Minimax Algorithm

2110327 Algorithm Design

นาย นิศรุจ รัตนอร่าม 6031033521

ภาควิชาวิศวกรรมคอมพิวเตอร์, คณะวิศวกรรมศาสตร์, จุฬาลงกรณ์มหาวิทยาลัย

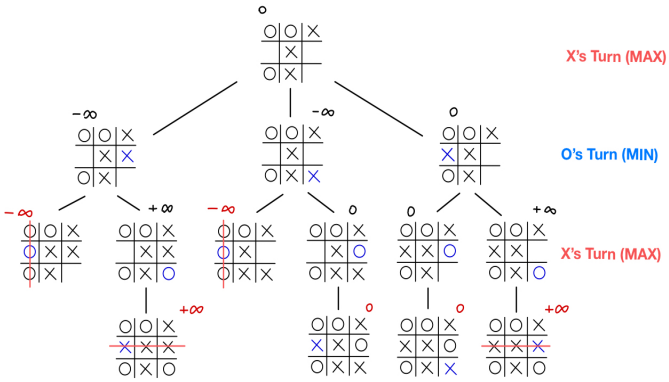
ขั้นตอนวิธี Minimax หรือการหาเกณฑ์ค่าเสียโอกาสมากน้อยที่สุด คือขั้นตอนวิธีในการตัดสินใจหลีกเลี่ยงโอกาสที่จะทำให้เกิดความสูญเสียมากที่สุดในการเล่นเกมเชิงตรรกะที่มีผู้เล่นสองคน กล่าวอีกนัยหนึ่งคือ "เป็นขั้นตอนในการตัดสินใจเดินหมากโดยให้ผู้ที่ทำการเดินได้เปรียบมากที่สุด" ขั้นตอนวิธีนี้ได้นำมาประยุกต์ใช้ได้ในหลายแขนงวิชา เช่นทฤษฎีเกม สถิติ ปรัชญา และโดยเฉพาะอย่างยิ่งทางด้านปัญญาประดิษฐ์ที่ปรากฏในเกมต่างๆ ตัวอย่างเกมที่นิยมใช้นำมาเป็นกรณีศึกษา Minimax คือ โอเอกซ์ หมากฮอส และหมากรุก

1. หลักการทำงาน

Minimax ใช้หลักการของการค้นแบบจำกัดความลึก (Depth Limited Search) ซึ่งถูกปรับปรุงมาจากการค้นหาเชิงลึก (Depth First Search) โดยมีการเพิ่มขีดจำกัดในการท่องกราฟว่าให้ค้นหาไม่เกินความลึกใด

ในระหว่างการวน recursive ของการค้นแบบจำกัดความลึกนี้ เราจะเก็บค่าตัวแปรของ**ค่าความได้เปรียบ**ของผู้เล่นที่พิจารณาในแต่ละสถานะของเกม ตัวแปรนี้สามารถมีค่าได้ตั้งแต่ติดลบอนันต์ ไปจนถึงอนันต์ โดยจะมีค่ามากกว่าศูนย์เมื่อฝ่ายเราได้เปรียบ และหากมีค่ามากจนถึงอนันต์ นั่นคือฝ่ายเราชนะแน่นอน ในทางตรงกันข้าม หากตัวแปรนี้มีค่าน้อยกว่าศูนย์จนถึงลบอนันต์ หมายความว่าฝ่ายเราเสียเปรียบน้อยไปจนถึงแพ้แน่นอน

ด้วยค่านี้ ทำให้เราสามารถตัดสินใจได้ด้วยให้ผู้เล่นที่พิจารณาเลือกเดินหมากที่ทำให้ได้ค่าตอบแทนสูงที่สุด (Max's player) ในขณะที่ให้ฝ่ายตรงข้ามจะเลือกเดินหมากที่ทำให้ผู้เล่นที่พิจารณานั้นได้ค่าตอบแทนที่น้อยที่สุด (Min's player) ซึ่งหมายความว่ากำหนดให้ฝั่งตรงข้ามเล่นเกมอย่างสุขุมที่สุดจนทำให้เราได้เปรียบน้อยที่สุดนั่นเอง



รูปที่ 1 ตัวอย่างต้นไม้เกมของเกม OX

รูปที่ 1 แสดงต้นไม้เกมของโอเอกซ์ในมุมมองของผู้เล่น X ซึ่งเป็น Max's player เมื่อโปรแกรมทำการ recursive จนถึงสถานะสุดท้าย (Terminal state) โปรแกรมจะคืนค่าความได้เปรียบของ X ออกมาตามผลแพ้ชนะ (+∞ เมื่อ X ชนะ, -∞ เมื่อ X แพ้ และ 0 คือเสมอ ตามตัวสีแดงของรูปที่ 1) หลังจากนั้นโปรแกรมจะทำการย้อนกลับไปสถานะก่อนหน้าเพื่อคำนวณค่าความได้เปรียบ โดยหากชั้นนั้นเป็นตาของ X (Max's player) ให้

เลือกค่าที่มากที่สุดจากทุกๆ สถานะถัดไปที่เป็นไปได้ แต่ถ้าชั้นนั้นเป็นตาของ O (Min's player) ให้เลือกค่าที่น้อยที่สุดที่เป็นไปได้แทน เมื่อทำเช่นนี้เรื่อยๆ ที่รากของต้นไม้จะเก็บค่าความได้เปรียบที่มากที่สุด

2. โปรแกรม

จากรหัสที่ 1 แสดงรหัสเทียมของฟังก์ชัน `minimax` ซึ่งมีลักษณะคล้ายการค้นแบบจำกัดความลึก ประกอบด้วยพารามิเตอร์ 3 ค่า ได้แก่ `presentState` ใช้สำหรับเก็บสถานะปัจจุบัน (อาจเป็นอาเรย์ 1 มิติ จำนวน 9 ช่อง ในกรณีเกม OX) `depth` เป็นจำนวนเต็มเก็บความลึก ณ สถานะปัจจุบันและ `isMaxPlayer` เป็นบูลลีนเพื่อบ่งบอกว่าที่สถานะนี้ เป็นตาเล่นของ Max's Player หรือไม่

เริ่มต้นฟังก์ชันนี้จะถูกเรียกด้วย `minimax (initialState, maxDepth, TRUE)` จากนั้นจะทำการ recursive ไปเรื่อยๆ จนกระทั่งเมื่อถึงสถานะสุดท้าย หรืออีกกรณีหนึ่งคือค่า `depth` เป็น 0 นั่นคือท่องกราฟจนถึงขีดจำกัดความลึกแล้ว ซึ่งมักเกิดกับเกมที่มีความเป็นไปได้ของกระดานเยอะมากๆ เช่น หมากรุก หรือ โกะ หากเป็นกรณีนี้ เราจะไม่สามารถตัดสินได้ว่าใครเป็นผู้ชนะ เราจึงทำได้เพียงประเมินค่าความได้เปรียบจากสถานะปัจจุบันเท่านั้น เราเรียกฟังก์ชันการประเมินนี้ว่า Heuristic function หรือ Evaluation function ซึ่งจะแตกต่างกันออกไปตามประเภทของเกมและความเก่งในการตัดสินใจของโปรแกรม (ระดับความยาก-ง่ายของ AI) สูตรของฟังก์ชันนี้จะคำนวณจากหลายปัจจัยภายในเกม เช่น ในสองทศวรรษที่ผ่านมา (1990) AI สำหรับเล่นหมากฮอสชื่อ Chinook สามารถเอาชนะผู้เล่นระดับโลกได้ มีการใช้สูตรที่คำนวณจากจำนวนและตำแหน่งของเบี้ย (Pawn) และตัวฮอส (King) รวมไปถึงจำนวนเบี้ยที่พยายามหนี

หากฟังก์ชัน `minimax` ไม่ได้มี `depth` เป็น 0 และไม่ได้เป็น terminal state ให้ทำการหาค่าความได้เปรียบจากทุกสถานะถัดไปที่เป็นไปได้ ซึ่งขึ้นอยู่กับผู้เล่นของตานั้นๆ โดยหาก `isMaxPlayer` เป็นจริง ให้ทำการหาค่าที่มากที่สุด และหากเป็นเท็จ ให้หาค่าที่น้อยที่สุด สังเกตว่า `isMaxPlayer` จะสลับกันไปเรื่อยเพราะว่าตาของผู้เล่นจะสลับไปเรื่อยๆ นั่นเอง

3. Alpha-Beta Pruning

การลดทอนโดยอัลฟาเบตา เป็นขั้นตอนวิธีที่พัฒนาต่อจาก Minimax เพื่อให้โปรแกรมทำงานได้เร็วยิ่งขึ้น โดยจะตัดบางกิ่งที่ไม่จำเป็นต้องพิจารณา เนื่องจากเราทราบว่ามีทางที่ดีกว่ากิ่งนั้นแน่ๆ

```
def minimax(presentState, depth, isMaxPlayer):
    if depth == 0 or presentState is a terminal state:
        return the heuristic value of presentState
    if isMaxPlayer:
        value = −∞
        for each nextState of presentState:
            value = max(value, minimax(nextState, depth − 1, FALSE))
        return value
    else:
        value = +∞
        for each nextState of presentState:
            value = min(value, minimax(nextState, depth − 1, TRUE))
        return value
```

รหัสที่ 1 ฟังก์ชัน minimax

ขั้นตอนวิธีจะแตกต่างจาก minimax เพียงเล็กน้อย โดยนอกจากที่แต่ละสถานะจะเก็บค่าความได้เปรียบแล้ว ให้ทำการเก็บเพิ่มอีกสองตัวแปร คือ ขีดจำกัดล่าง และขีดจำกัดบน เรียกว่า alpha และ beta โดยมีกฎว่า α ของโหนด Max คือขอบเขตน้อยที่สุดที่เป็นไปได้ ซึ่งถ้าหากค่าของโหนด Min มีค่าน้อยกว่าหรือเท่ากับ α ของโหนดพ่อแม่ (ซึ่งต่ำกว่าขอบเขตล่างที่เรากำหนด) ก็ให้หยุดลูปในการแตกกิ่งได้เลย ในขณะเดียวกัน เราให้ β ของโหนด Min คือขอบเขตที่มากที่สุด ซึ่งถ้าหากค่าของโหนด Max มีค่ามากกว่าหรือเท่ากับ β ก็ให้หยุดการแตกกิ่งต่อ

สำหรับรหัสเทียมฟังก์ชัน `alphabeta` จะเห็นได้ว่ามีการเพิ่มเติมจากฟังก์ชัน `minimax` ของรหัสที่ 1 นั่นคือมีการกำหนดค่า α และ β อีกทั้งเพิ่มการตรวจสอบเงื่อนไขเพื่อหยุดการแตกกิ่งต่อดังรหัสที่ 2 ส่วนการเริ่มเรียกฟังก์ชันให้ใช้อาร์กิวเมนต์ดังนี้ `alphabeta(initialState, maxDepth, -∞, +∞, TRUE)`

4. การวิเคราะห์ประสิทธิภาพการทำงาน

การทำงานของขั้นตอนวิธี minimax เป็นไปตามต้นไม้สถานะที่มี branching factor b (ในที่นี้คือจำนวนสถานะถัดไปที่เป็นไปได้) และต้นไม้มีความลึก d จะทำให้มีจำนวนโหนดใบที่มากที่สุดที่ต้องพิจารณาทั้งหมด $O(b^d)$ ซึ่งยังเป็นประสิทธิภาพการทำงานของ alphabeta ในกรณีที่แย่ที่สุดด้วย สำหรับกรณีที่ดีที่สุดจะเกิดขึ้นเมื่อสามารถตัดกิ่งได้ทันทีตั้งแต่แตกกิ่งครั้งแรกในชั้น Min's player ทำให้แตกกิ่งแค่ในชั้นของ Max's player ซึ่งมีอยู่ $\frac{d}{2}$ ชั้นทำให้โปรแกรมทำงานเพียง $O\left(b^{\frac{d}{2}}\right) = O(\sqrt{b^d})$

5. เอกสารอ้างอิง

[1] บุญเสริม กิจศิริกุล, คอมพิวเตอร์หมากรุกไทย, 2539

[2] https://th.wikipedia.org/wiki/ขั้นตอนวิธีมินิแมกซ์

[3] https://en.wikipedia.org/wiki/Minimax

[4] https://en.wikipedia.org/wiki/Alpha–beta_pruning

[5] https://en.m.wikipedia.org/wiki/Chinook_(draughts_player)

Let's play tic-tac-toe: https://github.com/nisaruj/tictactoe-ai

```
def alphabeta(presentState, depth, α, β, isMaxPlayer):
    if depth == 0 or presentState is a terminal state:
        return the heuristic value of presentState
    if isMaxPlayer:
        value = -∞
        for each nextState of presentState:
            value = max(value, minimax(nextState, depth - 1, FALSE))
            α = max(α, value)
            if α >= β:
                break
        return value
    else:
        value = +∞
        for each nextState of presentState:
            value = min(value, minimax(nextState, depth - 1, TRUE))
            β = min(β, value)
            if α >= β:
                break
        return value
```

รหัสที่ 2 ฟังก์ชัน alphabeta