***PROGETTO MAZE MUNA - MINA - COSTA***

**Link utili:**

**Generazione maze:** <https://youtu.be/Y37-gB83HKE>

<https://en.wikipedia.org/wiki/Maze_generation_algorithm>

**Risoluzione maze:** <https://youtu.be/yuf8cd1eypA>

**Qui quelli di risoluzione:** <https://en.wikipedia.org/wiki/Maze_solving_algorithm>

**Qua è spiegato passo passo un algoritmo di risoluzione:** <https://www.cs.bu.edu/teaching/alg/maze/>

**DOC Spanò:** <https://moodle.unive.it/mod/assign/view.php?id=127699>

**ASSEMBLY INFO**

Nessuna modifica al codice di Spanò

**PRELUDE**

Nessuna modifica al codice di Spanò

**CONFIG**

impostati i simboli del muro e del player (‘/219’ sta per il quadrato pieno)

**EXTERNALS**

Nessuna modifica al codice di Spanò

**GLOBALS**

Nessuna modifica al codice di Spanò

**GFX**

* aggiunto alla riga 32 comando per colorare e rappresentare un elemento grafico di tipo muro:
  + static member filled\_wall (fg : Color, ?bg : Color) = CharInfo.create (Config.wall\_pixel\_char, fg, ?bg = bg)
* Commentato codice alla riga 237-238 relativo alla scrittura di messaggi in console sulla creazione della finestra:
  + Log.msg "Console info:\n\tBufferWidth = %d\n\tBufferHeight = %d\n\tWindowWidth = %d\n\tWindowHeight = %d\n" Console.BufferWidth Console.BufferHeight Console.WindowWidth Console.WindowHeight

**ENGINE**

* Aggiunto alla riga 14 una variabile globale ShowFps che permetterà poi di selezionare se visionare o no gli FPS
  + let mutable public ShowFPS = false
* Riga 68 – 69 commentato il codice che stampava sulla console la registrazione di uno Sprite e aggiunto le due parantesi tonde (poiché la funzione doveva ritornare necessariamente tipo unit):
  + Log.msg "registered sprite #%d: x=%g y=%g z=%d width=%d height=%d" len spr.x spr.y spr.z spr.width spr.height
* Riga 83 sostituito il valore true di default che mostrava gli FPS (solo nel debug) con la variabile istanziata precedentemente, il valore false nella release viene lasciato così com'è:
* Commentati altri log messages alle righe 110 – 115 - 117 – 125 – 146

**TEST**

Nessuna modifica al codice di Spanò

**MAIN**

Eliminato completamente codice di spanò e le righe commentate permettono di avere una musica di sottofondo

**MAZE**

* Rimosso il do.generate🡪 E’ quindi necessario richiamare il costruttore del labirinto quando viene istanziato un oggetto

Osserviamo gli attributi della classe Maze:

* **walls (TASK 1):**
  + Matrice che rappresenta il labirinto, deve avere per forza dimensioni dispari, di default è settata su true:
    - True🡪rappresenta una cella dove c’è un muro
    - False🡪rappresenta una cella del percorso del labirinto
* **resolved (TASK 2):**
  + Matrice che serve a ricavare la soluzione:
    - 0-> via libera (alla fine rappresenterà la soluzione finale)
    - 1-> cella già visitata e bloccata
    - 2-> cella da controllare
    - 3-> valore iniziale della matrice

Osserviamo ora i vari metodi della classe Maze:

* **Generate:**
  + setta la cella (1;1) a false poiché il percorso deve partire da lì
  + funzione ricorsiva **create**:
    - il primo if si riferisce al caso base: le celle da visitare sono( w/2 \* h/2)-1
      * il -1 poiché la prima cella è già stata visitata
    - il secondo if controlla che una cella non sia mai stata visitata facendo attenzione di non controllare fuori dai limiti della matrice. Nel caso non ci siano celle libere (ma non abbiamo ancora visitato tutte le celle) è necessario fare un backtracking (utilizzando una lista passata come parametro che memorizza le ultime posizioni)
      * il controllo viene fatto poiché secondo la base dell’algoritmo metà caselle sono muri (messi a scacchiera) che nel caso siano scelti vengono eliminati🡪 per questo motivo la cella adiacente da controllare non è quella subito successiva alla nostra cella ma viene shiftata di una posizione: la cella a destra da controllare di (1;1) è (1;3) e nel caso venga selezionata il muro da rimuovere è quello in posizione (1;2). Come è ovvio non si può controllare a cella sopra e quella a sinistra della (1;1).
    - Se le celle non sono state visitate viene lanciato un random che determina in quale delle 4 direzioni andare:
      * 1🡪 sopra
      * 2🡪 sotto
      * 3🡪 sinistra
      * 4🡪 destra
    - Un if controlla di non uscire dai bordi della matrice e che la cella scelta non sia una di quelle già visitate→ se no richiama la funzione con un altro numero random
    - Ora nei vari casi viene generalmente effettuata questa operazione:
      * la cella diventa visitata (quindi walls i j=true)🡪 fa parte del percorso
      * Il muro tra le due celle viene eliminato🡪 diventa walls i j=false
      * viene richiamata la funzione🡪
        + incremento contatore
        + la prossima cella è quella appena visitata
        + cella viene aggiunta alla lista delle ultime posizioni
  + viene quindi lanciata la funzione ricorsiva segnando già una cella visitata e partendo dalla (1;1)
* **generateUscitaCasuale:** 
  + genera un'uscita controllando che di fianco all’ipotetica uscita sia presente il percorso altrimenti la funzione si richiama
* **generateUscitaAutoResolve:**
  + genera un'entrata in alto a sinistra e un’uscita in basso a destra (uscite di default)
* **StartPlayer:**
  + genera una coppia con le coordinate casuali di un player all’interno del labirinto (controllando di non generarlo su un muro )🡪 NON FUNZIONA
* **Autoresolve:**
  + deadpoint:

ricava i deadpoint:

* + - doppio for per scansionare la matrice del labirinto
      * Controlla se la cella selezionata ha intorno tre muri nei quattro casi possibili🡪 nel caso fosse vero la matrice associata alla risoluzione assegna il valore 2 a quel punto mettendolo come cella da cui partire a controllare, in tutti gli altri casi setta 0
  + fino\_al\_bivio:
    - primo if🡪 caso base contatore fino ad un numero grande🡪 CORREGGERE
    - doppio for per scansionare la matrice (escludendo i bordi, come prima)
    - Se la cella è una da visitare: (if matrice (i;j)=2)
      * Controlliamo che intorno abbia un solo possibile percorso ricordando che il percorso non può continuare se è presente un muro o se è una cella già visitata e bloccata quindi se:
        + la prima matrice ha valore true
        + La seconda matrice ha valore 1
      * Nel caso ci sia un solo percorso la prossima cella del percorso diventa 2 e quella su cui ci troviamo invece diventa 1
      * Altrimenti se abbiamo più di un percorso disponibile vuol dire che ci troviamo in un bivio, bisogna quindi mettere la cella a 0 e quel percorso la risulta bloccato (poiché tutti i precedenti sono 1 e non c’è neanche più un 2)

**NORMAL\_MODE**

Ricordando il funzionamento della classe Demo del motore di Spanò:

* w e h diventano variabili in modo da poter essere modificate nel menu
* Aggiunto un comando che permette di non buggare lo schermo passando dal menu ad un labirinto di dimensione minore:
  + Gfx.system\_console\_raster.at (0,0,Color.Black)
* Richiamo e generazione del labirinto:
  + let mymaze = new maze (w,h)

mymaze.generate

mymaze.generateUscitaCasuale mymaze.walls

* Controllo sui comandi:
  + w,a,s,d controlla che non si possa uscire dalla matrice e quindi dallo schermo, inoltre controlla che non ci si possa muovere se l’elemento della matrice nella posizione adiacente del player è true (cioè se è presente un muro)
  + Controllo ulteriore se la posizione del player è in uno dei bordi (possibile solo nel caso dell’uscita) in quel caso genera un rettangolo nero che copre tutto lo schermo e ci scrive sopra (centrandolo) SUCCESS, PRESS ESC/Q TO EXIT
  + Infatti, nell’ultima parte della funzione ho aggiunto che si possa uscire anche con esc:
    - st, key.KeyChar = '\027' || key.KeyChar = 'q'
  + Con il doppio for è molto semplice disegnare il labirinto (dove la matrice è true viene usato il colore del “path”, altrimenti viene utilizzato il colore del “wall”, presenti nella classe Maze)
  + Ricavo poi le coordinate del player
  + Infine, disegno il player (rettangolo di dimensione 1,1)

**AUTORESOLVEMODE**

Il principale cambiamento sta nella variabile “mode” che permette di selezionare la modalità nel menu, e nell’inizializzazione del Maze:

mymaze.generate

mymaze.generateUscitaAutoResolve mymaze.walls

mymaze.autoresolve mymaze.walls

Osserviamo ora le tre modalità:

**SOLUTION**

La modifica principale al codice di prima è che nel momento in cui viene disegnato il labirinto (doppio for) viene disegnata anche la soluzione utilizzando la matrice resolved (quando è uguale a 0)🡪 riga 76, 77

Inoltre, la visibilità del player (ultimo parametro del metodo createAndRegisterSprite) viene messa a -1 così che il player non venga visualizzato anche se presente

Questa modalità mostra semplicemente la soluzione

**PLAYABLE**

A differenza della precedente modalità, la visibilità del player resta normale, ma ogni volta che il giocatore si muove viene disegnato un rettangolo con visibilità 2 (in modo da sovrascrivere il giallo della soluzione) del colore del path all’interno del metodo my\_update

In questo modo il giocatore è libero di muoversi cancellando man mano la via della soluzione

**ANIMATION**

Questa modalità è quella un po’ più complicata: infatti il metodo my\_update viene sostituito dal metodo my\_update\_animation.

Inoltre, la soluzione non viene rappresentata subito.

Nel metodo my\_update\_animation il player si muove verso l’unica direzione disponibile rappresentata dallo 0 nella matrice resolved qualunque sia il tasto premuto. Una volta mosso però bisogna modificare il valore della cella precedente altrimenti il player avrebbe due possibili strade in cui il valore è 0 (viene quindi assegnato 4) e come prima viene disegnato un rettangolo 1-1 di visibilità 2 e colore giallo che sovrascrive il path e mostra man mano la soluzione

**MENU**

Ora vediamo in generale il funzionamento del menu che è uguale per tutte e quattro le classi:

Si basa sempre sulla classe Demo di Spanò dove il player è il selezionatore del menu e può muoversi solo in determinate posizioni definite all’interno del my\_update, se poi si trova in una posizione e viene premuto invio esegue un comando come quello che richiama il main etc...

Le scritte del menu sono dei rettangoli con all’interno del testo.

Per quanto riguarda le cose da selezionare, come la modalità o la larghezza/altezza, semplicemente ogni volta che viene premuto a/d in quella determinata posizione viene sovrascritta la scritta facendo attenzione a qualche particolare (tipo se le cifre della larghezza superano le due cifre allora il nella scritta viene tolto uno spazio per stare dentro i 16 pixel del player)

Infine, per evitare bug viene aggiornato ad ogni tick l’engine nel metodo my\_update:

engine <- new engine (w, h)