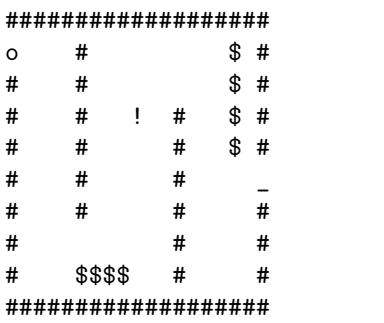
Snake Labyrinth



La figura sopra rappresenta un labirinto dove il nostro protagonista Snake `o` deve trovare l'uscita `_`. Snake percorre il labirinto raccogliendo o evitando gli oggetti nel piano di gioco finché non raggiunge l'uscita. Ogni oggetto può aumentare o diminuire il punteggio del nostro eroe. Anche la lunghezza del percorso è importante perché Snake vuole uscire il prima possibile.

L'obiettivo del gioco è di trovare il miglior percorso che porta all'uscita del labirinto.

Per *migliore* si intende il percorso che **produce il punteggio più elevato**. Ciò significa che il percorso migliore non è semplicemente il più breve.

Mosse possibili

Inizialmente Snake ha 1000 punti.

Ad ogni turno le mosse possibili sono:

- N: un passo verso l'alto (Nord)
- S: un passo verso il basso (Sud)
- O: un passo a sinistra (Ovest)
- E: un passo a destra (Est)

Ogni passo costa 1 punto.

Premi e Ostacoli

Durante il percorso Snake può incontrare:

- `#` pareti che non possono essere attraversate;
- `$` monete, ciascuna del valore di 10 punti e che inoltre incrementano la sua lunghezza (vedi sotto la sezione Monete);
- `!` imprevisti, che se incontrati dimezzano (divisione intera) il numero di monete raccolte, e la sua lunghezza;
- `T` trapano che permette di perforare le pareti. Al trapano è associato un numero di usi disponibili inizialmente pari a 0. Ogni vola che viene raccolta una `T`, il numero di usi disponibili del trapano aumenta di 3. Ogni volta che Snake entra in una casella con una parete il numero di usi diminuisce di 1, e la casella diventa vuota, cioè la parete nella casella scompare. Se il numero di usi è pari a 0, snake non può entrare in una cella con una parete. Le pareti esterne possono essere trapanate, ma non si possono superare i bordi del gioco.

Gli oggetti scompaiono dopo la prima interazione con Snake. Una volta raccolta una moneta, questa viene rimossa dal piano di gioco e non può essere raccolta due volte. In modo simile per gli imprevisti. Snake non può occupare una cella dove si trova una parete `#`, e quindi le pareti non scompaiono mai, a meno che non venga usato il trapano.

Una volta trovata l'uscita, il punteggio è dato dal **punteggio iniziale più la somma dei valori delle monete possedute all'uscita meno il numero di passi eseguiti**.

Monete

L'unico modo che Snake ha di raccogliere le monete è di "mangiarle", con il risultato di allungarsi di una cella. Quando la testa di Snake raggiunge l'uscita _ il gioco termina (non c'è bisogno che tutto lo Snake esca dal labirinto).

Sotto l'esempio di una partita in un minuscolo labirinto.

```
#####    (inizio)
o $ _
#####

#####    (1.o passo)
 o$ _
#####

#####    (2.o passo: mangio la moneta e mi allungo di uno °)
 °o _
#####

#####    (3.o passo)
  °o_
#####

#####    (4.o passo: fine)
   °o
#####
```

Il punteggio raggiunto è di 1000 + 10 - 4 = 1006 (1000 punti iniziali + 1 moneta - 4 passi).

Se la testa incontra una parte del corpo, allora viene persa la parte compresa tra la coda e il punto di incontro, e di conseguenza vengono perse le relative monete.

```
#########  #########  #########  #########
#°      #  #       #  #       #  #       #
#°°°°°°° #  #°°°°°° #  #  o°°° #  #  °o°° #
#     ° #  #  o  ° #  #  °  ° #  #  °  ° #
#  o°°° #  #  °°°° #  #  °°°° #  #  °°°° #
#       #  #       #  #       #  #       #
#########  #########  #########  #########
```

Nella seconda figura, Snake si è mosso a Nord. Nella terza si è mosso nuovamente a Nord perdendo le monete precedenti al punto di incontro (incluso). Notate che tutto il corpo dello snake si è spostato di 1 prima di determinare il punto di incontro. Nella quarta figura, Snake si è mosso a E senza incrociare il resto del corpo.

Nell'esempio sotto Snake ha una lunghezza tale che non genera punti di incontro con il resto del corpo.

```
#########  #########  #########  #########
#       #  #       #  #       #  #       #
# °°°°° #  #  °°°° #  #  o°°° #  #  °o°° #
#     ° #  #  o  ° #  #  °  ° #  #  °  ° #
#  o°°° #  #  °°°° #  #  °°°° #  #  °°°° #
#       #  #       #  #       #  #       #
```

```
#########  #########  #########  #########
```

Sotto un altro esempio nel caso particolare di un vicolo cieco.

```
#########  #########  #########  #########
# °°°°o #  #  °°°°o#  #     o°#  #    o° #
#  ######  #  ######  #  ######  #  ######
#       _  #       _  #       _  #       _
#########  #########  #########  #########
```

Nella seconda figura, Snake si è mosso a Est avvicinandosi alla parete.

Nella terza figura Snake si muove a Ovest:

- il corpo ha continuato a seguire la testa spostandosi di conseguenza
- la testa si sposta a Ovest incontrando il corpo e tenendo solo la parte dalla testa al punto di incontro (escluso)

Nella quarta mossa Snake si muove nuovamente a Ovest.

Riassumendo, prima si muove tutto il corpo e dopo si valutano eventuali punti di incontro.

Modalità interattiva

Viene chiesto all'utente quale sia la mossa successiva. Il piano di gioco ed il punteggio vengono aggiornati di conseguenza.

Modalità AI

Si richiede di implementare una funzione che calcoli una buona strategia di gioco.

Strategia 1: Random

Nella strategia Random, Snake si muove in maniera casuale ad ogni passo finché non viene trovata la via di uscita

Strategia 2: Sempre a Destra

Una strategia che consente di uscire sempre da qualunque labirinto é quella di appoggiare una mano su una parete e proseguire seguendo la parete. Sotto il percorso nel primo labirinto che si ottiene mantenendo la parete alla propria destra.

```
####################
..   #          $ #
#.   #     ...  $ #
#.   #   ! .#.  $ #
#.   #     .#.  $ #
#.   #     .#.   ..
#.   #     .#.   .#
#.         .#.   .#
#...........#.....#
####################
```

Altre Strategie

A voi piena libertà. Una opzione è una soluzione ricorsiva che implementi la ricerca del miglior percorso possibile.

Labirinti

Potete sperimentare il gioco con uno o più labirinti pensati da voi. Alcuni labirinti verranno pubblicati durante il corso.

Schema di implementazione

Il programma che implementerete dovrà chiedere al giocatore quale mossa giocare, e dovrà visualizzare il campo di gioco aggiornato (usando la funzione `printf` o equivalente). Dovrà verificare che la mossa scelta sia legale (non attraversi dei muri), e dovrà aggiornare il punteggio. Questi passi si ripetono fino alla fine della partita.

Nella modalità AI, il programma dovrà calcolare il percorso migliore, visualizzare a schermo il percorso trovato. Infine dovrà visualizzare la stringa con tutte le mosse eseguite (es. "NNSSONSEEE…").

Siete liberi di sperimentare e inventare schemi di gioco o varianti a vostro piacimento, pur rispettando i requisiti riportati più in basso in questo documento.

Consegna

Quando? Il progetto deve essere consegnato 4 giorni prima la data di ciascun appello. E' sempre obbligatorio iscriversi all'appello di "Esercizi" prof. Spanò per sostenere la discussione del progetto.

Dove? Il docente abiliterà la consegna tramite Moodle secondo le scadenze previste.

Cosa? Dovrete consegnare un unico file zip contenente:

1. Una relazione scritta di al massimo 3 pagine che descriva la struttura del vostro progetto, l'organizzazione del lavoro tra i componenti del gruppo, le principali difficoltà incontrate. Relazioni più lunghe verranno penalizzate.
2. Il codice sorgente del progetto in linguaggio C99; eventuali parti di codice scritto in altri dialetti vanno isolati in sorgenti separati ed il progetto dovrà compilare opportunamente.
3. Documentazione delle funzioni, dei tipi e dei file generata con Doxygen

Librerie esterne

E' possibile implementare l'intero gioco utilizzando semplicemente la standard library di C, usando le funzioni di lettura dei tasti come `scanf`, `getc` o `getchar` e le funzioni di stampa come `printf`. Coloro che vogliono cimentarsi nell'uso di librerie esterne per dare una veste grafica ed una interazione col giocatore più accattivanti e fluide possono farlo, tuttavia sarà propria responsabilità occuparsi degli aspetti di portabilità e compatibilità. Il programma deve compilare e funzionare correttamente sulle 3 piattaforme (Windows, Linux, Mac); in caso contrario, è necessario documentare e motivare opportunamente la scelta nella relazione che accompagnerà il progetto ed in sede d'esame.

Requisiti

Per i progetti di *gruppo*:

- progetto **sufficiente** se permette ad un utente di giocare inserendo le proprie mosse tramite input da tastiera (modalità interattiva); l'implementazione delle regole del gioco deve essere corretta.
- progetto **buono** se implementa la modalità AI utilizzando un algoritmo semplice; avete piena libertà di definire una strategia.

- progetto **ottimo** se implementa una strategia particolarmente interessante, come ad esempio un **algoritmo ricorsivo**, per il la modalità AI;

Tutti i membri del gruppo devono conoscere ogni riga del codice!

Nel caso di progetti *individuali*, ad esempio per studenti lavoratori:

- progetto **sufficiente** se permette ad un utenti di giocare inserendo le proprie mosse tramite input da tastiera; l'implementazione delle regole del gioco può essere ragionevolmente parziale.
- progetto **buono** se permette ad un utente di giocare inserendo le proprie mosse tramite input da tastiera; l'implementazione delle regole deve essere corretta.
- progetto **ottimo** se implementa una strategia in modalità AI; avete piena libertà di definire una strategia.

Importante! Il progetto è di gruppo, ma **la valutazione è individuale**. Questo significa che i componenti di un gruppo potrebbero ricevere un voto diverso.

Inoltre, è possibile per alcuni dei componenti presentare individualmente delle **migliorie** al progetto. Possibili migliorie potrebbero essere: grafica migliore, strategia di gioco più raffinata, menu di interazione dell'utente più usabile, ecc.

Non ci sono limiti alle aggiunte o modifiche che vorrete fare! Quindi non ponetevi limiti!

Challenges

Durante il corso vi proporremo alcuni labirinti su cui sperimentare la vostra strategia AI.

Fisseremo 3 scadenze intermedie:

- 18 Novembre
- 2 Dicembre
- 16 Dicembre

Il vostro codice verrà testato in modo da calcolare il punteggio della vostra soluzione.

Sarà un ottimo modo per avere dei commenti sul progresso del vostro lavoro e il vostro risultato verrà preso in considerazione nella discussione del progetto.

Specifiche per le challenges

Per le challenges il codice verrà testato in maniera *automatica*, pertanto la vostra implementazione dovrà essere molto precisa e rispettare il protocollo seguente.

Consegna. Dovrete consegnare in Moodle un unico file `nome-gruppo.zip`, contenente un'unica cartella denominata `iap` con al suo interno tutti i file sorgente. Basta che un solo membro del gruppo effettui la consegna in Moodle. Nella cartelal `iap` deve essere presente un file `main.c` e la prima riga di questo file deve indicare i componenti del gruppo (anche nel caso di consegna individuale) come segue:

```
// matricola1 matricola2 matricola3
```

Compilazione. Il vostro codice verrà compilato con il comando `gcc -O2 -std=c99 --pedantic *.c -o iap`. Questo significa che potrete usare anche più di un unico file `.c`, e che non è concesso usare altre estensioni diverse da `.c`. Il comando sopra genera i file eseguibile `iap`.

Esecuzione. Il vostro eseguibile verrà invocato con un unico parametro addizionale a linea di comando come segue: `./iap --challenge`. Il suggerimento è di implementare un unico progetto, dove il parametro

`--challenge` abilita la consegna per le challenge, mentre l'esecuzione interattiva/AI avviene in assenza del parametro.

INPUT / Piano di Gioco. Nella modalità challenge, il vostro codice dovrà eseguire diverse `scanf` per leggere il piano di gioco da standard input. La prima `scanf` legge un intero `M` corrispondente al numero di colonne del piano di gioco, mentre la seconda `scanf` legge un intero `N` corrispondente al numero di righe del piano di gioco, infine dovrà eseguire `N` operazioni di `scanf` per leggere le righe del piano di gioco codificate come stringhe. Un esempio di input è dato di seguito. *Suggerimento:* per testare velocemente il vostro codice vi consigliamo di "copia-incollare" le 12 righe di seguito nel terminale.

```
19
10
###################
o    #          $ #
#    #          $ #
#    #   !  #   $ #
#    #      #   $ #
#    #      #     _
#    #      #     #
#           #     #
#    $$$$   #     #
###################
```

OUTPUT. L'unico output del vostro programma deve essere una singola linea con le mosse del vostro miglior percorso (`N`:Nord, `S`:Sud, …), come nell'esempio sotto. Usate solo lettere maiuscole e non aggiungere altro testo (es. `"La soluzione è:"`).

```
ESSSSSSSEEEEEEEEEEENNNNNNEEEEESSSSEE
```

Specifiche per la prima challenge del 18 Novembre

Per la prima scadenza utilizzeremo solo piani di gioco molto semplici, ovvero:

- non ci sono pareti *interne*, ma solo le pareti *perimetrali* che delimitano il piano di gioco stesso
- i punti di inizio e di uscita possono trovarsi in una posizione qualunque del perimetro
- all'interno del piano di gioco gli unici elementi che si possono incontrare sono le monete

Labirinti più complessi saranno affrontati nelle challenge successive.

Di seguito due possibili input per il vostro programma

```
10
3
##########
o   $$$  _
##########



10
5
########o#
#        #
# $$$$$$ #
#        #
#_########
```

Specifiche per la seconda challenge del 2 Dicembre

In aggiunta alla prima challenge, dovrete gestire:

- `!` imprevisti, che se incontrati dimezzano (divisione intera) il numero di monete raccolte;
- `T` trapano che permette di perforare le pareti. Vedi descrizioni all'inizio di questo file;
- pareti interne accerchiabili. Vedi descrizione sotto.

Vi ricordo che è importante usare esattamente lo stesso nome del gruppo se volete che venga riconosciuto.

Pareti interne accerchiabili

Per introdurre il problema delle pareti interne, cominceremo con pareti che si possono sempre accerchiare. Questo significa che è sempre possibile fare un giro completo attorno a questo tipo parete. Le pareti di questo tipo non introducono vicoli ciechi, ed è quindi più facile trovare una soluzione al labirinto. Sotto alcuni esempi di pareti accerchiabili.

Una o più pareti costituite da una sola cella:

```
##########    ##########
#        #    #        #
o        _    o #   #  _
#   #    #    #   #    #
#        #    # #   #  #
#        #    #        #
##########    ##########
```

Una o più pareti costituite da una "linea" orizzontale o verticale di celle:

```
##########    ##########    ##########    ##########
#        #    #        #    #        #    #        #
o        _    o  #     _    o  ####  _    o  #  #  _
#  ####  #    #  #     #    #        #    #  #  #  #
#        #    #  #     #    #  ####  #    #  #  #  #
#        #    #        #    #        #    #        #
##########    ##########    ##########    ##########
```

Una o più pareti fatte da un "rettangolo" pieno o vuoto di celle:

```
##########    ##########
#        #    #        #
o  ####  _    o ###### _
#  #  #  #    # ###### #
#  ####  #    # ###### #
#        #    #        #
##########    ##########
```

Specifiche per la terza challenge del 16 Dicembre

In aggiunta alle precedenti challenges, dovrete gestire:

- allungamento di Snake quando viene incontrata una moneta
- accorciamento di Snake quando viene incontrato un imprevisto
- accorciamento di Snake quando incontra parte del suo stesso corpo

Il comportamente dello Snake è descritto in maggior dettaglio nella precedente sezione Monete.

Non saranno usati labirinti con vicoli ciechi nella terza challenge.

Specifiche per la consegna finale in corrispondenza dell'appello

La consegna per l'appello prevede l'implementazioni di tutto quanto descritto in questo documento. L'unica aggiunta rispetto alla terza challenge consiste nella presenza di vicoli ciechi nel labirinto, come nell'esempio sotto.

```
##########
#      # #
o ###### _
#    # # #
#  # # # #
#  #     #
##########
```

Il progetto finale deve prevedere la modalità AI e la modalità interattiva. Non sono previsti parametri a linea di comando come `--challenge` dovrete però gestire la lettura di un labirinto come nelle challenges sopra.