Come creare un Textgame.

Guida alla programmazione di the Tower.

Paolo Poli

2022

Indice

0.1 Cosa è un textgame.

ll soggetto introdotto in questo testo è il textgame. Con questa parola intendo giochi per il PC, senza interazione grafica, ma con solo l'utilizzo dei caratteri ASCII. Questa tipologia di giochi da spazio alle idee e alla fantasia del programmatore, soggetto allo stesso tempo a poche limitazioni derivanti da una mancata conoscenza di certi argomenti, il quale vanno oltre alla semplice programmazione. Quello che presento è un esempio di sviluppo di textgame, tramite la programmazione in python. Insieme alla spiegazione del gioco, sarà commentato anche il codice scritto, in modo da capire come è stato implementato il tutto.

0.2 Il gioco.

Il gioco che prendiamo in considerazione è una torre, formata da tante stanze ed ogni stanza può considerarsi come un puzzle. Ogni stanza presenterà una serie di enigmi e domande, tramite il quale se risposte in modo corretto permetteranno al giocatore di capire come sistemare le stanze, per poi aprire il cancello che nell'altra stanza lo manderà. Il gioco si presenta come la figura sottostante.

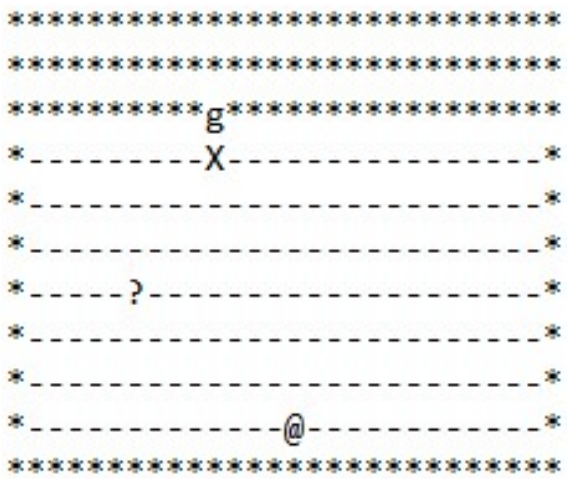

Figura 1: *Schema iniziale del gioco.*

La chiocciola è il vostro personaggio che si può muovere solo sul carattere -. Le X sono delle porte, gli asterischi muri, i punti di domanda sono iscrizioni riguardanti indovinelli e enigmi e g è il cancello che vi porterà nella stanza successiva una volta risolto l'enigma della stanza. Ia giocabilità è molto semplice (cosa che ritengo importante, quando si crea un gioco di questo tipo). Con le freccette muovete il vostro personaggio, con c compiete una azione, come prendere un oggetto, interrogare un iscrizione o aprire il cancello finale.

Con d, lasciate gli oggetti presi (potete averne al massimo 10). Quando premerai il tasto c sul simbolo ?, un messaggio comparirà, come forma di indovinello o indizio (questo lo dovrai capire te). Una volta che pensi che la stanza sia sistemata, dovrai andare sopra il cancello g (gate) e premere c, se la soluzione sarà esatta passerai.

0.3 Il design del gioco.

Il gioco è stato strutturato in modo che con poche righe di codice, sia possibile la creazione di un numero schemi a piacimento. Una volta sviluppato il codice (che dopo commenterò), la creazione degli schemi e delle soluzioni vengono implementate direttamente su files di testo. Questo tipo di approccio può sembrare elementare, ma ha permesso in poco tempo lo sviluppo del gioco, dando la possibilità di introdurre in modo facile un nuovo schema. L'unico incoveniente che ha, è che se l'utente vuole può guardare la soluzione facilmente, ma questo non è una novità per chi conosce l'enigmistica! l files sono messi in directory con nomi specifici, che nel nostro caso è sch susseguiti da un numero, che rappresenta il numero della stanza. Dentro tali directory troviamo i files, schema.txt, endschema.txt e questions.txt. Nel primo file trovate proprio la stanza disegnata con i caratteri (la stanza la disegnate proprio con i caratteri) come è stata mostrata nella figura sopra. Nel file endschema.txt, c'è la soluzione per potere passare alla stanza successiva. Il file questions.txt invece raccoglie gli indovinelli o indizi posti durante il gioco. All'interno di questions.txt troverete tutto il necessario riguardante la gestione di un indovinello o indizio trovato durante il gioco, in un codice scritto come quello sottostante:
$19, 3 :' Donaetisardonato!' ::^{:} 22, 2$
$13, 7 :' 1, 1, 2, 3, 5, 8, ?' : 13 : - : 12, 7$

Ogni riga si riferisce, a una domanda o indizio. Il carattere : serve a dividere in sezioni le informazioni riguardante il quesito. Il primo elemento, è una coppia di numeri che rappresenta in forma cartesiana la coordinata sulla stanza di dove è situato il punto di domanda. Il secondo elemento è la frase associata (indovinello o indizio) al punto interrogativo messo, il terzo elemento è la risposta al quesito (in caso di nulla scritto, significa che esso è un indizio). Il quarto elemento è l'evento associato, ovvero il simbolo nuovo che inserirà a risposta corretta. Nel caso il il quesito fosse solo un indizio, verrà riscritto un simbolo uguale in una posizione scelta. L'insieme di questi 3 files, permette così lo sviluppo di diversi schemi, dando totale spazio all'immaginazione e fantasia del textgame designer.

0.4 Il codice

0.4.1 Le librerie

Veniamo ora al cuore del videogioco. Il codice. In Python inizialmente devono essere chiamate le librerie fondamentali che contengono i comandi aggiuntivi necessari per il proprio programma.

```
import os
import keyboard
import time
```

la libreria os, permette l'interfacciamento del linguaggio con i comandi del sistema operativo. Questo a noi ci serve per pulire lo schermo con il comando cls di Ms-dos. La seconda libreria, serve per riconoscere gli eventi istantanei sulla tastiera, quindi il tasto premuto. La libreria time serve nel caso si vuole contare lo scorrimento del tempo, o gestire alcune pause temporanee.

0.4.2 Il personaggio

Chiamate le librerie utili, possiamo iniziare a impostare il codice. Come prima cosa, creiamo il nostro personaggio, dichiarando una classe con le proprie caratteristiche.

```
class Player:
    def __init__(self):
        self.x=0
        self.y=0
        self.objects=[]
```

Le variabili dichiarate sono la posizione del nostro personaggio (x,y), e una variabile lista di oggetti, ovvero l'inventario.

0.4.3 La presentazione

Come ogni gioco che si rispetta è necessaria una piccola presentazione, chiamata tramite la seguente funzione che richiama, il testo nel file initialpage.txt.

```
def presentation():
    fileschema='initialpage.txt'
    with open(fileschema) as f:
```

```
        page=f.read()
    print(page)
```

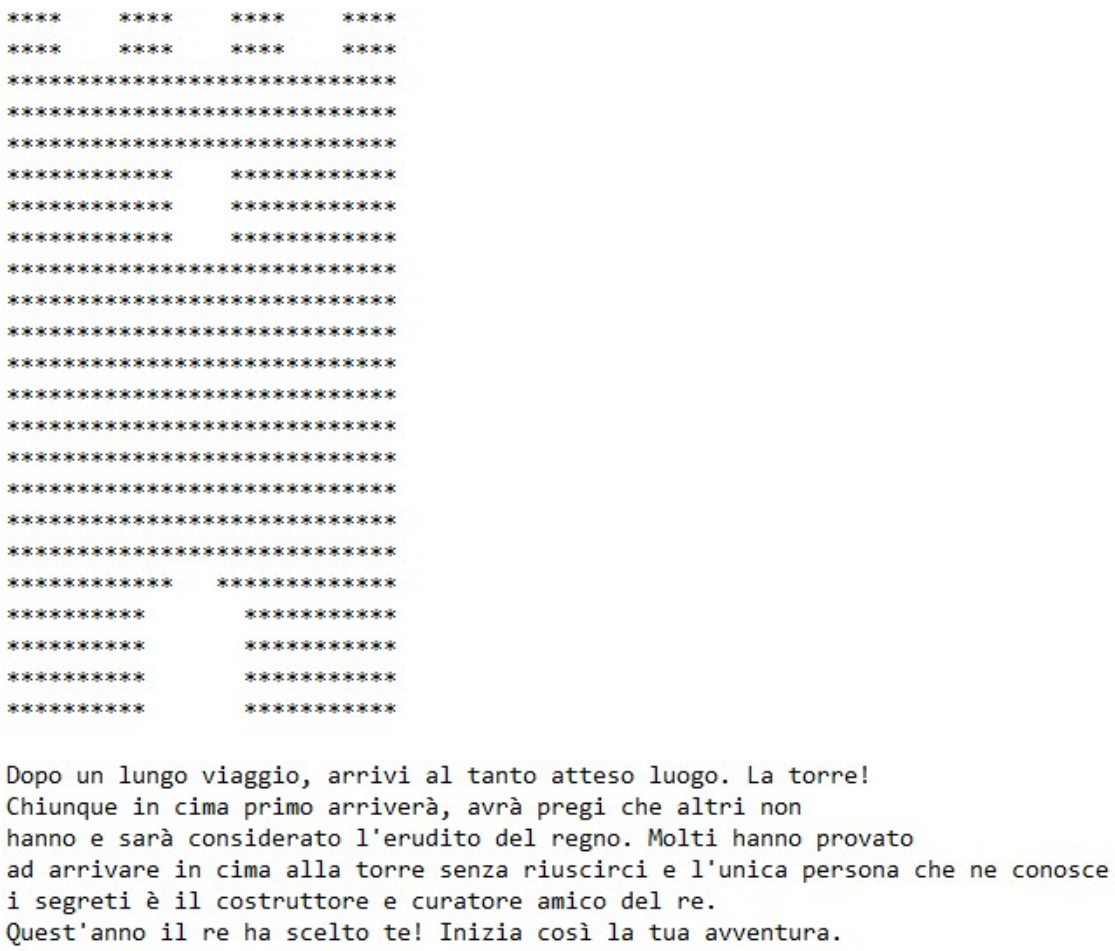

Figura 2: *Presentazione del gioco.*

0.4.4 Lo schema

Dichiarata la classe personaggio, dobbiamo fare in modo che il nostro codice legga lo schema iniziale del gioco, implementato nel file di testo. Questa procedura è stata effettuata tramite una funzione chiamata read_ schema, rappresentata sottostante.

```
def read_schema(nfile):

    fileschema='sch'+str(nfile)+'/schema.txt'
    with open(fileschema) as f:
        maps=f.read()
```

```
contalinee=0
contacolonne=0
for elemento in maps:
    if elemento!='\n':
        schema[contalinee][contacolonne]=elemento
        if elemento=='@':
            hero.x=contacolonne
            hero.y=contalinee

    else:
        contacolonne=-1
        contalinee+=1

    contacolonne+=1
```

In questa funzione viene letto il file dello schema tramite la variabile fileschema, che acquisice la directory di lettura tramite la variabile passata nfile, che etichetta il numero di directory dove leggere lo schema di riferimento. Con il comando with si legge il file testuale e lo si copia sulla variabile maps. Successivamente tutti i caratteri contenuti in maps, vengono memorizzati nella matrice schema, che è la variabile sul quale il gioco riconoscerà lo schema. La chiocciola in schema definisce proprio le coordinate del nostro eroe.

0.4.5 La soluzione

Il passo successivo è memorizzare lo schema soluzione in modo che il nostro gioco riconosca quando il puzzle è stato risolto. La seguente parte è stata messa in una funzione apposita chiamata read_solution.

```
def read_solution(nfile):

    fileschema='sch'+str(nfile)+'/endschema.txt'
    with open(fileschema) as f:
        maps=f.read()
    contalinee=0
    contacolonne=0
    for elemento in maps:
        if elemento!='\n':
            schemafinale[contalinee][contacolonne]=elemento
```

```
        else:
            contacolonne=-1
            contalinee+=1

        contacolonne+=1
```

Questa parte di codice è simile a quella precedente, con la differenza che il file letto è quello riferito alla mappa finale del gioco (la soluzione). Lo schema finale ovvero la soluzione stanza in questo caso è memorizzato nella matrice shemafinale.

La funzione che ha il compito di confrontare lo schema del gioco con la soluzione si chiama check_solution()

```
def check_solution(n):
    donot=False
    for i in range(n):
        for j in range(n):
            if schema[i][j]!=schemafinale[i][j]:
                donot=True
    #print (schemafinale)
    return donot
```

Essa non fa altro che confrontare le due matrici schema e schemafinale, in caso queste fossero identiche, la funzione conferma l'evento, segnalando che lo schema è stato risolto, tramite la variabile donot. Se donot rimane falso lo schema è risolto!

0.4.6 Lo schema a schermo

Lo schema per potere essere visualizzato dal giocatore, deve essere scritto sullo schermo fisicamente e per questo viene chiamata la funzione screen_schema().

```
def screen_schema(n):
    for i in range(n):
        if schema[i][0]=='"':
            break
        for j in range(n):

            if (schema[i][j]!=0 and schema!='"'):
```

```
            print(schema[i][j],end="")


        elif schema[i][j]!='"':
                print ("\n")
                break
```

Esso legge la matrice schema fino a quando trova le doppie virgolette. All'interno di tale ciclo il for disegna a schermo i simboli salvati in matrice, che non siano nulli. In questo modo il giocatore avrà sempre lo schema del gioco stampato a schermo.

0.4.7 Le questions

Ora dobbiamo gestire gli eventi domande. Ovvero gli indovinelli o indizi che interrogheremo durante il gioco. A questo scopo ho creato una classe apposita, di nome questions.

```
class Question:
    def __init__(self):
        self.pos=[]
        self.question=''
        self.answer=''
        self.object=''
        self.coordinateobj=[]
```

All'interno di essa si possono trovare rispettivamente. La posizione fisica della domanda inclusa in self.pos=[]. La domanda associata self.question=", la risposta esatta alla domanda self.answer=", l'oggetto generato (un oggetto dello schema, una apertura della porta) con self.object, la coordinata di posizione di tale oggetto self.coordinateobj=[]. Tale classe, sarà associata a un oggetto creato in seguito, per il quale dovrà essere compilato fisicamente, leggendo le istruzioni del file domande associato allo schema. Creato la classe, del quale verrà associato un oggetto, dobbiamo gestire gli eventi delle risposte agli enigmi e indovinelli. Per fare questo è stata creata una funzione di nome question_screening().

```
def question_screening(nfile):


    lista=[]
    fileschema='sch'+str(nfile)+'/questions.txt'
    q='*'
```

```
with open(fileschema) as f:
    while q!='':
        q=f.readline()
        if q!='':
            lista.append(q)
            questions.append(Question())

for (i,elemento) in enumerate(questions):
    segment=lista[i].split(':')

    for (j,info) in enumerate(segment):

        if j==0:
            table=info.split(',')

            questions[i].pos=[int(table[0]),int(table[1])]
        if j==1:
            table=info

            questions[i].question=table
        if j==2:
            table=info

            questions[i].answer=table
        if j==3:
            table=info

            questions[i].object=table
        if j==4:
            table=info.split(',')

            questions[i].coordinateobj=[int(table[0]),int(table[1])]
```

La funzione legge il file delle questions, creando una lista di oggetti associati alla classe question. Le variabili spiegate prima, vengono aggiornate sugli eventi scritti nel file questions.txt. All' interno di questa funzione vengono generati gli oggetti questions relativi allo schema giocato. I 6 dati di ogni stringa questions, vengono così processati e riconosciuti dal sistema, proprio tramite gli oggetti questions.

0.4.8 Inizializzazione e impostazione

Adesso vediamo l'inizializzazione e l'impostazione del gioco.

```
codelist=['cbgh','awer','prepo','minu','abrad','ulip','piol','jagbag','ulagu',

nfile=1
start=1
nschemi=len(codelist)+1
#verde = '\u001b[92m'
#rosso = '\u001b[31m'
os.system('cls')
print(codelist)
presentation()
print('Inserisci codice avventura:=')
codice=input('')
for (v,elemento) in enumerate(codelist):
    if elemento==codice:
        start=v+1
```

La prima variabile è una lista di codici che permette al giocatore di cominciare a giocare dall'ultima stanza visitata, utilizzando il codice dato a fine partita. La posizione dell'elemento di lista, sarà proprio il numero di riferimento dello schema (questo lo vediamo successivamente). In nschemi mettiamo i numeri totali degli schemi programmati. Con os.system('cls') si pulisce il terminale di Ms-dos. La chiamata di routine presentation(), richiama la presentazione iniziale del gioco. Con codice=input(), ci viene poprio chiesto il codice stanza. Il for successivo, esamina se il codice inserito sia corretto, in caso positivo aggiorna la variabile partenza start, altrimenti il giocatore dovrà ripartire dal primo schema.

Adesso arriva il cuore del codice, ovvero la parte che controlla la sequenza di chiamata delle funzioni e che gestisce le azioni del giocatore.

```
for s in range(start,nschemi):
    nfile=s
    n=60
    schema=[[0 for i in range(n)] for j in range(n)]
    schemafinale=[[0 for i in range(n)] for j in range(n)]
    hero=Player()
    val=read_schema(nfile)
    read_solution(nfile)
```

```
loop=True
key=''
mem='-'
os.system('cls')
screen_schema(n)
questions=[]
labels=['1','2','3','4','5','6','7','8','9']
question_screening(nfile)
```

Questa parte si apre con un ciclo for, che parte dal numero di schema associato al codice inserito (se inserito, altrimenti parte da 1), e arriva fino al totale numero di schemi (anche se in realtà, come vedremo successivamente il gioco può essere interrotto premendo q). nfile è proprio la variabile che identifica il numero dello schema. n è il massimo range della matrice che determina il massimo numero di caratteri per linea e colonna utilizzati dal gioco, qui settato a 60 numero sufficiente per il gioco in questione. Dopo il settaggio e dichiarazione delle variabili schema e shemafinale, viene creata la variabile oggetto hero, che acquisisce in questo modo tutte le variabili dichiarate nella classe Player(). La funzione read_solution() viene chiamata. Le dicharazioni successive sono tutte variabili che gesticono cicli e eventi. Con loop=True attiviamo il successivo ciclo while, key è la variabile che memorizza, il tasto premuto. Nella lista labels sono inseriti in nomi che etichetteranno gli oggetti dal punto di vista visivo. Con l'ultima riga si settano le variabili questions, che tramite la funzione question_screening(), va leggere i files di costruzione questions, associato allo schema di riferimento.

0.4.9 La gestione del ciclo di chiamata principale

Vediamo ora la macro parte successiva, che conclude la parte madre del codice.

```
while loop:

    keyboard.press_and_release('backspace')

    time.sleep(0.1)
```

Questa parte è inclusa in un macro loop, gestito dal comando while. Il comando successivo keyboard.press_and_release('backspace'), è il primo comando appartenente alla libreria keyboard che utilizziamo. Esso preme, e lascia il tasto backspace per ogni iterazione del loop, con scopo di ripulire tutti i tasti in memoria utilizzati. Il comando

time.sleep(0.1), appartenente alla libreria time, ferma per un decimo di secondo il ciclo, scandendo in questo modo la lettura dei tasti premuti. Comincia ora la parte di lettura di gestioni degli eventi, connessa alla lettura dei tasti.

```
if keyboard.is_pressed("q"):
 print(codelist[s-1])
    exit()
```

Questa operazione avviene, tramite il comando keyboars.is_pressed(). Il primo tasto evento scritto nel programma è il tasto q. Questo serve per uscire dal gioco tramite la funzione python exit(), dandoti prima il codice associato allo schema, tramite il comando print(codelist[s-1]). Dopo di questo comando, arriva la serie di controlli che gestiscono il movimento del giocatore (la chiocciolina).

```
if keyboard.is_pressed("up arrow"):
    if (schema[hero.y-1][hero.x]!='*') and
 (schema[hero.y-1][hero.x]!='X') and
(schema[hero.y-1][hero.x]!='^') and
(schema[hero.y-1][hero.x]!='<') and
(schema[hero.y-1][hero.x]!='>') and
(schema[hero.y-1][hero.x]!='v'):
          schema[hero.y][hero.x]=mem
          mem=schema[hero.y-1][hero.x]
          schema[hero.y-1][hero.x]='@'

          hero.y-=1
          os.system('cls')
          screen_schema(n)

if keyboard.is_pressed("down arrow"):

    if (schema[hero.y+1][hero.x]!='*') and
 (schema[hero.y+1][hero.x]!='X') and
 (schema[hero.y+1][hero.x]!='^') and
 (schema[hero.y+1][hero.x]!='<') and
 (schema[hero.y+1][hero.x]!='>')and
 (schema[hero.y+1][hero.x]!='v') :
```

```
            schema[hero.y][hero.x]=mem
            mem=schema[hero.y+1][hero.x]
            schema[hero.y+1][hero.x]='@'

            hero.y+=1
            os.system('cls')
            screen_schema(n)

if keyboard.is_pressed("right␣arrow"):
    if (schema[hero.y][hero.x+1]!='*') and
(schema[hero.y][hero.x+1]!='X') and
(schema[hero.y][hero.x+1]!='^') and
(schema[hero.y][hero.x+1]!='<') and
(schema[hero.y][hero.x+1]!='>') and
(schema[hero.y][hero.x+1]!='v'):

            schema[hero.y][hero.x]=mem
            mem=schema[hero.y][hero.x+1]
            schema[hero.y][hero.x+1]='@'

            hero.x+=1
            os.system('cls')
            screen_schema(n)

if keyboard.is_pressed("left␣arrow"):

    if (schema[hero.y][hero.x-1]!='*') and
 (schema[hero.y][hero.x-1]!='X') and
 (schema[hero.y][hero.x-1]!='^') and
 (schema[hero.y][hero.x-1]!='<') and
 (schema[hero.y][hero.x-1]!='>') and
 (schema[hero.y][hero.x-1]!='v'):

            schema[hero.y][hero.x]=mem
            mem=schema[hero.y][hero.x-1]
            schema[hero.y][hero.x-1]='@'
```

```
hero.x-=1
os.system('cls')
screen_schema(n)
```

Di questi ne commento solo il primo in quanto sono tutti simili. Tale controllo comincia con il comando if keyboard.is_pressed("up arrow"): che significa, se premi il tasto freccina up entri nel seguente if. Successivamente un altro if è implementato, concatenato da una serie di operatori and. Questo if controlla se il carattere corrispondente alla posizione dove la chiocciola dovrebbe andare, appartiene alla serie di caratteri leciti. I caratteri illeciti sono quelli che compaiono sulla riga concatenata da una serie di and. Questo significa che in caso il carattere apparentente alla matrice schema nella posizione, sopra il nostro eroe, è un carattere tra quelli inseriti nelle condizioni, come per esempio la X che rappresenta una porta, l'evento all'interno dell' if non si verificherà. In caso il carattere, appartiene a quei caratteri ammessi per l'entrata della condizione if, abbiamo il movimento del nostro eroe. Tale movimento è gestito da una variabile mem, che memorizza il carattere della casella di dove la chiocciola va a posarsi. Il risultato che al movimento successivo la nostra chiocciola viene sostituita proprio da questo carattere, e in mem, viene memorizzato il carattere sostituito dal carattere chiocciola come mostrato in figura. Alla fine lo schermo verrà pulito, e lo schema aggiornato verrà caricato a schermo.

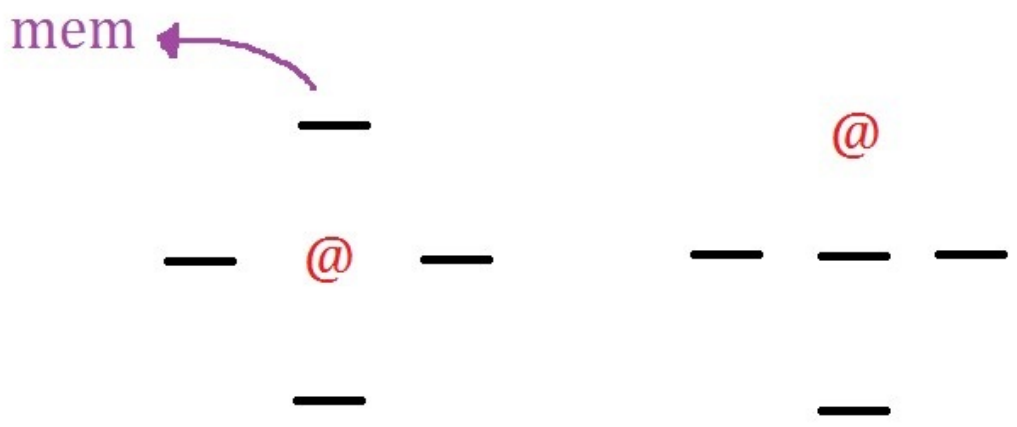

Figura 3: *Movimento del personaggio.*

```
if keyboard.is_pressed('h'):
    fileschema='help.txt'
    with open(fileschema) as f:
        page=f.read()
```

```
        print(page)
```

La piccola parte di codice sopra riportata, permette premendo h di avere le istruzioni per giocare al gioco. Veniamo ora alla parte dei comandi che permette al nostro personaggio l'interazione con gli oggetti e opzioni evento all'interno dello schema. I tasti sono due. Il tasto c per prendere oggetti, chiedere quesiti sui punti di domanda e aprire il cancello della stanza. Il tasto d per lasciare uno degli oggetti che si possiede.

```
if keyboard.is_pressed("c"):

    keyboard.press_and_release('backspace')
    if mem>='A' and mem<='Z':
        if len(hero.objects)<9:
            hero.objects.append(mem)
            mem='-'
    if mem=='?':

        for elemento in questions:
            if elemento.pos[0]==hero.x and elemento.pos[1]==hero.y:

                answer=input(str(elemento.question))

                if answer==str(elemento.answer):

                    if schema[int(elemento.coordinateobj[1])]
                [int(elemento.coordinateobj[0])]!='O':
                        schema[int(elemento.coordinateobj[1])]
                [int(elemento.coordinateobj[0])]=elemento.object
                        elemento.answer='sasfjsdfkowqkr2'
    if mem=='g':

        if check_solution(n)==False:
            print('You done!')
            loop=False

if keyboard.is_pressed("d") and mem=='-':
    os.system('cls')
    screen_schema(n)
```

```
print (labels)
print (hero.objects)
choice=-1
keyboard.press_and_release('backspace')
while choice==-1:
    if keyboard.is_pressed("0"):
        choice=-2
    if keyboard.is_pressed("1"):
        choice=1
    if keyboard.is_pressed("2"):
        choice=2
    if keyboard.is_pressed("3"):
        choice=3
    if keyboard.is_pressed("4"):
        choice=4
    if keyboard.is_pressed("5"):
        choice=5
    if keyboard.is_pressed("6"):
        choice=6
    if keyboard.is_pressed("7"):
        choice=7
    if keyboard.is_pressed("8"):
        choice=8
    if keyboard.is_pressed("9"):
        choice=9

if choice>0:
    if len(hero.objects)>=choice:
        schema[hero.y][hero.x]=hero.objects[choice-1]
        mem=hero.objects[choice-1]
        hero.objects.pop(choice-1)
```

Partiamo dalla condizione che gestisce il tasto c, sopra descritto. Premendo il tasto c, si entra nella condizione dell'azione del personaggio. All'interno di tale condizione ci sono ben tre sottocondizioni, dipendenti dalla variabile mem. Se in mem, c'è un carattere letterale maiuscolo che va dalla A alla Z, viene memorizzata nella variabile lista oggetti del giocatore la lettera stessa, e in mem viene assegnato il carattere pedonabile -. Questo evento si verifica fintanto che la lunghezza della lista è minore di 9. Nel caso il carattere

è il punto interrogativo, si apre un ciclo for per la gestione delle domande o enigmi, spiegati precedentemente.

0.5 Facciamo una partita.

In questo paragrafo vediamo alcuni schemi da risolvere, quindi in caso volete giocarci non leggetelo, altrimenti leggete solo lo schema che non riuscite a risolvere (se è tra quelli citati qui). Alcuni schemi possono risultare particolarmente difficili o di difficile interpretazione, per enigmi troppo complessi, o per la difficoltà a volte di capire il da farsi per evolvere la soluzione della stanza. Quello che è importante è capire che una volta creato un codice, che dia la disponibilità di creare schemi facilmente, ogni persona può dare sfogo la propria fantasia e ragione migliorando sempre il design e giocabilità di ogni singola stanza.

0.5.1 Schema numero 4

Lo schema iniziale si presenta con un oggetto e una parete laterale aperta e due punti di domanda (figura 4). Andando sul punto di domanda vicino al cancello e interrogandolo, otteniamo il seguente indovinello "Un mezzo ne è un terzo. Che cos'è?". La risposta al quesito è 1.5. La risposta esatta a questo quesito fa comparire un altro oggetto. Possiamo prendere entrambi gli oggetti e interrogare il punto interrogativo rimasto. Il quesito uscente dal punto interrogativo è il seguente "Solo arginando il fiume ti salverai". Questo è evidentemente un indizio per risolvere lo schema dato. Guardando lo schema notiamo un muro (gli asterischi), che lascia, due spazi aperti. Questo può essere quindi l'argine del fiume da arginare inteso dall'enigma. La soluzione quindi sarà quindi la chiusura, di tale corridoio con i due oggetti presi!

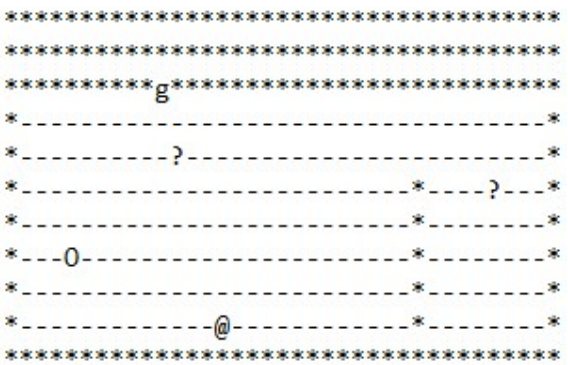

Figura 4: *Schema iniziale dello schema numero 4.*

0.5.2 Schema numero 5

In questo schema troviamo una porta davanti al cancello di uscita. Vediamo anche una stanza chiusa con all'interno un oggetto da prendere che probabilmente ci servirà a risolvere il puzzle. Per aprire la porta che chiude la stanza, dovremo rispondere a qualche quesito.

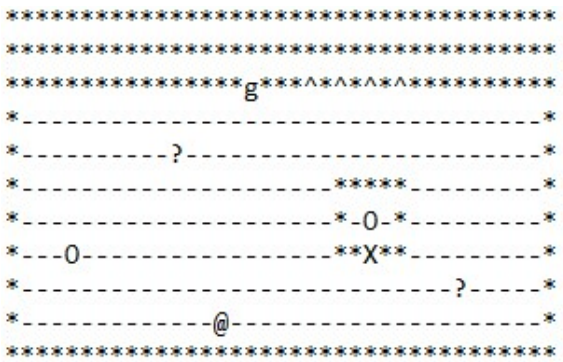

Figura 5: *Schema iniziale dello schema numero 5.*

Muoviamoci sul punto di domanda in basso a destra e lo consultiamo. L'indovinello uscente da questa consultazione sarà il seguente. "Navi che salpano, una ogni 12 ne fa ritorno, l'altra ogni 15 e la più lenta ogni 20, oggi dopo tanti giorni possono festeggiare....". Questo è un classico quesito matematico fatto alle scuole medie, scritto in una forma più enigmatica. Quindi dobbiamo dare una risposta matematica. Si tratta di dire dopo quanti giorni tali navi si ritroveranno allo stesso punto. Per rispondere bisogna trovare il minimo numero che divide in modo intero, i giorni dati, dando in questo modo il numero di viaggi completi fatti da ciascuna nave. Tutto questo porta al calcolo del minimo comune multiplo, che nel nostro caso è 60. La risposta esatta apre la porta della stanza come si può notare in figura 6, quindi sbloccando parte del puzzle.

Aprendo la porta possiamo raccogliere l'oggetto e successivamente anche l'altro oggetto rimasto nello schema. Rimane l'altro punto interrogativo, da consultare che dice, "solo i 5 passarono". In questo caso il tentativo di dare una risposta a tale quesito è inutile, in quanto esso si riferisce a un' azione che devi svolgere. Guardando lo schema si nota che ci sono dei segni, (^) a destra del cancello (gate). Questa parte del quesito può risultare particolarmente complicata per chi non capisce che tali simboli vanno utlizzati. La soluzione sta nella consapevolezza che bisogna utilizzare gli oggetti che la stanza ci ha permesso di raccogliere. Avendo due oggetti in mano e una volta consapevoli che tali oggetti sono correlati in qualche como ai simboli sulla parete, ci rimane da interpretare il secondo quesito della stanza. La soluzione sta che il numero cinque può essere scritto con cifre binarie (il costruttore della torre era un signore molto avanti con i tempi) e bastano

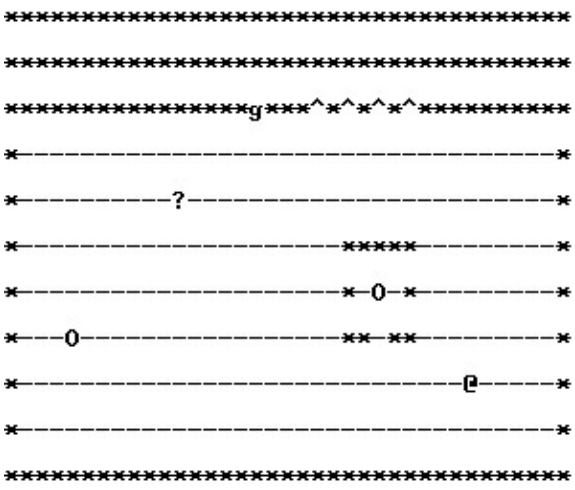

Figura 6: *Apertura porta schema numero 5.*

solo due unità per avere tale. Quindi posando tali oggetti davanti ai simboli, intendendo il numero 5, in forma binaria, sarà la soluzione risolutiva dello schema. Questo significa che i due oggetti vanno lasciati sotto il secondo e l'ultimo simbolo impressi nel muro.

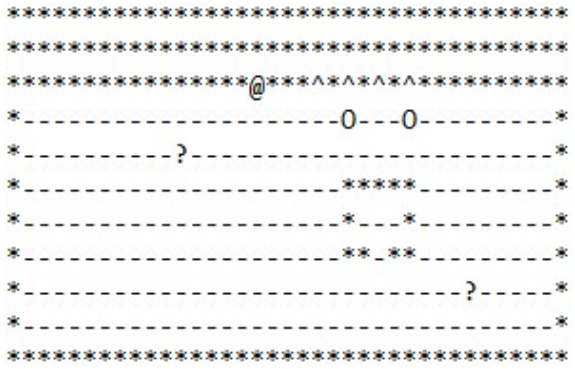

Figura 7: *Schema finale dello schema numero 5.*

0.6 Conclusioni

In questo breve testo abbiamo visto, come è possibile realizzare un gioco interessante utilizzando il minimo necessario che un personal computer può darci, ovvero dei caratteri ASCII e un linguaggio di programmazione. Questo semplice gioco (ma per questo non meno interessante della moltitudine di giochi che si trovano su internet) porta l'interessato a imparare metodi basilari di programmazione e in particolare modo a pensare alla progettazione di un gioco senza esserne un esperto nel campo. Adesso tocca a voi lettori progettare il prossimo textgame!