مبانی و کاربرد های هوش مصنوعی

فاز چهارم پروژه

جست و جو در محیط های تخاصمی

Search in Adversarial Environments

And

Game Theory

محمد حسن حیدری – آرین جعفری

- معرفی

هدف این پروژه پیاده‌سازی یک عامل هوشمند است که بتواند در محیط بازی Pacman به فعالیت بپردازد . در این محیط که توسط دانشگاه کالیفرنیا – برکلی پیاده سازی شده است ، عامل درون یک هزارتو میگردد و تعداد زیادی نقطه کوچک و چند نقطه بزرگ را میخورد . در این بازی هدف این است که عامل نقطه هارا بخورد و در عین حال از برخورد با شبح بپرهیزد .

تفاوت این محیط با محیط های فاز های قبلی پروژه ، این است که محیط Pacman یک محیط با چند عامل و تخاصمی است . در صورتی که شبح ها به عامل برخورد کنند ، بازی تمام میشود ، در عین حال با خوردن تمامی نقطه ها نیز بازی تمام خواهد شد . عامل علاوه بر اینکه باید میزان امتیاز خود را از طریق خوردن نقطه ها به حداکثر برساند ، باید حداقل تعامل ممکن را با اشباح داشته و از آنها فاصله بگیرد .

- الگوریتم Minimax

این الگوریتم ، نوعی الگوریتم BackTracking است که در تصمیم گیری و نظریه بازی استفاده میشود ، هدف این الگوریتم پیدا کردن عمل بهینه برای عامل در هر موقعیت است با فرض اینکه رقیب نیز به صورت بهینه عمل میکند .

در مینی مکس با دو عامل ، یکی از عامل ها Maximizer و دیگری Minimizer نامیده میشوند . ماکسیمایزر تلاش میکند تا بیشترین امتیاز ممکن را از بازی بگیرد اما مینیمایزر بر خلاف این هدف عمل میکند . بازی هایی مانند شطرنج و Tic-Tac-Toe از این موارد هستند .

- شبه کد و نحوه عملکرد الگوریتم MiniMax

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

- هرس آلفا – بتا

با وجود اینکه درخت جست و جوی مینی مکس کمک میکند تمامی حالت های ممکن برای عمل حریف فرض شوند ، بسیاری از بخش ها و شاخه های این درخت ، تاثیری در خروجی نخواهند داشت و محاسبه آنها فقط باعث تلف شدن زمان و پرداخت هزینه برای پردازش شاخه های بدون نتیجه خواهد شد . هرس آلفا – بتا کمک میکند تا درخت جست و جو بسیار بهینه تر شود .

$\alpha =$ the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX. Think: $\alpha =$ "at least."

$\beta =$ the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN. Think: $\beta =$ "at most."

- شبه کد و نحوه عملکرد هرس آلفا – بتا

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

- هرس آلفا – بتا به کمک تابع اکتشاف

به جهت بهینه تر کردن فرایند جست و جو ، میتوانیم از تابع اکتشافی نیز بهره بگیریم . وظیفه این تابع در محیط بازی تخمین سودمندی هر راس در درخت است . به علت محدودیت در زمان و منابع ، استفاده از تابع اکتشاف میتواند به عامل کمک کند تمام محیط بازی را نگردد و تنها به سمت راس هایی با سودمندی بیشتر حرکت کند .

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s,\text{MAX}) & \text{if IS-CUTOFF}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if TO-MOVE}(s) = \text{MIN}. \end{cases}$$

((در این فاز ، ما از الگوریتم مینی ماکس به همراه استفاده از تکنیک هرس آلفا – بتا و تابع اکتشاف برای جست و جو در بازی کمک میگیریم))

- پیاده سازی

```
# evaluation function for minimax with alpha beta prunning
# heuristic is based on the distance of the closest food, the distance of the
closest ghost and the number of ghosts in 1 distance

def evaluationFunction(self, game_state, action):
    # get the next game state and the position of pacman
    s_game_state = game_state.generatePacmanSuccessor(action)
    position = s_game_state.getPacmanPosition()

    # get the food ositions
    food = s_game_state.getFood()

    # list of food positions
    foods = food.asList()

    # distance of closest food
    d_foods = -1
    for food in foods:  # for each food calculate the distance
        distance = manhattanDistance(position, food)
        # calculate the distance based on manhattan distance
        if d_foods >= distance or d_foods == -1:  # if the distance is
smaller than the previous one or if it is the first one
            d_foods = distance  # update the distance

    # distance of closest ghost
    d_ghosts = 1

    # number of ghosts in 1 distance
    p_ghosts = 0
    for ghost_state in s_game_state.getGhostPositions():  # for each ghost
calculate the distance
        distance = manhattanDistance(position, ghost_state)
        d_ghosts += distance
        if distance <= 1:  # if the distance is smaller than 1
            p_ghosts += 1  # increase the number of ghosts in 1 distance

    # return the score of the game state plus the heuristic
    # heuristic is based on the distance of the closest food, the distance of
the closest ghost and the number of ghosts in 1 distance
    # hyperparameters are chosen based on trial and error :)

    score = s_game_state.getScore() + (10000 / float(d_foods)) - (0.01 /
float(d_ghosts)) - p_ghosts

    return score
```

- تابع اکتشافی پیاده شده ، نیاز برای حرکت به سمت غذا ها و در عین حال اجتناب از اشباح و مجاورت نسبت به آنهارا متعادل میکند . تابع ابتدا موقعیت عامل پس از انجام کنش را تولید کرده و موقعیت فعلی عامل را در می یابد . سپس فاصله منهتن از تمام

نقطه غذاهارا محاسبه کرده و نزدیک ترین آنهارا در نظر میگیرد ، این عامل را تشویق میکند به سمت نزدیک ترین نقطه غذا حرکت کند . تابع سپس برای تمامی اشباح موجود در بازی ، فاصله آنهارا محاسبه میکند و تعداد ارواح مجاور با فاصله برابر با 1 را ذخیره میکند . این مورد به عامل کمک میکند از نزدیکی زیاد به اشباح اجتناب کند . در نهایت با جمع ارزش تمامی موارد موثر بر سودمندی ، با وزن 1000 برای نقطه غذا ها و 0.01 برای تمامی اشباح و 1 برای اشباح مجاور ، امتیاز محاسبه شده را باز میگرداند (هایپر پارامتر های استفاده شده بر اساس آزمون و خطا انتخاب شده اند) .

سپس در بخش اصلی کلاس ، در تابع getAction() ، بر اساس تابع اکتشافی محاسبه شده و درخت جست و جوی مینی مکس ، کنش بهینه در هر موقعیت را برای عامل محاسبه میکنیم . این تابع شامل توابع max_value() ، min_value() و a_b_prunning() است که هر بخشی از الگوریتم مینی ماکس با هرس آلفا – بتا را انجام میدهند .

```
def getAction(self, gameState):

    def max_value(agent, depth, game_state, a, b):  # max_value function
        # if the game is won/lost or the defined depth is reached return the
utility

        value = -10e12  # set the value to a very small number
        for s in game_state.getLegalActions(agent):  # for each action
calculate the value

            prunned = a_b_prunning(1, depth,
game_state.generateSuccessor(agent, s), a,
                                   b)  # calculate the value of the next
level

            value = max(value, prunned)  # update the value

            if value > b:  # if the value is bigger than beta return the
value
                return value

            a = max(a, value)  # update alpha

        return value

    def min_value(agent, depth, game_state, alpha, beta):  # min_value
function


        value = 10e12  # set the value to a very big number

        next_agent = agent + 1  # calculate the next agent and increase depth
accordingly.
        if game_state.getNumAgents() == next_agent:  # if the next agent is
the last one
            next_agent = 0  # set the next agent to pacman
        if next_agent == 0:  # if the next agent is pacman
            depth += 1  # increase the depth

        for s in game_state.getLegalActions(agent):  # for each action
calculate the value
            value = min(value, a_b_prunning(next_agent, depth,
game_state.generateSuccessor(agent, s), alpha,
                                             beta))  # calculate the value of
the next level
            if value < alpha:  # if the value is smaller than alpha return
the value
                return value
            beta = min(beta, value)  # update beta
        return value

    def a_b_prunning(agent, depth, game_state, alpha, beta):  # alpha beta
prunning function
        if game_state.isLose() or game_state.isWin() or depth == self.depth:
```

```
            # return the utility in case the defined depth is reached or the
game is won/lost.
            return self.evaluationFunction(game_state)

        if agent == 0:  # if the agent is pacman
            return max_value(agent, depth, game_state, alpha, beta)  # return
the max level

        else:  # if the agent is a ghost
            return min_value(agent, depth, game_state, alpha, beta)  # return
the min level

    utility, alpha, beta = -10e12, -10e12, 10e12  # set the utility, alpha
and beta to a very small numbers respectievly

    action = Directions.NORTH  # set the action to north

    legal_actions = gameState.getLegalActions(0)  # get the legal actions of
pacman
    random.shuffle(legal_actions)  # Shuffle the actions to add randomness

    for s in legal_actions:  # for each action calculate the utility
        ghostValue = a_b_prunning(agent=1, depth=0,
game_state=gameState.generateSuccessor(0, s), alpha=alpha,
                                  beta=beta)  # calculate the utility of the
next level

        # if the utility is bigger than the previous one update the utility
and the action
        if ghostValue > utility:
            utility = ghostValue  # update the utility
            action = s  # update the action
        if utility > beta:  # if the utility is bigger than beta return the
utility
            return utility
        alpha = max(alpha, utility)  # update alpha

    return action
```

- تابع getAction ، تابع اصلی انتخاب گر کنش عامل است ، ابتدا مقادیر آلفا و بتا و سودمندی را مقدار دهی اولیه میکنیم ، سپس تمامی کنش های قانونی ای که عامل میتواند انتخاب کند در نظر میگیریم . به ازای تمامی کنش ها ، سودمندی وضعیت متناظر با آن را از طریق تابع a_b_prunning محاسبه میکنیم ، اگر سودمندی آن کنش بهتر از سودمندی فعلی بود ، سودمندی را به روز کرده و همچنین کنش متناظر با آن را در نظر میگیریم . اگر سودمندی از بتا بزرگ تر بود ، در همان لحظه سودمندی را باز میگردانیم و در نهایت کنشی با بیشترین سودمندی را باز میگردانیم .

- تابع max_value را وقتی فراخوانی میکنیم که بازیکن فعلی ، عامل ما یعنی Pacman باشد (ماکسیمایزر) ، این تابع بیشترین سودمندی ای که عامل در وضعیت فعلی میتواند بدست بیاورد را محاسبه میکند . تابع بر روی تمامی کنش های ممکن میگردد و وضعیت وارث را محاسبه میکند . سپس به کمک a_b_prunning سودمندی هر وضعیت وارث را محاسبه میکند ، اگر سودمندی آن وضعیت از بتا بیشتر بود ، بلافاصله آن را بر میگرداند ، همچنین بزرگترین آلفای دیده شده تا کنون را نیز مورد توجه قرار میدهد .

- تابع min_value زمانی فراخوانی میشود که کنشگر ، شبح باشد (مینیمایزر) ، این تابع کمترین سودمندی ای که اشباح ، پکمن را مجبور به دریافت میکند حساب میکند .این تابع مشابه با max_value کار میکند با این تفاوت که کوچک ترین بتا را در مورد توجه قرار میدهد .

- تابع a_b_prunning منطق اصلی هرس آلفا بتا را پیاده سازی میکند . ابتدا بررسی میکند که آیا بازی تمام شده و یا به حداکثر عمق رسیده ایم ، در این صورت سودمندی وضعیت فعلی را باز میگرداند . در غیر این صورت توابع max_value و یا min_value را متناسب با اینکه بازیکن پکمن و یا شبح است فرا می خواند

- عبارت **random.shuffle(legal_actions)** کنش های ممکن را به صورت رندوم مرتب میکند تا عامل در صورت گیر کردن در نقطه مینیمم محلی ، به جای انتخاب عمل توقف ، به صورت رندوم عمل کند

- منابع استفاده شده

- اسلاید های تدریس شده در دوره مبانی و کاربرد های هوش مصنوعی دانشگاه اصفهان تدریس شده توسط دکتر حسین کارشناس ، پاییز 1402
- داکیومنت پروژه Pacman از دانشگاه کالیفرنیا – برکلی
- AI,A Modern Approach, Russell-Norvig, 4th Edition
- Youtube
- Microsoft Copilot
- Github Copilot