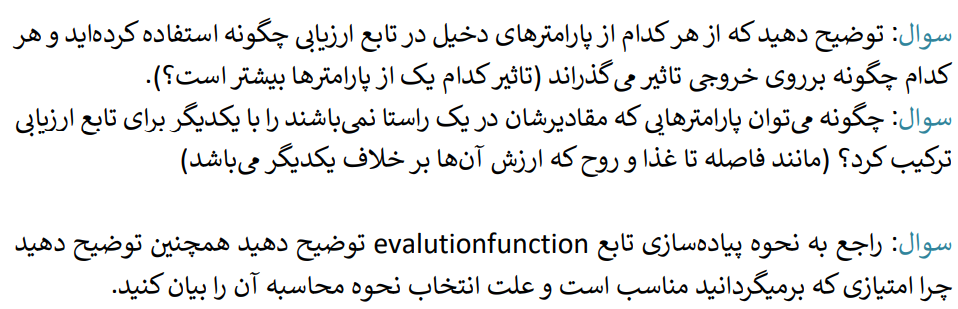
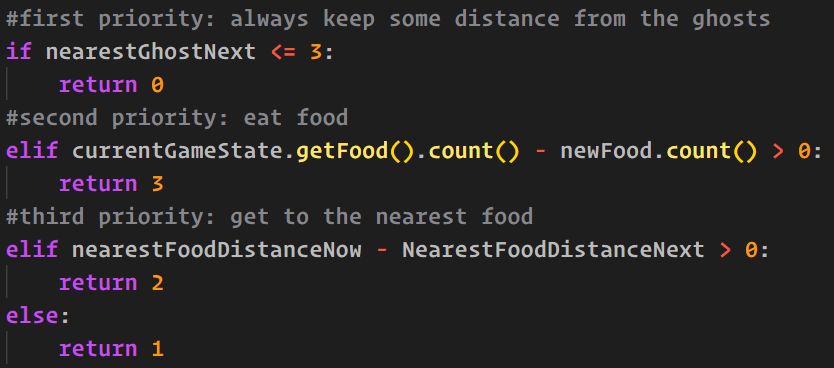
**1) عامل عکس العمل**

****

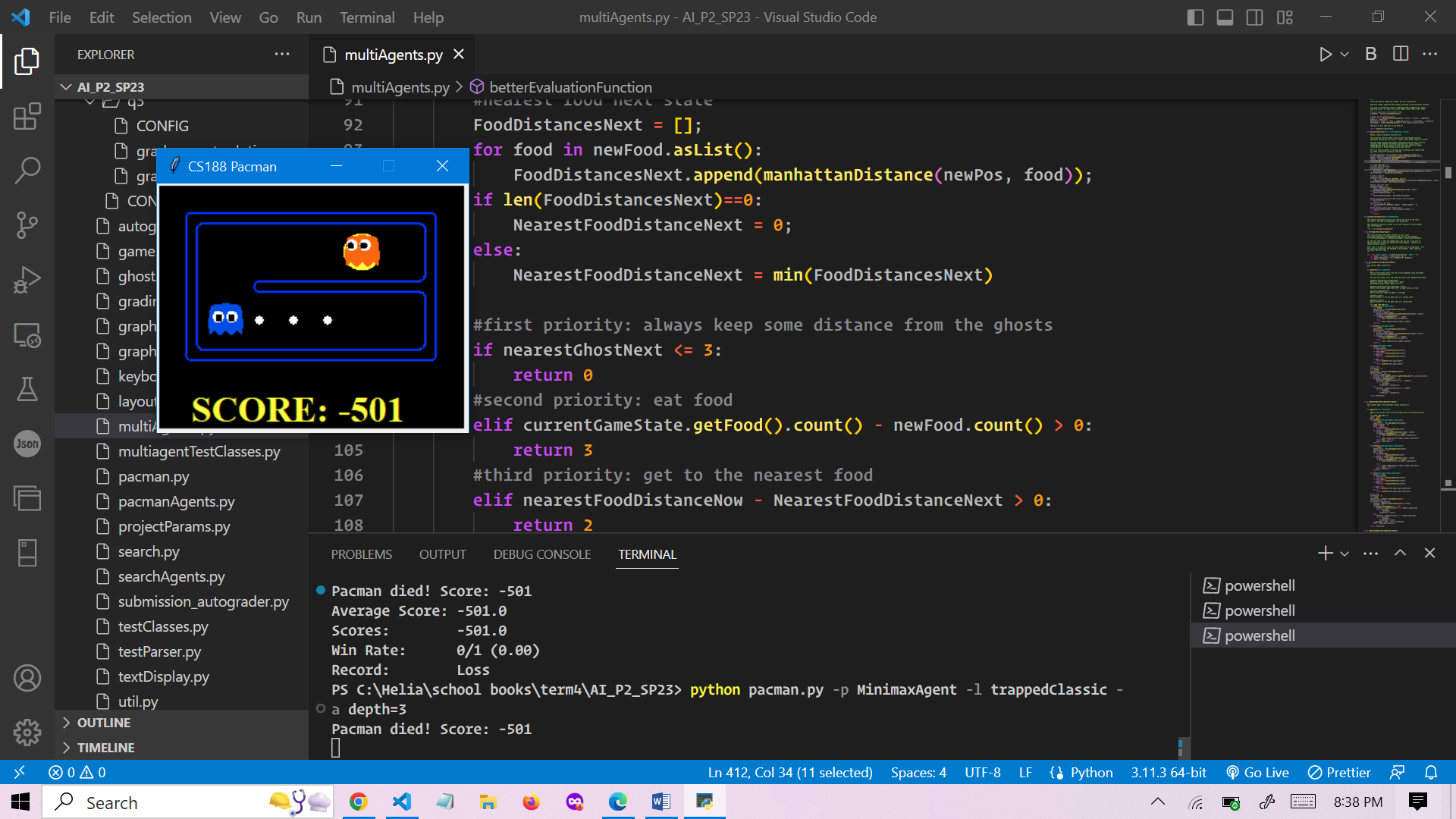
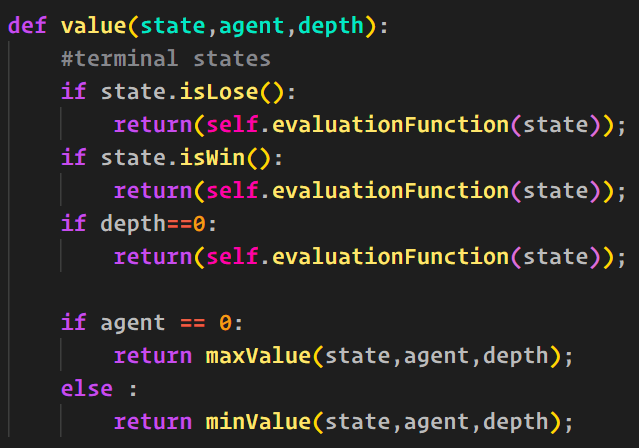


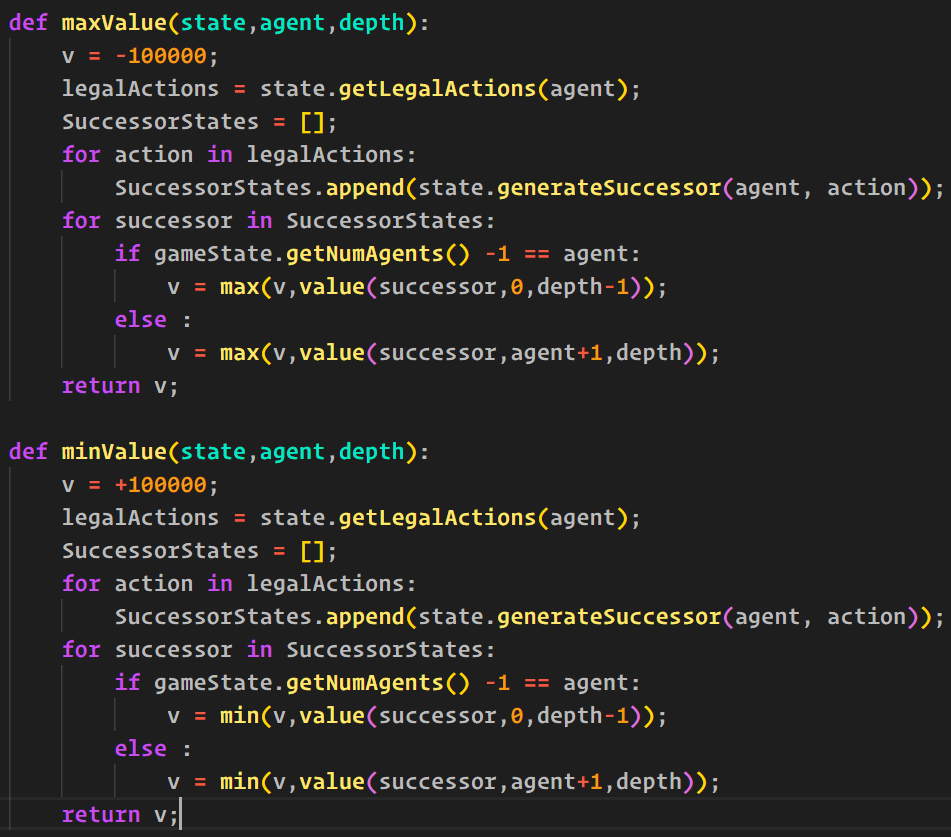
من سه موقعیت در نظر گرفتم، به موقعیت خطرناک این که پکمن در 3 قدمی روح باشد(با استفاده از ghostposition و (manhattan distance، کمترین امتیاز را دادم. سپس به خوردن غذا بیشترین نمره(currentFood.count - newFoodcount)، به نزدیک شدن به نزدیک ترین غذا نیز نمره ای کمتر دادم(currentFoodPosition,NewFoodPosition, manhattan distance).

در اینجا من پارامتر ها را ترکیب نکردم و اولویت بندی ای که انجام دادم به مقدار کافی هوشمند بود که تست ها را قبول شود.

دلیل اولویت بندی من این بود که ما به هیچوجه نمیخواهیم پکمن ببازد، پس باید حتما فاصله از روح ها را حفظ کنیم. همچنین میخواهیم در سرعت قابل قبولی، همه ی غذا ها را بخوریم پس سعی میکنیم به سمت نزدیک ترین غذا همواره در حرکت باشیم و بخوریم آن را.

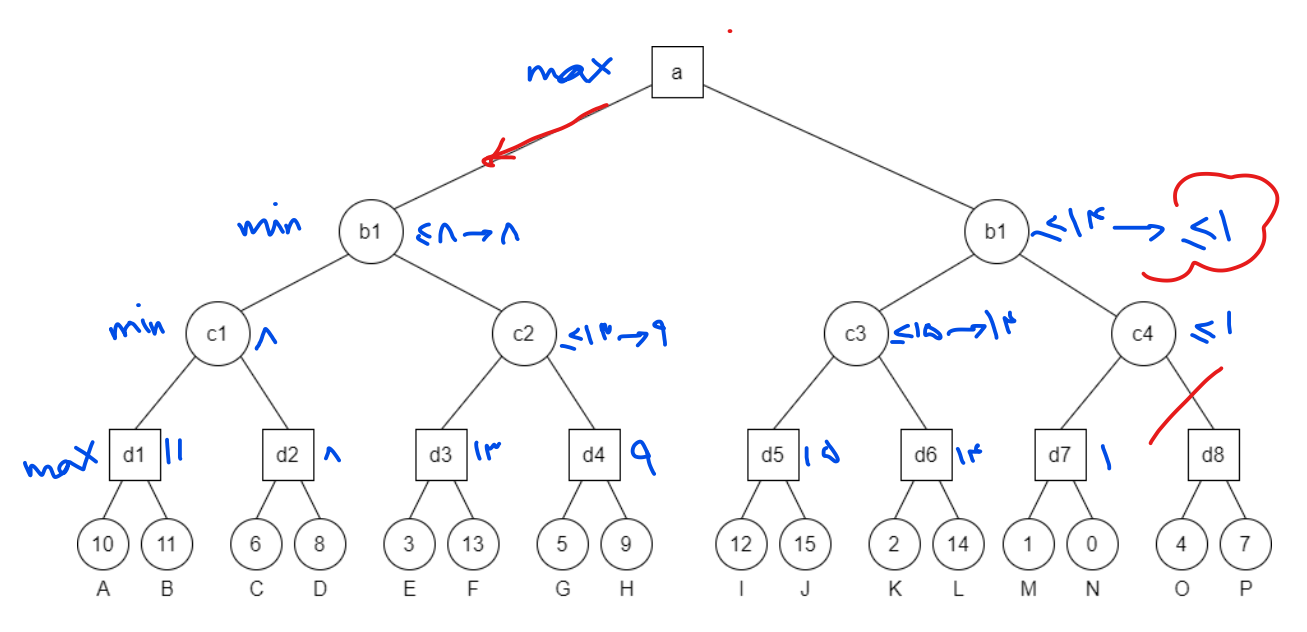
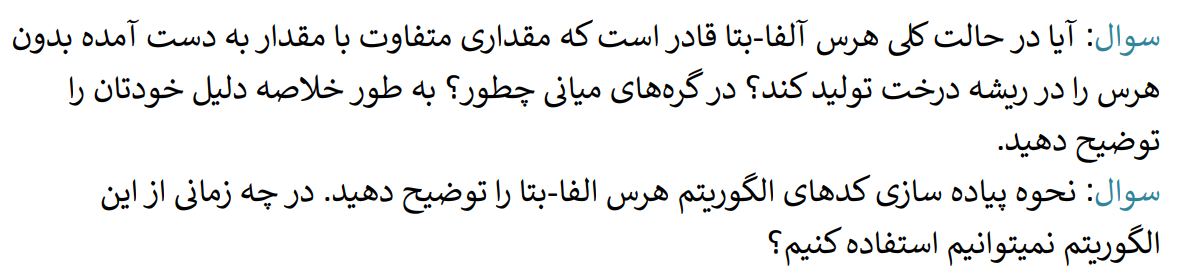
**2) مینیمکس**

****چون پکمن بین 2 روح است و فرض میکند هر دو خصمانه عمل میکنند، پس میداند به او نزدیک تر میشوند و کشته میشود. پس در این حالت، بیشترین امتیازی که میتواند بگیرد با سریع تر مردن است تا هزینه ی زندگی کمتری ازش کم شود.



همانند دیاگرام اسلاید های درس، عمل کرده ام. در هر لایه، فانکشن value با مشاهده عامل بعدی، اگر 0 باشد، پکمن است و maxValue را صدا میزند و برای بقیه عامل ها، minValue را صدا میزند. همچنین اگر به پایان عمق رسیده باشد یا موقعیت در حال بررسی، موقعیت پایانی برد یا باخت باشد، امتیاز آن state را برمیگرداند. این امتیاز به تابع ارزیابی ای که داریم استفاده میکنیم بستگی دارد. تابع maxValue, minValue تقریبا شبیه اند. به عنوان مثال برای مکس این چنین است که ابتدا با ارزشی بینهایت منفی شروع میکنیم. سپس روی همه ی استیت های ممکن بعدی، تابع value را صدا میزنیم و ماکسیمم امتیاز استیت ها را به عنوان جواب برمیگردانیم.

**3)Alpha-beta pruning**



بله، به محض اینکه درخت بداند کدام راه را ترجیح میدهد، به آن سمت میرود و ممکن است به اینکه مقدار اصلی را دست یابد نرسد چون قبل از رسیدن به آن نود، راه مشخص شده است. در گره های میانی نیز همینطور، ممکن است قبل از مشخص شدن دقیق ارزش گره های میانی، محدودیت عددشان مشخص شده باشد و همان برای ما کافی باشد.

کد مینیماکس:

legalActions = gameState.getLegalActions(0);

        SuccessorStates = [];

        for action in legalActions:

          SuccessorStates.append((gameState.generateSuccessor(0,action),action));

        for successor in SuccessorStates:

کد آلفابتا:

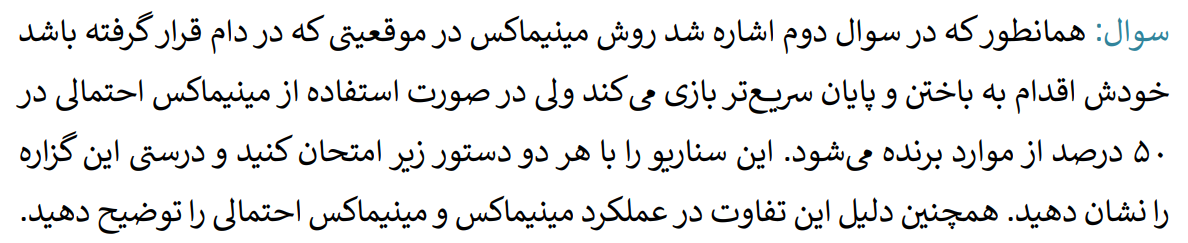
if not legalActions:

            return self.evaluationFunction(state)

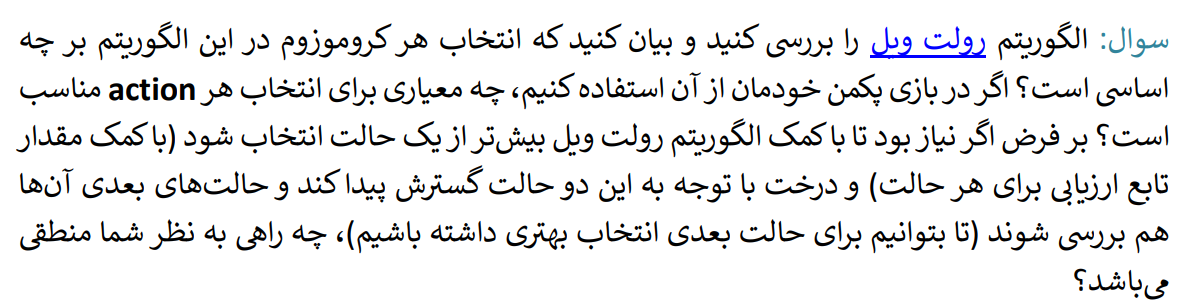
        for action in legalActions:

            successor = state.generateSuccessor(agent, action);

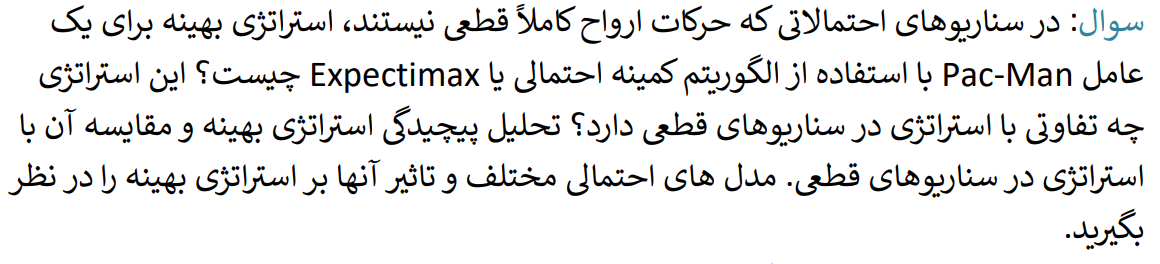
تفاوت این دو کد در این است که حتما باید ابتدا ارزش هر گره باید بررسی شود سپس در حالات خاصی، گسترش داده میشود. در این الگوریتم همچنین مقادیر الفا و بتا نیز به فانکشن ها پاس داده میشوند. آلفا بهترین امتیاز تا این لحظه برای ریشه را برمیگرداند و بتا مینیموم مقدار برای انتخاب گره های مینی را برمیگرداند و با توجه به این دو مقدار، برخی گره ها گسترش داده نمیشوند.

**4)Expectimax**

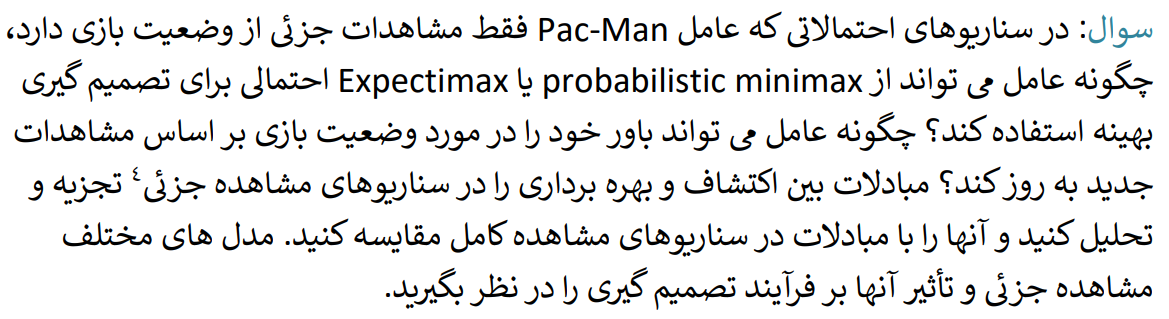
به این دلیل است که تصور پکمن در مینیمکس این است که روح ها خصمانه عمل میکنند پس فکر میکند روح ها به او نزدیک میشوند و او را میکشند پس سعی میکند با زودتر مردن، امتیاز بیشتری کسب کند پس هر بار میمیرد.

اما وقتی فکر میکند حرکات ارواح احتمالی است، پس 50درصد امکان دارد روح پایینی به راست برود و راه را برای اینکه پکمن نقطه بخورد باز بذارد و پکمن برنده شود و 50 درصد امکان دارد به سمت پکمن بیاید و پکمن خورده شود. اگر میانگین نمرات بازی در حالت راست رفتن پکمن را با چپ رفتن پکمن مقایسه کنیم، پکمن ترجیح میدهد به سمت غذاها حرکت کند و اینگونه در مواقعی که روح راست را انتخاب میکند، برنده میشود.

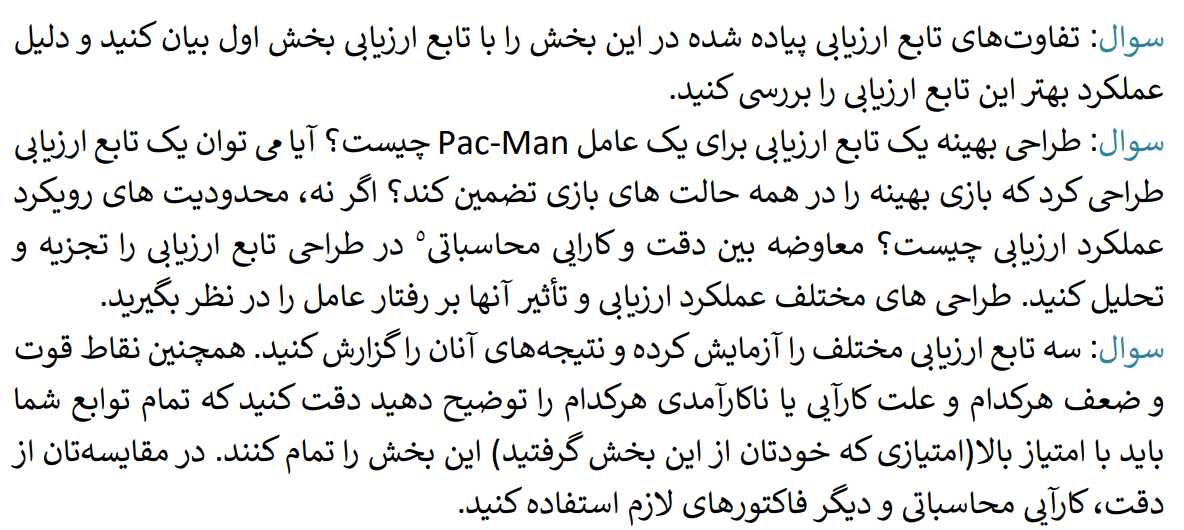
the larger the fitness of an individual is, the more likely is its selection

در پایین ترین بخش، احتمال انتخاب هر موقعیت، باید متناسب با تابع ارزیابی آن باشد و همینطور هر مرحله را اینچنین طی میکنیم تا به انتخاب پکمن برسیم آنگاه ماکسیمم را انتخاب میکنیم.

در استراتژی بهینه expectimax، گره های بیشتری مورد بررسی قرار میگیرند چون نیاز به بدست اوردن میانگین ارزش داریم، اما در minimax چون به دنبال مین یا مکس بودیم، گره های کمتری نیاز به بررسی بود و هرس بیشتر بود.



ابتدا پکمن یک احتمال p برای اینکه روح ها expectimax هستند و 1-p احتمال minimax در نظر بگیرد. پس از هر بار بازی، این احتمال را تغییر دهد و بتواند احتمالی که بهترین نتیجه را داد، انتخاب کند و بر اساس آن بازی کند.

**5)better evaluation function**

در این تابع، اینده(نزدیک نشدن به روح) و هدف بزرگتر(نزدیک تر شدن به غذا) نیز در نظر گرفته میشود درحالی که تابع ارزیابی قبلی صرفا امتیاز بازی بود که حرکت به سمت هدف نهایی یا دوری از ارواح از چند قدم قبل تر را در نظر نمیگرفت.

هر چی سعی کنیم تابع ارزیابی را به دنیای واقعی نزدیک تر کنیم، دقت آن بیشتر اما مدت محاسبه ی آن بیشتر میشود. باید بین عمقی که پیشبینی میکنیم با میزان محاسبات در تابع ارزیابی مان، بالانس برقرار کنیم.