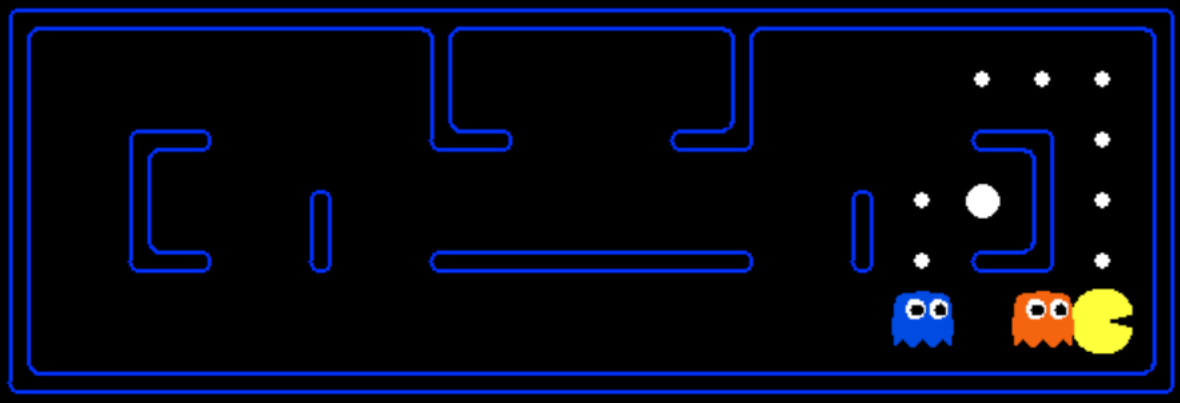

SCORE: 531

SCORE: 8

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v > β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v < α return v
        β = min(β, v)
    return v
```

multiagent > multiAgents.py > ExpectimaxAgent > getAction

```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning (question 3)
    """

    def getAction(self, gameState: GameState):
        """
        Returns the minimax action using self.depth and self.evaluationFunction
        """
        def minimizer(gameState, alpha, beta, agentIndex, depth):
            v = float('inf')
            nextIndex = (agentIndex + 1) % gameState.getNumAgents()
            for action in gameState.getLegalActions(agentIndex):
                successor = gameState.generateSuccessor(agentIndex, action)
                v = min(v, value(successor, alpha,beta, nextIndex, depth + 1 if nextIndex==0 else depth ))
                if v < alpha:
                    return v
                else:
                    beta = min(v, beta)
            return v

        def maximizer(gameState, alpha, beta, depth):
            v = float('-inf')

            for action in gameState.getLegalActions(0):
                successor = gameState.generateSuccessor(0, action)
                v = max(v, value(successor, alpha, beta, 1, depth))

                if v > beta:
                    return v

                else:
                    alpha = max(v, alpha)
            return v

        def value(gameState, alpha, beta, agentIndex, depth):
            if gameState.isWin() or gameState.isLose() or depth == self.depth:
                return self.evaluationFunction(gameState)
            if agentIndex == 0:
                return maximizer(gameState, alpha,beta, depth)
            else:
                return minimizer(gameState, alpha, beta, agentIndex, depth)


        alpha, bestMove = float('-inf'), None

        for action in gameState.getLegalActions(0):
            v = gameState.generateSuccessor(0, action)
            x = value(v, alpha, float('inf'), 1, 0)
            if x > alpha:
                alpha = x
                bestMove = action
        return bestMove
```

```python
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (question 4)
    """
    def getAction(self, gameState: GameState):

        def expectimax(gameState: GameState,action,agentIndex, depth):
            if gameState.isWin() or gameState.isLose() or depth == self.depth:
                return self.evaluationFunction(gameState)

            if agentIndex == 0:
                return maximizer(agentIndex, action, gameState, depth)
            else:
                return expected(agentIndex, action, gameState, depth)

        def expected(agentIndex,action,gameState,depth):
            EV = 0
            actions = gameState.getLegalActions(agentIndex)

            for action in actions:
                successor = gameState.generateSuccessor(agentIndex, action)
                nextAgentIndex = (agentIndex + 1) % gameState.getNumAgents()
                EV += expectimax(successor, action,nextAgentIndex, depth + 1 if nextAgentIndex == 0 else depth)

            return EV / len(actions)

        def maximizer(agentIndex, action,gameState,depth):
            score = float('-inf')

            for action in gameState.getLegalActions(agentIndex):
                successor = gameState.generateSuccessor(agentIndex, action)
                a = expectimax(successor, action,(agentIndex+1) % gameState.getNumAgents(), depth)
                score = max(score,a)
            return score
        #Returns whether or not the game state is a losing state


        score, bestMove = float('-inf'), None
        for action in gameState.getLegalActions(0):
            v = gameState.generateSuccessor(0, action)
            x = expectimax(v, action, 1, 0)
            if x > score:
                score = x
                bestMove = action
        return bestMove
```