Progetto 2048 Cannas-Sarritzu

July 10, 2014

Lo scopo del progetto è implementare all'interno del gioco 2048, già sviluppato da B. Borges, un giocatore automatico che, in caso di necessità da parte del giocatere, effettui una mossa.

Nel progetto dovrà essere modificata la classe *GameManager* contenente gli oggetti grafici del gioco. Si farà utilizzo anche di alcuni oggetti definiti da B.Borges, come le *Location* e i *Tile* che rappresentano rispettivamente le coordinate delle caselle e il riempimento delle stesse, entrambe vengono conservate all'interno di *GameManager* nell'attributo *gameGrid* il quale è un oggetto di tipo HashMap.

Per iniziare il progetto sono state create le due interfacce richieste dalle specifiche, *Griglia* e *GiocatoreAutomatico,* e successivamente le due classi che le implementano: *MyGiocatoreAutomatico*, nel package *giocatoreAutomatico.player* e *MyGriglia*, nel package *game2048* (essa è stata inserita in questo package al fine di poter applicare la nostra interfaccia al programma creato da altri gruppi e viceversa).

MyGiocatoreAutomatico è una classe che possiede un costruttore vuoto ed il metodo *prossimaMossa(Griglia g)*.

MyGriglia è invece una classe che contiene solamente un attributo ed un costruttore. L'attributo è un oggetto di tipo *HashMap<Location,Integer>* e il costruttore non prende valori in input ma crea una nuova HashMap assegnandola all'attributo. Gli oggetti *MyGriglia* vengono usati per copiare la griglia del gioco (*gameGrid*), la quale è un attributo privato della classe *GameManager* e quindi inutilizzabile per creare il giocatore automatico. Il costruttore di *MyGriglia* è stato richiamato nella classe *GameManager* all'interno dell'evento *gameHelp()*, il quale viene lanciato dal bottone *aiuto* definito nel metodo *createScore()*. Abbiamo inoltre modificato il CSS per ottenere la grafica del nuovo bottone simile al resto del gioco.

L'evento *gameHelp()* crea un nuovo *MyGiocatoreAutomatico*, una nuova *MyGriglia* e, attraverso due cicli for, sono state collocate le *Location* di *gameGrid* all'interno di *MyGriglia* inserendo, se presente un *Tile*, il valore corrispondente altrimenti il valore -1. Successivamente è stato definito un intero inizializzato attraverso il metodo *prossimaMossa(Griglia g)* implementato nella classe *MyGiocatoreAutomatico*. Questo metodo prende in input l'oggetto *MyGriglia* costru-

ito in precedenza e restituisce un numero casuale da 0 a 3 che rappresenta una nuova mossa. La mossa implementata "non è intelligente", in generale questo comporta la possibilità che cliccando il bottone non venga modificata la situazione del gioco. Per evitare ciò è stato implementato il metodo come segue.

Per prima cosa è stato scritto un ciclo do-while la cui condizione d'uscita è che una variabile booleana assuma valore false; al suo interno abbiamo creato l'intero random che rappresenta la mossa automatica, si consideri il caso in cui essa sia 0 (per gli altri casi il ragionamento è analogo). La mossa implementata deve mandare le celle verso l'alto, condizione che è possibile se ci sono almeno due *Tile* di pari valore nella stessa colonna oppure se vi è almeno una cella vuota nelle prime tre righe al di sotto della quale vi è un *Tile*. Il metodo accetta la mossa come valida solo se è verificato il secondo di questi due casi, infatti sono stati implementati due cicli for (uno per scorrere le quattro colonne e uno per le tre righe) per cercare una *Location* vuota. Una volta trovata la *Location* si controlla, attraverso un terzo ciclo for, che al di sotto di essa vi sia almeno una *Location* con valore diverso da -1 (cioè che nella griglia iniziale, *gameGrid*, ci sia un *Tile*). Se ciò accade la variabile booleana diventa false, si esce dal do-while e viene restituito il numero 0.

Questo metodo, così descritto, presenta un problema quando *gameGride* ha tutte le celle piene, infatti il ciclo do-while non terminerebbe mai. Per evitare questo problema sono state inizializzate altre quattro variabili booleane con false che vengono modificate nel momento in cui inizia il controllo su ciascuna mossa. Se la griglia è piena, dopo almeno quattro iterazioni del do-while diventano tutte e quattro true e attraverso un controllo viene restituita l'ultima mossa random creata.

Una volta restituita la mossa nell'evento *gameHelp()*, il programma controlla il numero restituito e associa ad esso la direzione che, attraverso il metodo *move(),* viene applicata alla griglia del gioco la quale verrà finalmente modificata.

In conclusione, si osserva che il metodo *prossimaMossa(Griglia g)* controlli in maniera quasi soddisfaciente che la mossa possa essere effettivamente compiuta dal gioco, anche se nel caso in cui la griglia sia piena potrebbe accadere che lo stato del gioco non venga modificato. Per evitare ciò si potrebbe aggiungere una porzione di codice che controlli attraverso le *Location* di *MyGriglia* che siano presenti, nella stessa colonna o riga, almeno due valori uguali.

Inoltre come precisato in precedenza, la mossa non è ottimale per vincere il gioco. Per ottenere una mossa di questo tipo bisognerebbe approfondire argomenti di intelligenza artificiale.