-۱

- از پارامتر مکان غذاها برای پیدا کردن نزدیک‌ترین غذا استفاده می‌کنیم. این عاملی است که هر چقدر مقدار آن بیش‌تر باشد باید امتیاز کم‌تری برای آن بگیریم، در نتیجه در محاسبه‌ی امتیاز یک را بر آن تقسیم می‌کنیم.
همچنین از مجموع فاصله‌ی عامل پک‌من با روح‌ها به عنوان یک فاکتور دیگر استفاده می‌کنیم. این یک عامل منفی است و آن را در مخرج یک قرار داده و در یک ضریب منفی ضرب می‌کنیم. به این‌گونه با افزایش آن مقدار کمتری از امتیاز کم می‌شود. همچنین اگر فاصله در حرکت بعدی تا روح بسیار کم باشد، یک عامل بسیار منفی حساب شده و باید مستقیماً در عدد منفی با مقیاسی بزرگ ضرب شود تا این حرکت انتخاب نشود.
همچنین از زمان در حالت ترس بودن روح‌ها نیز با استفاده از میانگین آن به عنوان یک عامل مثبت مستقیم استفاده می‌کنیم.
در پیاده‌سازی تأثیر زمان در حالت ترس بودن روح‌ها و همچنین نزدیک بودن روح‌ها از عوامل دیگر بیش‌تر در نظر گرفته شده است.

- می‌توان این کار را با ضریب منفی و مثبت دادن به عوامل، و همچنین در صورت لزوم به کم یا زیاد کردن اولویت، می‌توان با قرار دادن آن امتیاز در مخرج ۱ هنگام جمع زدن امتیاز نهایی انجام داد.

- ابتدا با متود getScore امتیاز عامل با استفاده از successor موقعیت حاضر گرفته شده است. حال باید تأثیر عوامل ذکر شده در بالا را بر روی این امتیاز اعمال کرده و سپس آن را به صورت خروجی تحویل بدهیم.
به یک بر روی فاصله تا نزدیک‌ترین غذا ضریب یک داده شده است، زیرا اولویت آن نسبت به عوامل دیگر کمتر است. همچنین مجموع فاصله تا روح‌ها به عنوان یک عامل منفی در نظر گرفته شده است که ضریب بیش‌تری نیز نسبت به فاصله تا نزدیک‌ترین غذا دارد، و همچنین فاصله‌ی بسیار کم با روح‌ها به عنوان یک عامل با ضریب بسیار منفی‌تر در نظر گرفته شده است. همچنین زمان در حالت ترس بودن به عنوان یک عامل مثبت که ضریب مثبت بالایی نیز دارد در نظر گرفته شده است و مستقیماً با امتیاز نهایی جمع شده است. با این کار اولویت‌بندی این عوامل به شکل نسبتاً خوبی رعایت شده و عامل در این نقشه همواره پیروز می‌شود.

-۲

- زیرا در بازی پک‌من گذشتن زمان و حرکت کردن عامل پک‌من خود هزینه‌ای در بردارد، و در صورتی که به عامل به زنده ماندن خود ادامه دهد، در حالیکه می‌داند در نهایت می‌میرد، فقط باعث کم‌تر شدن امتیاز خود می‌شود.

- در ابتدا در تابع minimax چک می‌شود که به حالت نهایی نرسیده باشیم. حالت نهایی می‌تواند شامل سه حالت باشد، پک‌من برده باشد، باخته باشد، یا به عمقی از درخت که می‌خواستیم بررسی را انجام دهیم رسیده باشیم.
سپس ایندکس ایجنتی که در عمق حال حاضر قرار دارد را بررسی کرده، و در صورتی که ایجنتی غیر از پک‌من باشد، یک عدد از عمق کم می‌کند. سپس نیز در صورتی که ایجنت پک‌من باشد تابع maxvalue و در غیر این صورت minvalue صدا زده می‌شود. ای توابع در قسمت اولیه‌ی کد شبیه به یک‌دیگر هستند، به این صورت که لیستی خالی از اکشن‌ها را درست کرده، سپس اکشن‌های مجاز را در وضعیت فعلی به دست آورده، روی آن‌ها پیمایش کرده، و برای هر یک از آن‌ها تابع minimax را بر روی successor آن وضعیت صدا می‌زند. همچنین به دلیل یک لایه پایین رفتن، هنگام فراخوانی این تابع یک عدد نیز به ایندکس ایجنت اضافه می‌شود. سپس در تابع maxvalue گره با بیش‌ترین مقدار و برای minvalue گره با کم‌ترین مقدار برگردانده می‌شود.

-۳

- زیردرخت b1 به طول کامل بررسی می‌شود و مقدار ۸ به b1 نگاشت می‌شود. حال می‌دانیم که مقدار b2 برابر با مینیمم مقادیر موجود در گره‌های d5, d6, d7 و d8 خواهد بود. پس اگر در حین بررسی این زیردرخت، مقدار

یکی از این چهار گره برابر با کمتر از ۸ شد، لازم نیست به بررسی ادامه دهیم. این اتفاق در گره‌ی d7 می‌افتد، زمانی که مقدار آن برابر با ۱ می‌شود. در نتیجه پس از آن نیازی به بررسی زیردرخت d8 نیست و مقدار گره‌ی ریشه برابر با ۸ می‌شود، و پکمن نیز باید به سمت چپ حرکت کند.

- ممکن نیست که هرس آلفا-بتا مقداری متفاوت با مقدار بدون هرس را در ریشه‌ی درخت ایجاد کند، زیرا این هرس در واقع فقط از بررسی گره‌هایی که تأثیری در جواب نهایی نخواهند داشت امتناع می‌کند. ولی ممکن است که در گره‌های میانی مقادیری متفاوت با بررسی کل درخت داشته باشیم، زیرا در هنگام هرس کردن ممکن است در جایی به مقادیری برسیم که موجب هرس قسمتی از درخت بشود، و در این صورت مقدار گره‌های میانی دیگر دست‌نخورده بماند، در حالیکه در صورت بررسی کل درخت ممکن بود مقدار آن تغییر کند.

- پیاده‌سازی تا حد زیادی شبیه به پیاده‌سازی minimax در سؤال قبل است، با این تفاوت که هنگام بررسی عامل‌های غیرپک‌من مقدار بتا و هنگام بررسی عامل پک‌من مقدار آلفا در نهایت حساب می‌شود و هر گاه در حین بررسی درخت مقدار بتا کم‌تر از آلفا شود، لوپ بررسی و اضافه کردن اکشن‌ها تمام می‌شود و از ادامه‌ی بررسی آن زیردرخت اجتناب می‌شود.

-۴

- در مینیماکس معمولی، ما تصور می‌کنیم که ایجنت‌های دیگر همواره به صورت خصمانه عمل کرده و بهترین عمل برای خودشان و بدترین عمل برای ما را انتخاب می‌کنند، در نتیجه هنگامی که عامل ما در دام قرار می‌گیرد، تصور می‌کند که قطعاً روح‌ها در حرکات بعدی آن را خواهند خورد، در نتیجه خودش اقدام به سریع‌تر باختن می‌کند تا امتیازهای کم‌تری از دست بدهد. ولی در مینیماکس احتمالی، عامل ما پنجاه درصد احتمال می‌دهد که روح آبی به جای این که به سمت راست حرکت کند و عامل را به دام بیندازد، حرکت غیربهینه‌ای کرده و به سمت پایین می‌رود. در این صورت امتیاز ما بسیار بیش‌تر خواهد شد، در نتیجه عامل این شاخه را انتخاب می‌کند، و در نتیجه با احتمال ۵۰ درصد نیز حرکتش درست می‌شود و برنده می‌شود.

- در این الگوریتم ما احتمال هر حالت را بر حسب fitness بودن آن تعیین می‌کنیم. سپس با انتخاب یک متغیر تصادفی، یکی از حالات انتخاب می‌شود. با انجام این کار در ادامه می‌توان با ترکیب کروموزوم‌ها به کروموزوم‌های جدیدی رسید. در بازی پک‌من نیز مانند این الگوریتم، می‌توان actionها را به دو حالت یک و صفر تقسیم کرد و با آن‌ها کروموزوم ساخت و با استفاده از این الگوریتم، به پاسخ رسید.

- در هنگامی که حرکات ارواح کاملاً قطعی نیستند، باید از این الگوریتم استفاده کنیم، ولی در این صورت دیگر همه‌ی نودها دارای وزن‌های یکسان نخواهند بود، و باید از آن‌ها در هر مرحله‌ای میانگین وزن‌دار بگیریم. تفاوت این استراتژی با استراتژی قبلی در یکنواخت یا غیریکنواخت بودن وزن گره‌ها هنگام میانگین‌گیری است. همچنین مشخصاً محاسبات ما در این استراتژی به دلیل بیش‌تر بودن عملیات‌ها در هر مرحله بیش‌تری خواهد بود.

- در هنگامی که عامل پک‌من دارای مشاهده‌ی جزئی از وضعیت بازی است، باید هنگام استفاده از expectimax، با استفاده از فاکتورهایی که در دسترس دارد و می‌دانیم که در ارزیابی وضعیت‌های بازی دارای نقش مهمی هستند، آن وضعیت را ارزیابی کرده و سپس بین حالات مختلف بهترین راه را انتخاب کند. همچنین به مرور و با بازی کردن می‌تواند بهتر متوجه نقش عوامل مختلف در وضعیت‌های مختلف بازی شود و باور خود را در مورد وضعیت بازی به روز کند.

-۵

- پیاده‌سازی تابع ارزیابی در این بخش بسیار شبیه به تابع ارزیابی بخش اول است، با این تفاوت به جای گرفتن successorهای استیت فعلی و ارزیابی آن حالات، به ارزیابی وضعیتی که در آن هستیم پرداخته‌ایم. این مورد

سبب می‌شود که با بررسی دقیق وضعیت کنونی، گام‌های بعد را نیز بررسی کنیم، در حالیکه در تابع اول ما فقط می‌توانیم یک گام بعد را بررسی کنیم.

- نمی‌توان تابع ارزیابی طراحی کرد که مطمئن بود در همه‌ی حالات بازی، بازی بهینه را تضمین می‌کند، زیرا ممکن است که مسائل بسیار پیچیده باشند و یا متغیرها و وابستگی‌های بسیار زیادی داشته باشند که کار مدل کردن آن‌ها را بسیار پیچیده سازد. همچنین ممکن است مسأله دارای edge caseهای مختلفی باشد که با استفاده از تابع ما قابل تشخیص نباشد و سبب انتخاب‌های اشتباه شود. طبیعتاً می‌توان دقت تابع ارزیابی را تا حدی با پیچیده کردن محاسبات بیش‌تر کرد، تا در حالات مختلف برآورد درست‌تری از آن وضعیت بدهند، ولی این مورد در مقابل کارایی و سرعت قرار می‌گیرد و از سادگی تابع می‌کاهد.

- تابع ارزیابی که به صورت دیفالت از آن استفاده می‌کردیم، امتیاز وضعیت فعلی را می‌گرفت و آن را به صورت خروجی تحویل می‌داد که به طور مشخصی چندان تابع دقیقی نیست و می‌توان آن را بهبود داد. تابع ارزیابی که در قسمت اول نوشتیم ارزیابی را بر اساس حرکاتی که از وضعیت فعلی می‌توانیم انجام دهیم انجام می‌داد و برای هر یک علاوه بر امتیازی که در آن وضعیت داشتیم، مواردی دیگر مانند نزدیکی به غذاها، روح‌ها و عوامل دیگر را نیز در نظر می‌گرفت. و در نهایت آخرین تابعی که نوشتیم، این عوامل را بر روی وضعیت فعلی که بودیم می‌سنجید و تأثیر آن‌ها را بر امتیاز فعلی که در وضعیت فعلی داریم اعمال می‌کرد. در نتیجه به صورت خلاصه، تابع دیفالت ما از کارآیی محاسباتی بهتری برخوردار است، ولی دقتش نسبت به دو تابع دیگر کم‌تر می‌شود. در مقابل دو تابع دیگر دارای محاسبات بیش‌تری در هر مرحله هستند، ولی در عوض دقت بیش‌تری دارند و حرکات بهینه‌تری را خروجی می‌دهند.