

**گزارش پروژه مبانی منطق**

**برنامه حل مسئله بازی هیولا**

**توسط:**

سید حسین محمدی

**زیر نظر استاد محترم دکتر هادی فراهانی**

# مقدمه

انتظار میرود که با قوانین بازی آشنایی داشته باشید، پس برای یادآوری: هدف آن است که در نقشه ای از خانه شروع به خانه طلا برسیم بدونه آنکه بمیریم (حرکت بر روی خانه شامل هیولا یا دره) و در هر حرکت به یکی از خانه اطراف میتوان حرکت کرد. هر خانه در کنار هیولا، "بو" میدهند و خانه های کناره دره، "نسیم" میوزند.

حال برای سادگی چند جمله را در اینجا تعریف:

خانه های اطراف(around): اشاره به 4 خانه دار بالا، پایین، چپ و راست خانه دیگر.

عامل مرگ: عناصری مانند هیولا و یا دره که میتوانند باعث مرگ شوند.

خانه مرگ: خانه ای که در آن عامل مرگ باشد. (مثل خانه ای که در آن هیولا است)

عامل خطر: عناصری مانند "بو" و یا "نسیم".

خانه خطر(Hazard Tile): خانه ای که در آن عوامل خطر باشند. (مثل خانه های در اطراف خانه مرگ)

خانه عادی(Normal Tile): برعکس خانه خطر (بدون خطر)

خانه با احتمال مرگ(GuessDeath): خانه هایی که در اطراف خانه خطر قرار دارند.

خانه امن (SafeTile): خانه هایی که می‌دانیم امن‌اند.

خانه طلا: خانه ای که در آن طلا قرار دارد.

با در نظر گیری این قوانین، میتوان دو نتیجه کلیدی گرفت که:

1- اگر خانه ای عادی باشد، آنگاه خانه های اطراف آن نمیتوانند خانه مرگ باشند.

2- اگر خانه ای خطر باشد، آنگاه خانه های اطراف آن نمیتوانند احتمال مرگ خواهند داشت.

حال به نحو پردازش در برنامه خواهیم پرداخت:

-ابتدا جزئیاتی درباره نقشه خواهیم داد که به شکل زیر اند:

یک ترم برای نشان دادن سایز نقشه. مثل نقشه 4\*4:

mapSize(4,4).

یک رابطه برای نشان دادن نقطه شروع. مثل:

startPoint(1,1).

تعدادی رابطه برای نشان دادن خانه های مرگ. مثل:

deathTile(hole,4,2).

deathTile(monster,1,3).

تعدادی رابطه برای نشان دادن خانه طلا. مثل:

gold(4,4).

-همچنین روابطی میسازیم که با کمک به ترم های دیگر دسترسی یابیم مورد نیاز خود را دریافت کنیم:

یک رابطه که مختصات (i,j) میگیرد و مختصات خانه های اطراف را به عنوان (x,y) می‌پردازد:

/\* Rule to check all adjacent tiles\*/

around(X1, Y1, X2, Y2) :- X is I, Y is J+1;

Y is J, X is I+1;

X is I, Y is J-1;

Y is J, X is I-1.

یک رابطه که مشخص میکند که آیا مختصات (i,j) در نقشه قرار دارد یا خیر:

/\*check if in map\*/

inMap(I,J) :- mapSize(M,N), I < M+1, J < N+1, I > 0, J > 0.

دو رابطه که مختصات اطراف خانه (i,j) گرفته و به لیستی می‌افزاید و نتیجه را به صورت یک لیست جدید می‌دهد می‌پردازد:

/\*Rule to add surrounding tiles to guess-lists\*/

removeAroundTile(I,J,GuessList,NewGL) :- remove([[I+1,J],[I-1,J],[I,J+1],[I,J-1]]

,GuessList,NewGL).

addAroundTile(I,J,GuessList,NewGL) :- remove([[I+1,J],[I-1,J],[I,J+1],[I,J-1]]

,GuessList,TempL), append([[I+1,J],[I-1,J],[I,J+1],[I,J-1]],TempL,NewGL).

دو رابطه برای جمع‌آوری اطلاعات درباره خانه های خطر و مرگ برای راحتی کار:

/\*hazardous Tile\*/

mapHazard(Death,I,J) :- around(I,J,X,Y), inMap(X,Y), deathTile(Death,X,Y).

/\*collecting hazards catalog\*/

danger(I,J) :- mapHazard(Hazard,I,J).

dead(I,J) :- deathTile(T,I,J).

چند رابطه که به عنوان ابزار مورد استفاده قرار میگیرند:

/\*Tools\*/

/\*checks if x is in a list\*/

member(X, [X|\_]).

member(X, [\_|T]):- member(X, T).

/\*removes contents of one list from another list\*/

remove(\_, [], []).

remove(Targets, [X|Tail], Result):- member(X, Targets), !, remove(Targets, Tail, Result).

remove(Targets, [X|Tail], [X|Result]):- remove(Targets, Tail, Result).

/\*adds X to list\*/

add(X,[],[X]).

add(X,[H|T],[H|L]) :- add(X,T,L).

/\*appends 2 lists\*/

append(X,[],Result).

append(Z,[X|Y],[X|W]) :- append(Z,Y,W).

-حال رابطه ای مینویسیم که تمام این اطلاعات را جمع‌آوری کرده و براساس آن تصمیم می‌گیرد.

آن را safeStepFrom(I,J,HistoryList,SafeList,GuessDeath) مینامیم که به معنا حرکت کردن به مختصات i و j می‌باشد.

HistoryList لیست مسیر پیموده تا الان، SafeList لیست خانه هایی که می‌دانیم امن‌اند (در آن خانه‌ها نمی‌میرم) و GuessDeath لیست خانه ها با احتمال مرگ‌اند.

هربار که این رابطه را صدا زنیم به معنای آن است که به آن خانه با مختصات i و j حرکت کردیم، به ترتیب دو حالت خواهیم داشت:

1- بروی خانه خطر قرار داریم ؛ که آنگاه خانه های اطراف آن را به لیست خانه ها با احتمال مرگ (GuessDeath) می‌افزاییم، سپس خانه هایی که میدانیم امن‌اند (SafeList) را از آن کم میکنیم. حال خانه‌ای در اطراف که درون لیست GuessDeath نیست انتخاب میکنیم و به آن حرکت می‌کنیم (پس safeStepFrom برای موقعیت جدید مورد برسی میگذاریم).

2- بروی خانه عادی قرار داریم؛ که آنگاه خانه های اطراف را به لیست خانه های امن می افزاییم و سپس خانه هایی که میدانیم امن‌اند (SafeList) را از آن کم میکنیم. حال خانه‌ای قبلاً به آن رفتیم را انتخاب میکنیم و به آن حرکت می‌کنیم (پس safeStepFrom برای موقعیت جدید مورد برسی میگذاریم).

این کار آنقدر ادامه میابد تا به طلا برسیم، برای آن باید رابطه ای از safeStepFrom اختصاصاً برای چنین وضعیتی درست کنیم. (این رابطه باید در بالای حالت 1 و 2 باشند)

سرانجام برای راحتی کار رابطه ای مانند start میسازیم که چک کند که آیا کاراکتر را میتوان در خانه شروع قرار داد (خانه بیرون از نقشه نباشد و خانه مرگ نباشد) و سپس کاراکتر را در خانه شروع قرار دهد (safeStepFrom را برای آن خانه اجرا کند). کد آن به شکل زیر است:

/\*Targets\*/

safeStepFrom(I,J,[\_|Path],Irrelevent1,Irrelevent2) :- gold(I,J), inMap(I,J)

, write("path taken: " + [[I,J]|Path]).

/\*path checking (can be upgraded to have more holes and monsters)\*/

safeStepFrom(I,J,HistoryList,SafeList,GuessDeath) :-

%if we landed on a hazardus tile

danger(I,J), addAroundTile(I,J,GuessDeath,TempGD)

, remove(SafeList, TempGD, NewGD), around(I,J,X,Y), inMap(X,Y)

, not(member([X,Y],NewGD),safeStepFrom(X,Y,[[X,Y]|HistoryList],[[X,Y]|SafeList],NewGD);

%if we landed on a normal tile

not(danger(I,J)), addAroundTile(I,J,SafeList,NewSL), around(I,J,X,Y)

, inMap(X,Y), not(member([X,Y],HistoryList))

, safeStepFrom(X,Y,[[X,Y]|HistoryList],[[X,Y]|NewSL],NewGD).

/\*check if there is an assured path from start point to the gold\*/

start :- startPoint(I,J), inMap(I,J), not(dead(I,J))

,safeStepFrom(I,J,[[I,J]],[[I,J]],[]).

امکانات این روش

خوبی این برنامه آن است که میتوان به راحتی در آن تغییر ایجاد کرد، برای مثال میتوان تعداد عامل های مرگ را بیشتر کرد و یا میتوان نقشه را بزرگ تر یا کوچکتر نمود و حتی با تغییر رابطه around، نقشه با شکل غیر مستطیلی ساخت.

همچنین در صورت نیاز میتوان به راحتی قوانینی جدید به بازی افزود برای مثال، افزودن مکانیک "کشتن هیولا":  
 درصورت دانستن تعداد هیولا های موجود، با ایجاد لیستی جدید که خانه ها با احتمال هیولا را در خود دارد، میتوان نتیجه گرفت که اگر تعداد عناصر متمایز درون آن با تعداد هیولا ها برابر شود، آنگاه، هر عنصر درون این لیست، مکان هیولا را مشخص می‌نماید. برای راحتی میتوان تمام عملیات انجام شده بروی لیست های مرتبط به عامل های خطر را در یک رابطه گذاشت.