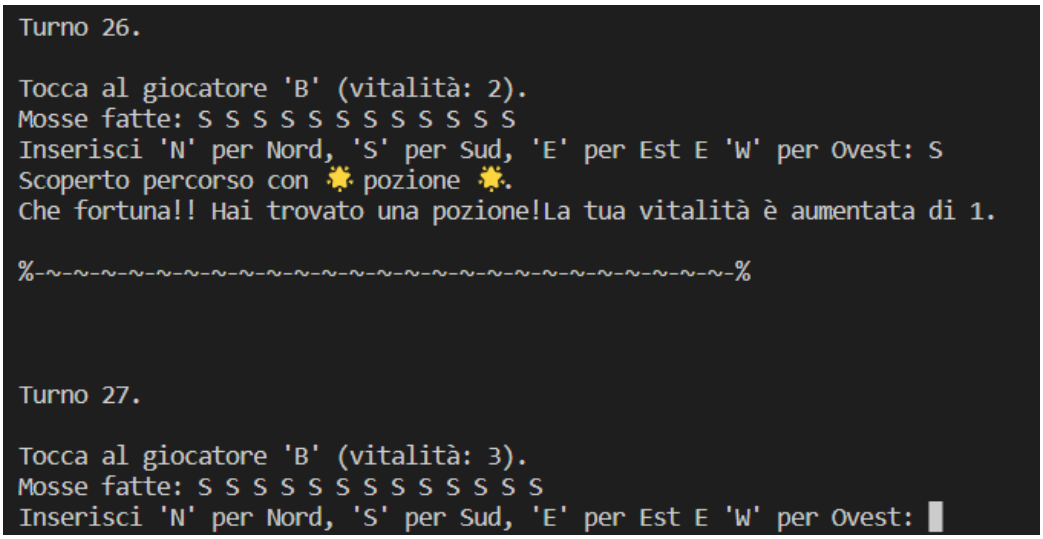

Figura 6: Path con pozione ed effetti.

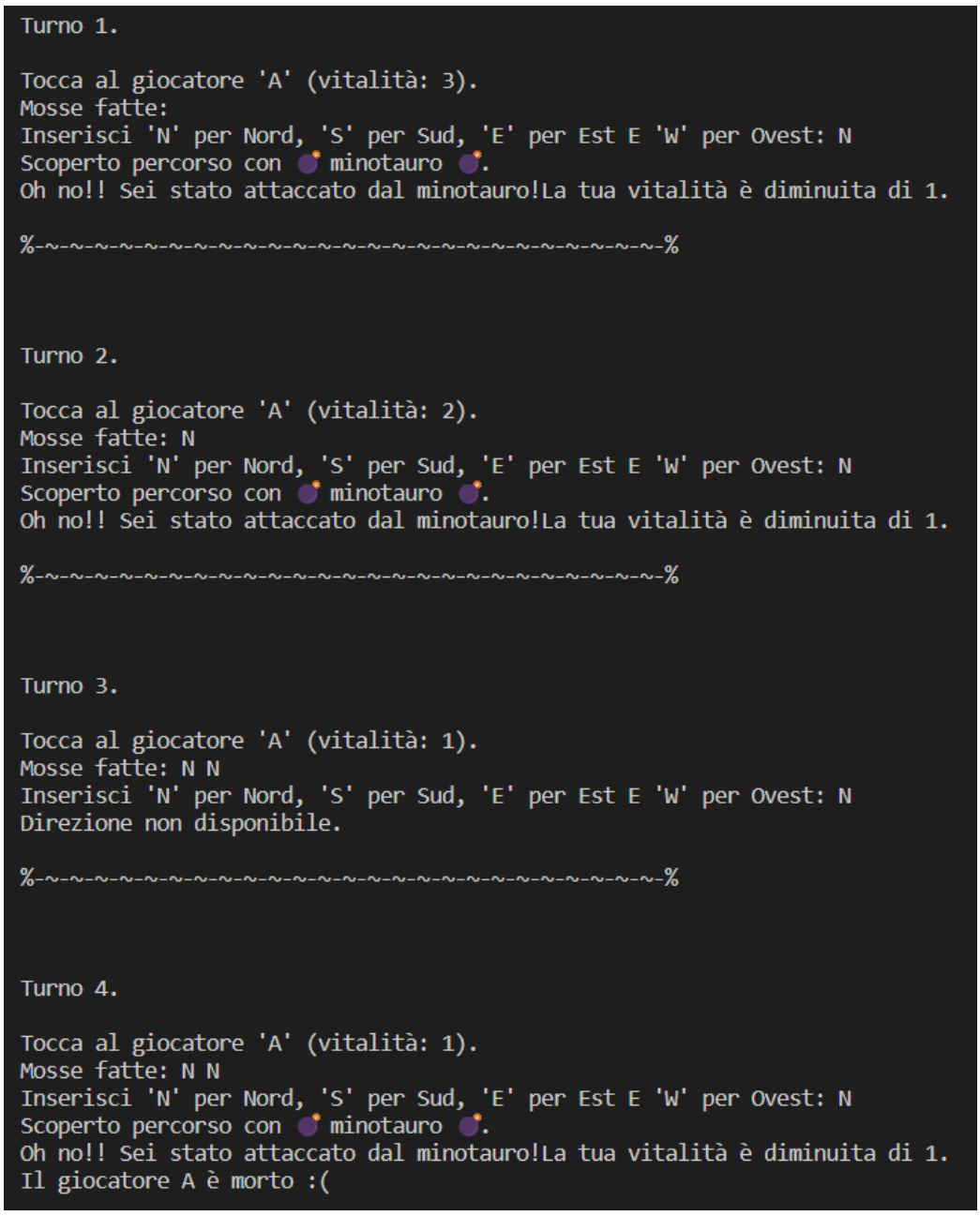

Figura 7: Morte del giocatore.

Figura 8: Prova gioco 1.

Figura 9: Prova gioco 1.

Figura 10: Prova gioco 2.

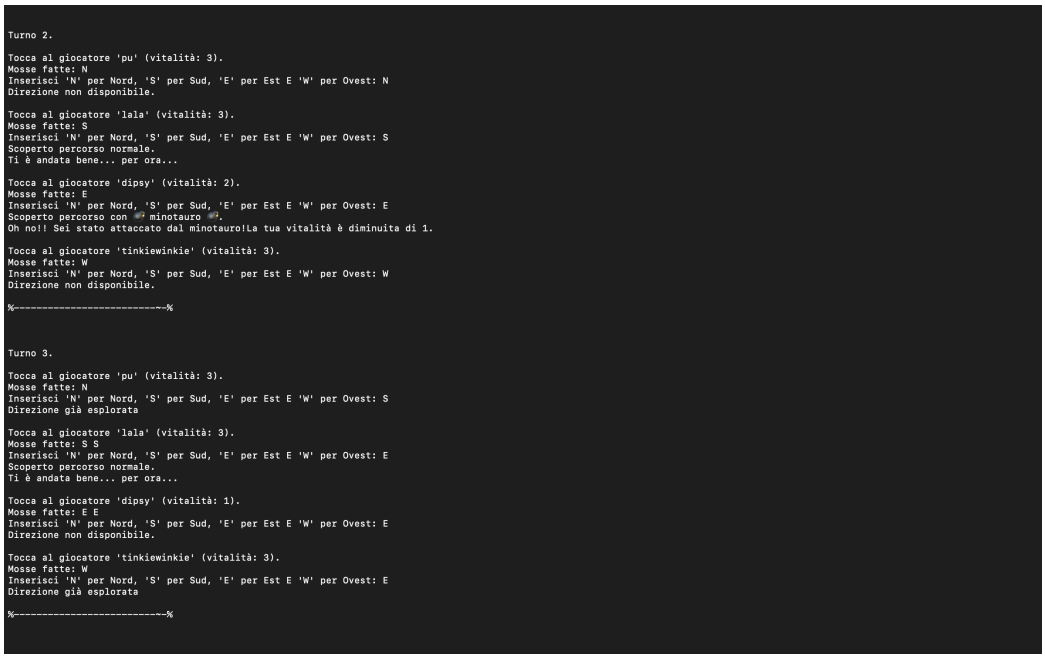

Figura 11: Prova gioco 2.

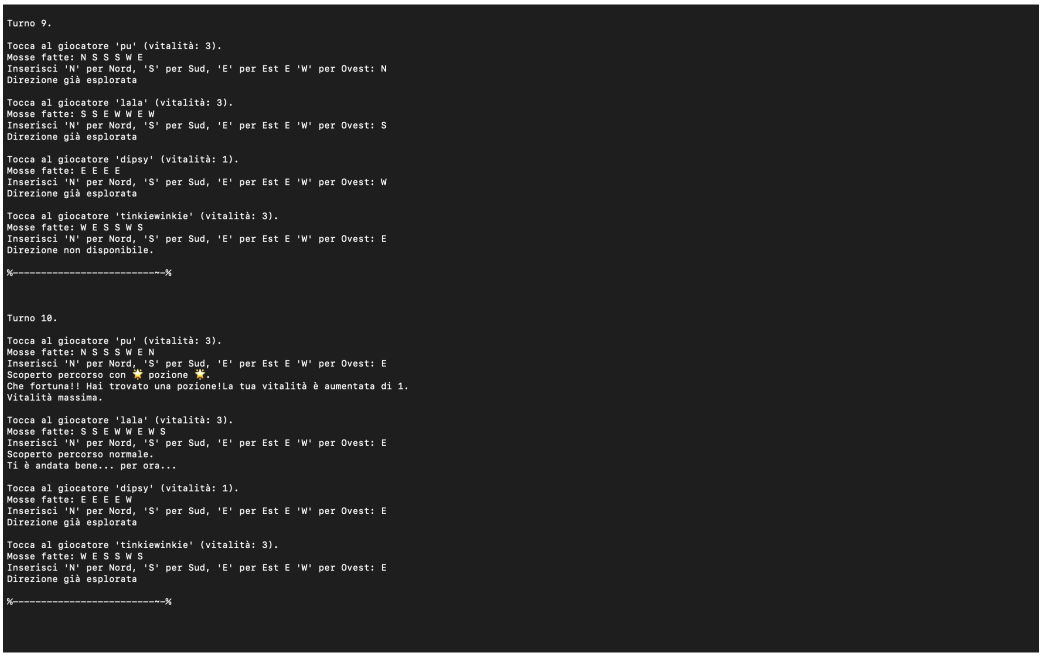

Figura 12: Prova gioco 2.

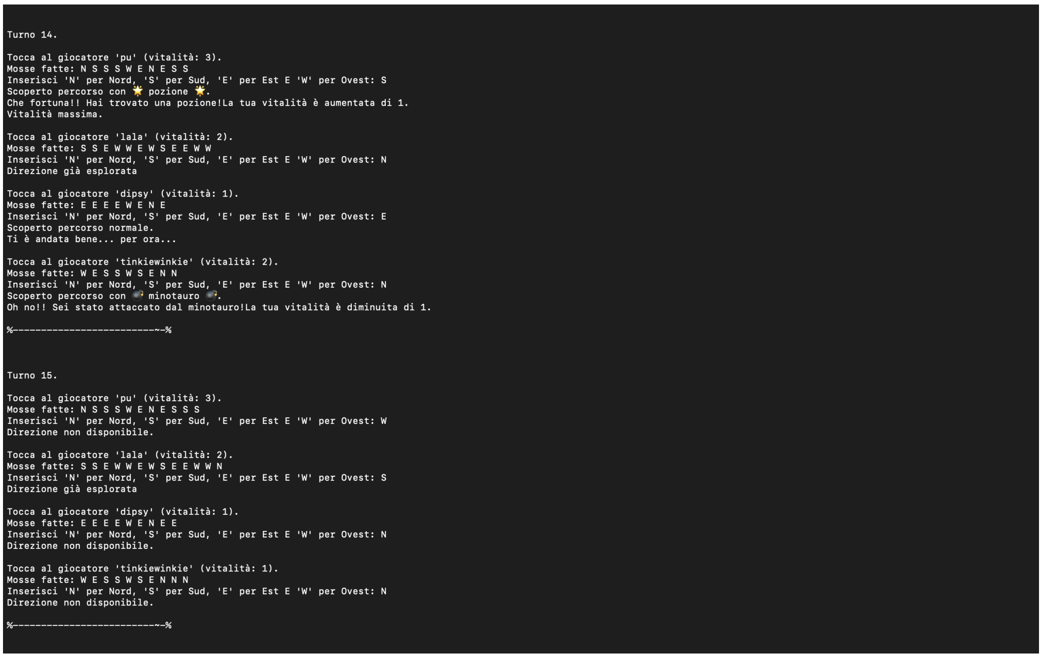

Figura 13: Prova gioco 2.

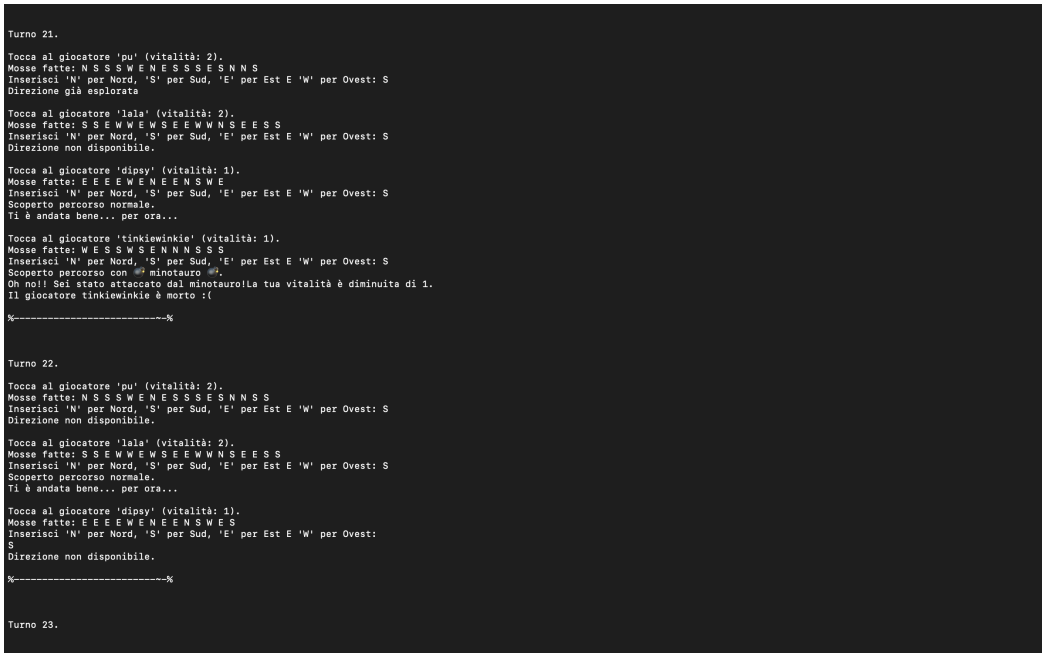

Figura 14: Prova gioco 2.

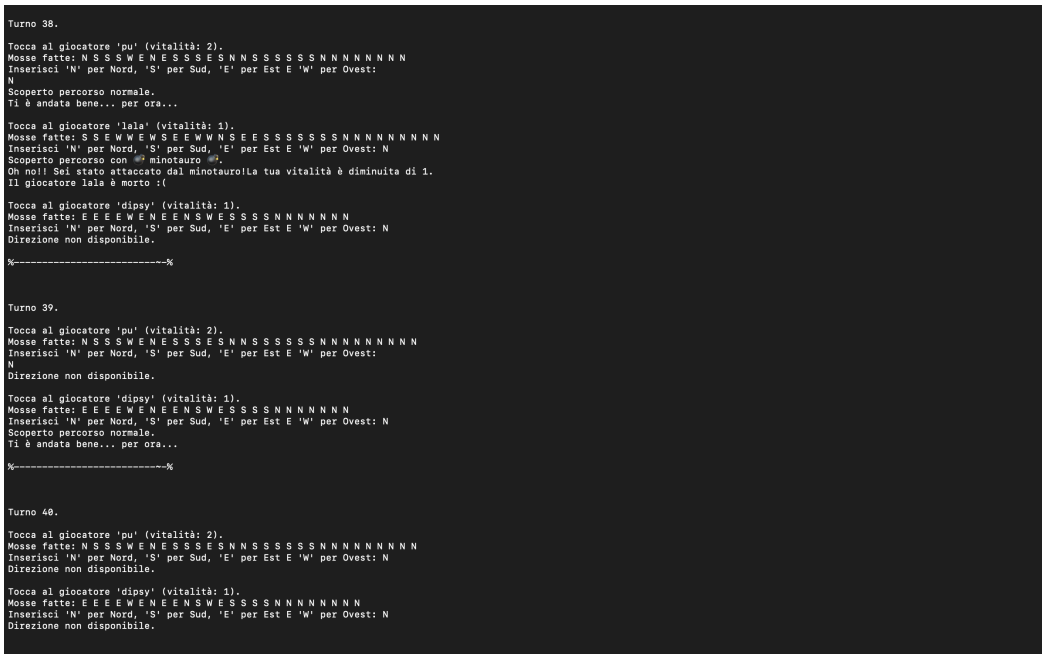

Figura 15: Prova gioco 2.

```
%-----------------------~-%



Turno 47.

Tocca al giocatore 'pu' (vitalità: 2).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: E
Scoperto percorso con 💣 minotauro 💣.
Oh no!! Sei stato attaccato dal minotauro!La tua vitalità è diminuita di 1.

Tocca al giocatore 'dipsy' (vitalità: 2).
Mosse fatte: E E E E W E N E E N S W E S S S S N N N N N N N N N N E E
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: E
Direzione non disponibile.

%-----------------------~-%



Turno 48.

Tocca al giocatore 'pu' (vitalità: 1).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N E
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: E
Direzione non disponibile.

Tocca al giocatore 'dipsy' (vitalità: 2).
Mosse fatte: E E E E W E N E E N S W E S S S S N N N N N N N N N N E E
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: E
Scoperto percorso con 💣 minotauro 💣.
Oh no!! Sei stato attaccato dal minotauro!La tua vitalità è diminuita di 1.

%-----------------------~-%



Turno 49.

Tocca al giocatore 'pu' (vitalità: 1).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N E
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: E
Scoperto percorso normale.
Ti è andata bene... per ora...

Tocca al giocatore 'dipsy' (vitalità: 1).
Mosse fatte: E E E E W E N E E N S W E S S S S N N N N N N N N N N E E E
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: E
Direzione non disponibile.

%-----------------------~-%
```

Figura 16: Prova gioco 2.

```
%-----------------------~-%



Turno 57.

Tocca al giocatore 'pu' (vitalità: 1).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N E E E E W
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: W
Direzione già esplorata

%-----------------------~-%



Turno 58.

Tocca al giocatore 'pu' (vitalità: 1).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N E E E E W W
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: W
Direzione già esplorata

%-----------------------~-%



Turno 59.

Tocca al giocatore 'pu' (vitalità: 1).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N E E E E W W W
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: W
Direzione già esplorata

%-----------------------~-%



Turno 60.

Tocca al giocatore 'pu' (vitalità: 1).
Mosse fatte: N S S S W E N E S S S E S N N S S S S S S N N N N N N N N N N E E E E W W W W
Inserisci 'N' per Nord, 'S' per Sud, 'E' per Est E 'W' per Ovest: W
Scoperto percorso con 🌈 vittoria 🌈.
Complimenti!
Sei arrivato all'uscita.
I minotauri non ti hanno preso.

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...none found.

[Processo completato]
```

Figura 17: Prova gioco 2.

nel path con la vittoria, questo non viene riconosciuto. Per ovviare questo "problema" abbiamo inserito la famosa citazione "alle scale piace cambiare".

- Nel momento in cui un giocatore raggiunge la vittoria, tutti gli altri devono obbligatoriamente arrivare al path di tipo `victory` dalla stessa direzione (per il punto sopra). Per migliorare la giocabilità andrebbe inserito anche un `cout` che informava il percorso che il giocatore che ha scoperto la vittoria ha seguito in modo da consentire agli altri giocatori di ripercorrerlo senza necessità di scorrere ogni volta il terminale alla ricerca di tali mosse.

5. Riferimenti

- Appunti presi a lezione e dispese del professore;
- Per rand(), <cstdlib> | Microsoft Docs (ultimo accesso il 24/02/2022);
- per le emoji, `https://emojiterra.com/it/` (ultimo accesso il 24/02/2022).