**گزارش پروژه دوم**

**اشکان شکیبا (9931030)**

**بخش اول**

توضیح کد:

score = 0.0  
score += successorGameState.getScore()  
score += 2 \* (1 / food\_score)  
score += -2 \* (1 / total\_ghost\_distances)  
score += 5 \* ghost\_distance\_score  
score += scared\_time\_score

در متد evaluationFunction، به شکلی برای هر از یک از ویژگی‌های مورد نظر از جمله گرفتن غذاها و دوری از روح‌ها، امتیازی تعریف و محاسبه شده و در نهایت این امتیازات با ضرایبی که بسته به اهمیت آن مورد متفاوت هستند، با هم جمع زده شده و به عنوان حاصل متد بازگردانی می‌شوند.

سوال ۱)

برای هر کدام امتیازی محاسبه می‌شود که می‌تواند مثبت یا منفی باشد.

امتیاز اولیه ضریب ۱، معکوس فاصله نزدیک‌ترین غذا ضریب ۲، مجموعه فاصله روح‌ها ضریب ۲- و تعداد روح‌هایی که بسیار نزدیک هستند ضریب ۵ دارد.

سوال ۲)

می‌توان به پارامتری که افزایش آن ما را از هدف دور می‌کند، ضریب منفی داد.

**بخش دوم**

توضیح کد:

def mini\_max(state, iter\_count):  
 if state.isWin() or state.isLose() or iter\_count >= self.depth \* agents\_count:  
 return self.evaluationFunction(state)  
 iter\_mod\_agents = iter\_count % agents\_count  
 if iter\_mod\_agents != 0:  
 result = sys.maxsize  
 for action in state.getLegalActions(iter\_mod\_agents):  
 if action != "Stop":  
 new\_state = state.generateSuccessor(iter\_mod\_agents, action)  
 mini\_max\_result = mini\_max(new\_state, iter\_count + 1)  
 if mini\_max\_result < result:  
 result = mini\_max\_result  
 return result  
 else:  
 result = -sys.maxsize  
 for action in state.getLegalActions(iter\_mod\_agents):  
 if action != "Stop":  
 new\_state = state.generateSuccessor(iter\_mod\_agents, action)  
 mini\_max\_result = mini\_max(new\_state, iter\_count + 1)  
 if mini\_max\_result > result:  
 result = mini\_max\_result  
 if iter\_count == 0:  
 action\_score.append(result)  
 return result

ابتدا تابع mini\_max بدین شکل پیاده‌سازی شده و سپس با یک بار فراخوانی آن که خود را به شکل بازگشتی فراخوانی می‌کند، امتیاز actionها را محاسبه و action مناسب‌تر را بازگردانی می‌کنیم.

در پیاده‌سازی این تابع ابتدا وضعیت کلی بازی بررسی می‌شود که آیا به پایان رسیده یا به عمق مورد نظر رسیده‌ایم یا نه. در صورتی که هنوز به این حالت نرسیده باشیم، ابتدا با بررسی باقی‌مانده تقسیم iter\_count به تعداد عامل‌ها بین پک‌من و روح‌ها تمایز قائل می‌شویم. برای پک‌من ماکسیمم و برای روح‌ها مینیمم بررسی و محاسبه می‌شود و در صورتی که هنوز به پایان نرسیده باشد، با فراخوانی مجدد تابع با عمق بیشتر، فرآیند ادامه می‌یابد.

سوال)

با توجه به اینکه در یک درخت مینیماکس بدترین حالت بررسی می‌شود، با اطمینان یافتن از نتیجه آن بهتر است هر چه زودتر بازی به پایان برسد تا امتیازی که به خاطر زمان از دست می‌دهیم کمینه شود.

**بخش سوم**

توضیح کد:

def alpha\_beta(state, iter\_count, alpha, beta):  
 if state.isWin() or state.isLose() or iter\_count >= self.depth \* agents\_count:  
 return self.evaluationFunction(state)  
 iter\_mod\_agents = iter\_count % agents\_count  
 if iter\_mod\_agents != 0:  
 result = sys.maxsize  
 for action in state.getLegalActions(iter\_mod\_agents):  
 if action != "Stop":  
 new\_state = state.generateSuccessor(iter\_mod\_agents, action)  
 alpha\_beta\_result = alpha\_beta(new\_state, iter\_count + 1, alpha, beta)  
 if alpha\_beta\_result < result:  
 result = alpha\_beta\_result  
 if result < beta:  
 beta = result  
 if beta < alpha:  
 break  
 return result  
 else:  
 result = -sys.maxsize  
 for action in state.getLegalActions(iter\_mod\_agents):  
 if action != "Stop":  
 new\_state = state.generateSuccessor(iter\_mod\_agents, action)  
 alpha\_beta\_result = alpha\_beta(new\_state, iter\_count + 1, alpha, beta)  
 if alpha\_beta\_result > result:  
 result = alpha\_beta\_result  
 if result > alpha:  
 alpha = result  
 if iter\_count == 0:  
 action\_score.append(result)  
 if beta < alpha:  
 break  
 return result

ابتدا تابع alpha\_beta بدین شکل پیاده‌سازی شده و سپس با یک بار فراخوانی آن که خود را به شکل بازگشتی فراخوانی می‌کند، امتیاز actionها را محاسبه و action مناسب‌تر را بازگردانی می‌کنیم.

پیاده‌سازی این تابع تا حد زیادی مشابه تابع mini\_max در بخش دوم است، با این تفاوت که در هنگام بررسی روح‌ها مقدار بتا در نهایت محاسبه می‌شود و در هنگام بررسی پک‌من مقدار آلفا در نهایت محاسبه می‌شود و در هر دوی آنها اگر مقدار بتا از آلفا کمتر شود، حلقه بررسی actionها به پایان می‌رسد و بدین شکل هرس آلفا-بتا صورت می‌پذیرد.

سوال ۱)

مقادیر اعضای درخت پس از اجرای الگوریتم:

a = 8 (alpha = 8, beta = +∞)

b1 = 8 (alpha -∞, beta = 8)

b2 = 1 (alpha = 8, beta = 1)

c1 = 8 (alpha = -∞, beta = 8)

c2 = 9 (alpha = -∞, beta = 8)

c3 = 14 (alpha = 8, beta = 14)

c4 = 1 (alpha = 8, beta = 1)

d1 = 11 (alpha = 11, beta = +∞)

d2 = 8 (alpha = 8, beta = 11)

d3 = 13 (alpha = 13, beta = 8)

d4 = 9 (alpha = 9, beta = 8)

d5 = 15 (alpha = 15, beta = +∞)

d6 = 14 (alpha = 14, beta = 15)

d7 = 1 (alpha = 1, beta = 14)

d8 = 4 (alpha = 4, beta = 1)

فقط گره P هرس می‌شود، به این دلیل که ماکسیمایزر d8 با مقایسه ۴ و beta که برابر ۱ است، شاخه بعدی را هرس می‌کند.

حرکت بعدی پک‌من نیز به سمت چپ است.

سوال ۲)

در ریشه مقداری متفاوت تولید نمی‌شود، اما در گره‌های میانی ممکن است؛ دلیل این موضوع هم این است که هرس کردن بدون توجه به شاخه‌های بعدی انجام می‌شود و ممکن بود در شاخه‌های بعدی به مقداری بزرگتر (برای ماکسیمایزر) یا کوچک‌تر (برای مینیمایزر) دست یابیم.