پیاده‌سازی 1)

```
"*** YOUR CODE HERE ***"
res = 1
currentPos = currentGameState.getPacmanPosition()

disToGhosts = []
for gs in newGhostStates:
    manDisGhost = manhattanDistance(gs.getPosition(), newPos)
    disToGhosts.append(manDisGhost)
minDisGhost = min(disToGhosts)
if minDisGhost < 2: return 0

disToFoods = []
for f in currentGameState.getFood().asList():
    manDisFood = manhattanDistance(f, currentPos)
    disToFoods.append(manDisFood)
minDisFood = min(disToFoods)

newDisToFoods = []
for f in newFood.asList():
    manDisFood = manhattanDistance(f, newPos)
    newDisToFoods.append(manDisFood)
if len(newDisToFoods) == 0: minNewDisToFoods = 0
else: minNewDisToFoods = min(newDisToFoods)

if successorGameState.getScore() > currentGameState.getScore(): res = 1000
elif minNewDisToFoods < minDisFood: res = 100
elif sum(newScaredTimes) > 0: res = 10
return res
```

در قسمت اول کمترین فاصله ممکن مختصات جدید پکمن با یک روح محاسبه می‌شود. که اگر آن فاصله کمتر از 2 بود، تابع ما مقدار 0 را برمی‌گرداند که این یعنی این موقعیت جدید به هیچ عنوان مطلوب ما نیست (هرچه مقدار خروجی تابع ارزیابی بیشتر باشد یعنی آن حرکت بهتر است). در قسمت دوم کمترین فاصله موجود یک غذا با مختصات فعلی پکمن حساب می‌شود. در قسمت سوم همین فاصله برای موقعیت جدید پکمن محاسبه می‌شود (اگر غذایی در صفحه نباشد، این متغیر مقدار 0 می‌گیرد). در آخر در سه اولویت، اگر حرکت باعث شد امتیاز ما بیشتر شود، مقدار 1000، در غیر این صورت اگر حرکت باعث شد به نزدیک‌ترین غذا نزدیک‌تر شویم، مقدار 100، و در غیر این صورت اگر حرکت باعث شود یک boost بخوریم، مقدار 10 را برمی‌گردانیم (اگر هیچ کدام از اینها برآورده نشد، مقدار دیفالت 1 برای res برگردانده می‌شود).

سوال 1) چگونگی استفاده از پارامترها و تاثیر آنها در خروجی بالا توضیح داده شده است و طبق اولویت های بالا، تاثیر score از بقیه بیشتر است و همینطور به سمت پایین تاثیرها کمتر می‌شود.

سوال 1) یک راه دادن اولویت مانند پیاده‌سازی من و برسی آنها به ترتیب اولویت است. یک راه دیگر در صورتی که مثلا هر پارامتر در یک عددی جمع می‌شود و خروجی نهایی را ایجاد می‌کنند می‌تواند این باشد که آن پارامتری که هرچه کمتر باشد بهتر است را معکوس کنیم و سپس جمع کنیم (مثل فاصله تا غذا).

پیاده‌سازی 2)

```
"*** YOUR CODE HERE ***"
def minimizer(agentIndex, state, depth):
    v = float('inf')
    if state.isLose() or state.isWin(): return self.evaluationFunction(state)
    for a in state.getLegalActions(agentIndex):
        successorState = state.generateSuccessor(agentIndex, a)
        if state.getNumAgents()-1 == agentIndex: v = min(v, maximizer(successorState, depth))
        else: v = min(v, minimizer(agentIndex + 1, successorState, depth))
    return v

def maximizer(state, depth):
    v = float('-inf')
    if depth+1 == self.depth or state.isLose() or state.isWin(): return self.evaluationFunction(state)
    for a in state.getLegalActions(0):
        successorState = state.generateSuccessor(0, a)
        v = max(v, minimizer(1, successorState, depth+1))
    return v

currentScore = float('-inf')
for a in gameState.getLegalActions(0):
    newGameState = gameState.generateSuccessor(0, a)
    bestScore = minimizer(1, newGameState, 0)
    if bestScore > currentScore:
        res = a
        currentScore = bestScore
return res
```

مینیمایزر (روح) با توجه به اندیس‌اش و حالت و عمق، کار می‌کند. مقدار آن ابتدا بینهایت است (بدترین مقدار ممکن برای آن). اگر به حالت باخت یا برد برسیم مقدار تابع ارزیابی را برمی‌گرداند (که در اینجا همان مقدار score است). سپس در یک حلقه روی تمام action های ممکن آن روح، کمترین مقدار value را با توجه به عمل انجام شده بین آن روح و روح های بعدی‌اش به صورت بازگشتی، پیدا می‌کنیم و آن همان value مینیمایزر ما می‌شود (در صورتی که روح ما آخرین روح باشد، مینیموم value و مکسیمایزر (پکمن) برگردانده می‌شود). کد مکسیمایزر نیز مشابه است با این تفاوت که اندیس آن همیشه 0 است که همان پکمن ماست (فقط یک پکمن در نقشه داریم). v اولیه آن نیز در ابتدا بدترین مقدار ممکن یعنی منفی بینهایت است. و علاوه بر رسیدن به حالت برد یا باخت، رسیدن به عمق جستجو نیز باعث می‌شود که آن، مقدار تابع ارزیابی را برگرداند. در یک حلقه روی عمل‌های ممکن و به صورت بازگشتی، مقدار ماکسیمم v را با مقایسه با مقدار مینیمایزر 1 و با حرکت به سمت عمق پایین پیدا می‌کنیم. در بدنه اصلی کد نیز، در یک حلقه با توجه به عمل‌های ممکن پکمن، مینیمایزر را برای اولین روح صدا می‌زنیم تا score را در آن حالت مشخص استخراج کنیم و از بین آن score های استخراج شده، آن عملی که مارا به بیشترین score رساند خروجی و جواب ما می‌شود.

سوال 2) پکمن همواره برسی می‌کند که از بین عمل‌های ممکن کدام یک باعث می‌شود که در نهایت بیشترین امتیاز را داشته باشد درنتیجه هنگامی که بین چند روح گیر می‌کند، با اینکه عمل‌های ممکن در نهایت امتیاز به نسبت پایینی را به آن می‌دهند، ولی پکمن طبق پیاده‌سازی‌اش عملی را که به آن بیشترین امتیاز موجود از بین این امتیازهای ممکن را می‌دهد را انتخاب و طی می‌کند (وقتی isLose برقرار باشد، مقدار score از گره‌ها برمی‌گردد). در موقعیتی که مردن اجتناب‌ناپذیر است، هرچه پکمن بیشتر فرار کند، با گذر زمان امتیازش کمتر و کمتر می‌شود بنابراین بیشترین امتیاز در زودهنگام‌ترین مرگ است.

پیاده‌سازی 3)

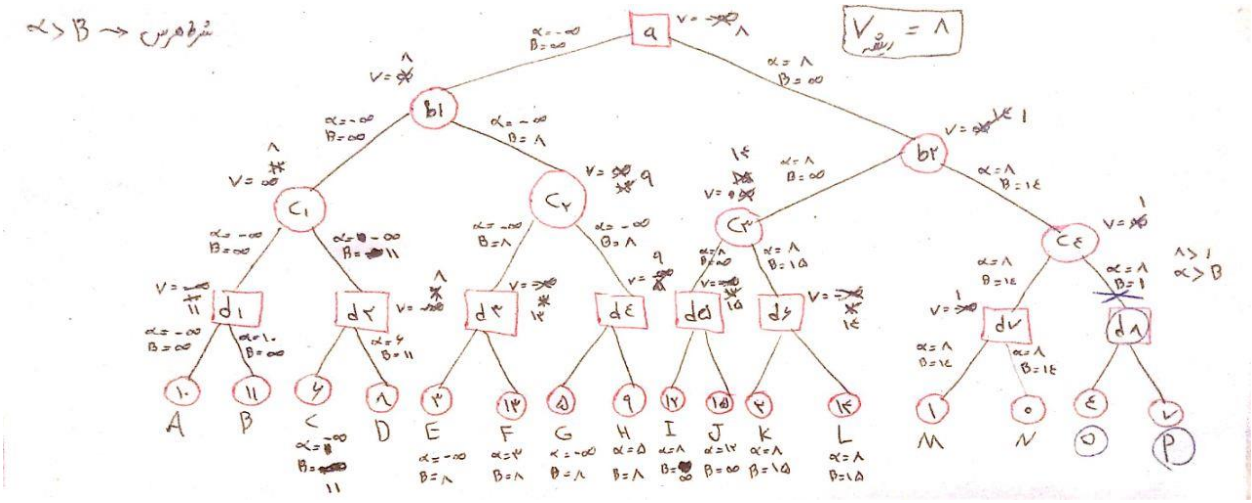
```
"*** YOUR CODE HERE ***"
def minimizer(agentIndex, state, depth, alpha, beta):
    v = float('inf')
    if state.isLose() or state.isWin(): return self.evaluationFunction(state)
    betaHolder = beta
    for a in state.getLegalActions(agentIndex):
        successorState = state.generateSuccessor(agentIndex, a)
        if state.getNumAgents()-1 == agentIndex:
            v = min(v, maximizer(successorState, depth, alpha, betaHolder))
            if v < alpha: return v
            betaHolder = min(betaHolder, v)
        else:
            v = min(v, minimizer(agentIndex + 1, successorState, depth, alpha, betaHolder))
            if v < alpha: return v
            betaHolder = min(betaHolder, v)
    return v

def maximizer(state, depth, alpha, beta):
    v = float('-inf')
    if depth+1 == self.depth or state.isLose() or state.isWin(): return self.evaluationFunction(state)
    alphaHolder = alpha
    for a in state.getLegalActions(0):
        successorState = state.generateSuccessor(0, a)
        v = max(v, minimizer(1, successorState, depth+1, alphaHolder, beta))
        if v > beta: return v
        alphaHolder = max(alphaHolder, v)
    return v

alpha = float('-inf')
beta = float('inf')
currentScore = float('-inf')
for a in gameState.getLegalActions(0):
    newGameState = gameState.generateSuccessor(0, a)
    bestScore = minimizer(1, newGameState, 0, alpha, beta)
    if bestScore > currentScore:
        res = a
        currentScore = bestScore
    if bestScore > beta: return res
    alpha = max(alpha, bestScore)
return res
```

کد این قسمت عمداتا مانند قسمت قبل است و فقط به تفاوت‌ها می‌پردازم. به هدر مکسیمایزر و مینیمایزر دو متغیر آلفا و بتا اضافه شده است. در کد مینیمایزر پس از call بازگشتی، شرط هرس را چک می‌کنیم و در صورت برقرار بودن آن، return می‌کنیم و ادامه نمی‌دهیم و سپس مقدار بتا را آپدیت می‌کنیم. در کد مکسیمایزر نیز به همین صورت اگر شرط هرس برقرار بود، پیشروی نمی‌کنیم و در ادامه مقدار آلفا را آپدیت می‌کنیم. در بدنه اصلی الگوریتم، به آلفا و بتا بدترین مقادیر ممکن را مقداردهی اولیه می‌کنیم. تنها تفاوت در اینجا نیز همان چک کردن شرط هرس و آپدیت کردن آلفا است. در صورتی که شرط هرس در اینجا برقرار باشد تابع ما خروجی (بهترین امتیاز) را برمی‌گرداند و در واقع هرس صورت می‌گیرد.

سوال 3)



گره‌های d8 و O و P به دلیل برقرار شدن شرط هرس، هرس شدند. حرکت پکمن نیز باید به سمت چپ باشد زیرا مقدار بهینه بدست آمده برای ریشه (v = 8) با عمل سمت چپ رفتن میسر شد.

سوال 3) خیر. مقدار ریشه با استفاده از این هرس باید برابر با مقدار ریشه بدون هرس باشد زیرا در غیر این صورت یعنی هرس ما باعث شده است مقدار بهینه را در ریشه نداشته باشیم (بدون هرس همواره مقدار بهینه را در ریشه داریم). هرس تنها باعث می‌شود که رسیدن ما به جواب بهینه سریع‌تر شود و جواب نباید تغییر کند. با این حال مقدار گره‌های میانی با استفاده از هرس ممکن است تغییر کند. زیرا اساسا مفهوم هرس این است که چه ما هرس نکنیم و ادامه دهیم و چه هرس بکنیم (که قاعداتا ممکن است باعث شود مقادیر یک سری از گره‌ها این وسط تغییر کند)، تفاوتی در مقدار نهایی آن گره بالایی که با آن مقایسه انجام شد، رخ نمی‌دهد. مثلا در شکل بالا اگر دو گره O و P مقدار 100- داشتند، گره‌های c4 و b2 در صورت هرس کردن یا نکردن مطابق شکل، مقادیر متفاوتی اختیار می‌کردند ولی مقدار ریشه باز همان 8 می‌شد.

پیاده‌سازی 4)

```
"*** YOUR CODE HERE ***"
def expecti(agentIndex, state, depth):
    sumV = 0
    if state.isLose() or state.isWin(): return self.evaluationFunction(state)
    for a in state.getLegalActions(agentIndex):
        successorState = state.generateSuccessor(agentIndex, a)
        if state.getNumAgents()-1 == agentIndex: v = maximizer(successorState, depth)
        else: v = expecti(agentIndex + 1, successorState, depth)
        sumV += v
    if not len(state.getLegalActions(agentIndex)): return  0
    res = sumV / len(state.getLegalActions(agentIndex))
    return res

def maximizer(state, depth):
    v = float('-inf')
    if depth+1 == self.depth or state.isLose() or state.isWin(): return self.evaluationFunction(state)
    for a in state.getLegalActions(0):
        successorState = state.generateSuccessor(0, a)
        v = max(v, expecti(1, successorState, depth+1))
    return v

currentScore = float('-inf')
for a in gameState.getLegalActions(0):
    newGameState = gameState.generateSuccessor(0, a)
    bestScore = expecti(1, newGameState, 0)
    if bestScore > currentScore:
        res = a
        currentScore = bestScore
return res
```

کد بسیار مشابه کد قسمت (2) می‌باشد ولی به جای تابع مینیمایزر، تابع expecti داریم که مشابه آن است با این تفاوت که مقادیر v برای عمل‌های ممکن راء در متغیر sumV جمع می‌کنیم و در آخر بر تعداد عمل‌های ممکن تقسیم می‌کنیم (به شرط غیر صفر بودن مخرج) تا خروجی نهایی را که میانگین هزینه گره‌های فرزند است را داشته باشیم (همانطور که سوال گفته توزیع احتمال بین فرزندان مساوی است. بنابراین خروجی هر گره expecti میانگین فرزندانش می‌شود). تنها فرق مکسیمایزر و بدنه اصلی تابع با کد قسمت (2) نیز در این است که به جای مینیمایزر قاعداتا expecti را صدا می‌زنند.

سوال 4)

```
C:\Users\Amir47\Desktop\AI-P2>python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

C:\Users\Amir47\Desktop\AI-P2>python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Average Score: 221.8
Scores:        532.0, -502.0, 532.0, -502.0, 532.0, -502.0, 532.0, 532.0, 532.0, 532.0
Win Rate:      7/10 (0.70)
Record:        Win, Loss, Win, Loss, Win, Loss, Win, Win, Win, Win

C:\Users\Amir47\Desktop\AI-P2>
```

در این نقشه در 50 درصد مواقع روح آبی در جهت مخالف پکمن حرکت می‌کند که این باعث می‌شود حالتی به وجود بیاید که پکمن قبل اینکه توسط روح‌ها نابود شود، بتواند تمام غذاها را بخورد و برنده شود. همان طور که در قسمت (2) توضیح داده شد با استفاده از مینی‌مکس پکمن خودش را با سرعت نابود می‌کند چون فقط امتیاز را معیار قرار می‌دهد ولی با مینی‌ماکس احتمالی، پکمن با راست رفتن حتما نابود خواهد شد و با چپ رفتن، 50 درصد شانس برنده شدن دارد (امتیاز زیادی در برنده شدن است) بنابراین با توجه به معماری مینی‌ماکس احتمالی و این شانس، زیردرخت سمت راست پکمن مقداری کمتر از زیردرخت سمت چپ برمی‌گرداند و در نتیجه همیشه مسیر سمت چپ توسط عامل انتخاب می‌شود و پکمن همواره به سمت چپ می‌رود (و در 50 درصد مواقع که روح آبی هم در جهت مخالف آن حرکت می‌کند، برنده می‌شود).

سوال 4) در الگوریتم رولت ویل هر کدام از کروموزوم‌ها بر اساس میزان مناسب بودنشان (تابع برازش) یک زاویه متناسب روی ویل خواهند داشت و درنتیجه به میزان مناسب بودنشان شانس برای انتخاب شدن دارند. در بازی ما هم با توجه به مقداری که action های مختلف در درخت برمی‌گردانند و یا تابع ارزیابی حالتی که این action ها ما را به آن می‌برند، می‌توان به آنها زاویه و شانس داد. اگر در هر انتخاب بخواهیم بیش از یک حالت داشته باشیم، به هر کدام ازین گره‌ها که شامل چندین حالت اند می‌توان میانگین تابع ارزیابی حالت هایش را نسبت داد و با توجه به این معیار به آنها شانس و زاویه انتخاب شدن در رولت ویل داد.

پیاده‌سازی 5)

```
"*** YOUR CODE HERE ***"
res = 0

disToFoods = []
for f in foods.asList():
    manDisFood = manhattanDistance(f, pacmanPosition)
    disToFoods.append(manDisFood)
if not disToFoods: minDisFood = 1
else: minDisFood = min(disToFoods)
res += 1 / float(minDisFood)

disToGhosts = []
for g in ghostPositions:
    manDisGhost = manhattanDistance(g, pacmanPosition)
    disToGhosts.append(manDisGhost)
if not disToGhosts: minDisGhost = 0
else: minDisGhost = min(disToGhosts)
res += float(minDisGhost) / 10

res += sum(scaredTimers)
res += currentGameState.getScore()
return res
```

کد این قسمت مشابه قسمت (1) است. در بخش اول فاصله با نزدیک ترین غذا محاسبه می‌شود و در صورت غیر صفر بودن، معکوسش به حاصل نهایی تابع ارزیابی ما اضافه می‌شود (زیرا این فاصله هرچه کمتر باشد بهتر است). در بخش دوم نیز فاصله تا نزدیک‌ترین روح محاسبه و پس از scale شدن به صورت مستقیم با خروجی جمع می‌شود (زیرا هرچه این فاصله بیشتر باشد بهتر است). در ادامه هم جمع تایم ترسیده بودن روح‌ها و امتیاز حالت فعلی، به مقدار نهایی اضافه می‌شوند و این مقدار برگردانده می‌شود.

سوال 5) تابع ارزیابی این قسمت برخلاف قسمت (1)، حالت‌های حاصل از action ها را در نظر نمی‌گیرد و تنها حالت فعلی عامل را معیار قرار می‌دهد. همچنین خروجی تابع با دقت بیشتری محاسبه می‌شود و یک عدد دقیق‌تر نسبت به قسمت (1) می‌دهد (در قسمت (1) خروجی فقط یکی از چند مقدار مشخص می‌توانست باشد) و به خاطر همین دقت بیشتر است که عامل ما بهتر به سمت حالت‌های خوب هدایت می‌شود و این تابع ارزیابی عملکرد بهتری دارد. به خاطر کمتر بودن محاسبات از پیچیدگی زمانی کمتری نیز برخوردار است.