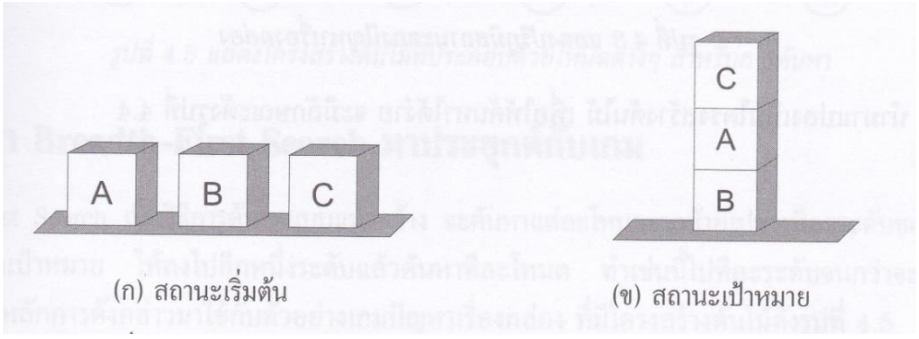
บทที่ 2
รูปแบบของปัญญาประดิษฐ์ที่ใช้ภายในเกม

ปัญญาประดิษฐ์ได้ถูกนำมาช่วยอำนวยความสะดวก และแก้ปัญหาต่างๆ ให้กับมนุษย์ จุดเด่นของเกมที่มีปัญญาประดิษฐ์คือ สามารถนำเสนอความน่าสนใจ และให้อารมณ์ในการเล่นเกม ที่มีความเสมือนจริง ราวกับผู้เล่นกำลังแข่งขัน กับสิ่งที่สามารถตัดสินใจได้รวดเร็วเทียบเท่าหรือเร็วกว่ามนุษย์ ทำให้รูปแบบของเกมมีความน่าสนุก ในบทนี้จะยกตัวอย่างเกมกับการประยุกต์ด้วย Blind Search Technique เกมกับการประยุกต์ด้วย Heuristic Search Technique และเกมกับการประยุกต์ด้วย Adversarial Search Techniques

เกมกับการประยุกต์ด้วย Blind Search Technique

ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. (2552: 83-87) อธิบายไว้ว่า การประยุกต์โดยใช้เทคนิคการค้นหาแบบ Blind Search Technique ร่วมกับเกมต่างๆ เป็นเทคนิคการค้นหาแบบไม่มีข้อมูลมาประกอบการพิจารณา ตัวอย่างโครงสร้างที่นำมาใช้อธิบายเป็นรูปแบบต้นไม้ ตัวอย่างอัลกอริทึมที่นำมาใช้คือ Breadth-First Search และ Depth-First Search

ตัวอย่างเกมที่นำมาใช้ในการพัฒนาคือ เกมปัญหาเรื่องกล่อง ประกอบด้วยกล่องจำนวน 3 กล่อง ด้านหน้าแต่ละกลล่องจะมีตัวอักษร “A” “B” และ “C” ตามลำดับ วางอยู่บนโต๊ะ ผู้เล่นสามารถหยิบกล่อง เพื่อนำไปวางในตำแหน่งต่างๆ ได้ เช่น วางบนโต๊ะ หรือวางบนกล่องอื่นก็ได้ โดยหยิบได้เพียงกล่องเดียวต่อหนึ่งครั้ง และไม่อนุญาตให้หยิบกล่องอื่นที่ซ้อนทับอยู่ ดกำหนดให้สถานะเป้าหมายของปัญหามีลักษณะดังภาพที่ 2.1



ภาพที่ 2.1 สถานะเริ่มต้นและสถานะเป้าหมายสำคัญเกมปัญหาเรื่องกล่อง
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 83)

หากนำปัญหามาเขียนในรูปแบบ Problem Formulation จะได้ดังนี้

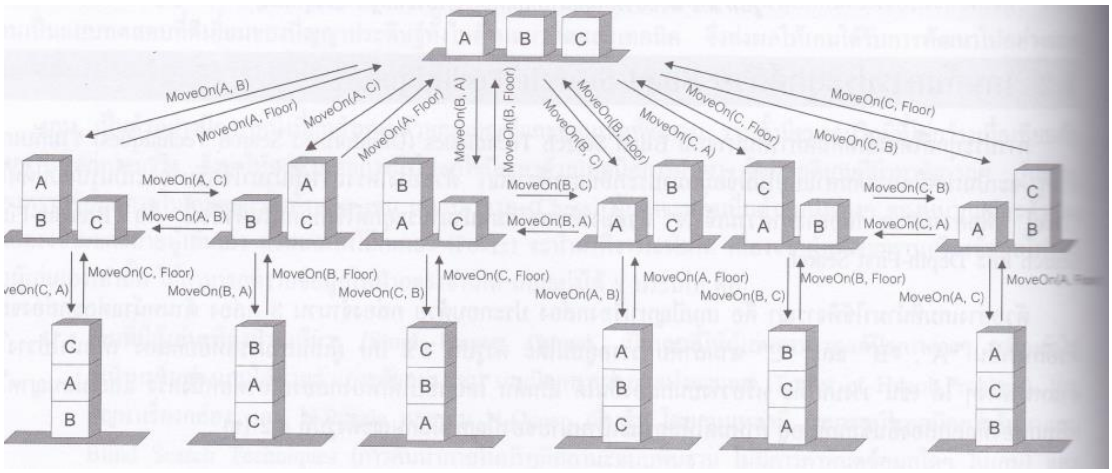
Inittial State: สถานะเริ่มต้น ดังภาพที่ 2.1

Successor Function: กำหนดให้ MoveOn(x,y) หมายถึง เคลื่อนย้ายกล่อง x นำมาวางไว้บนกล่อง y

Goal State: สถานะเป้าหมายดังภาพที่ 2.1

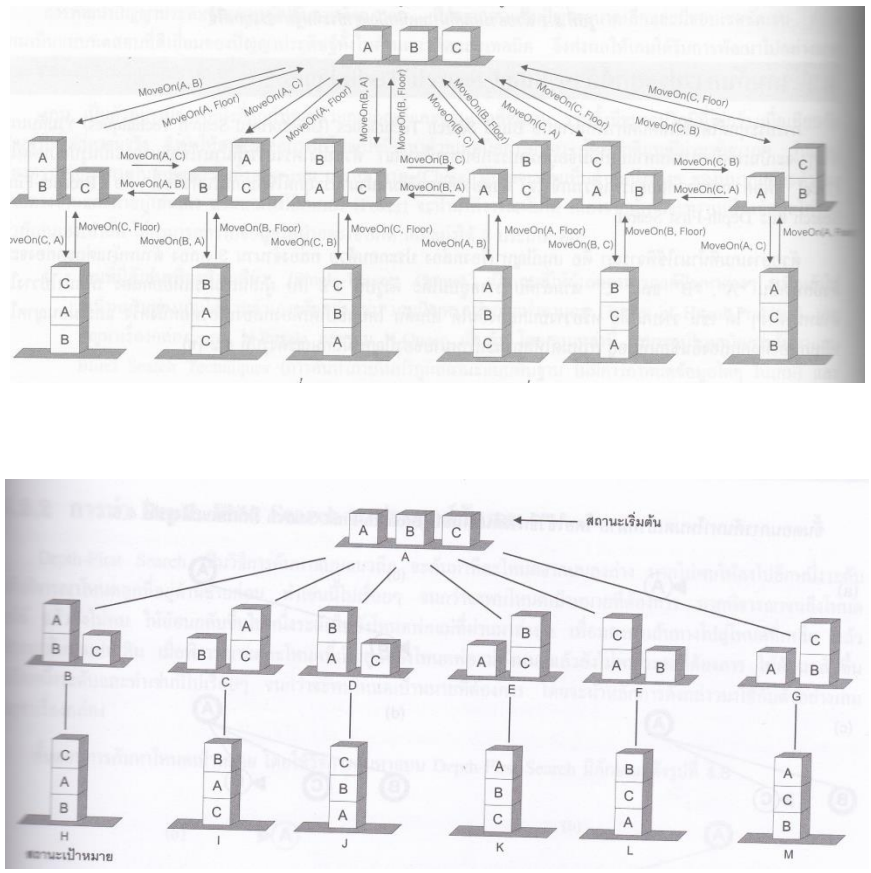
Path Cost: การหยิบกล่องไปวางตำแหน่งอื่น ในแต่ละครั้งจะใช้ทรัพยากรเป็น 1 ดังนั้น

Path Cost ที่จะใช้ขึ้นอยู่กับจำนวนครั้งของการหยิบกล่องทั้งหมด



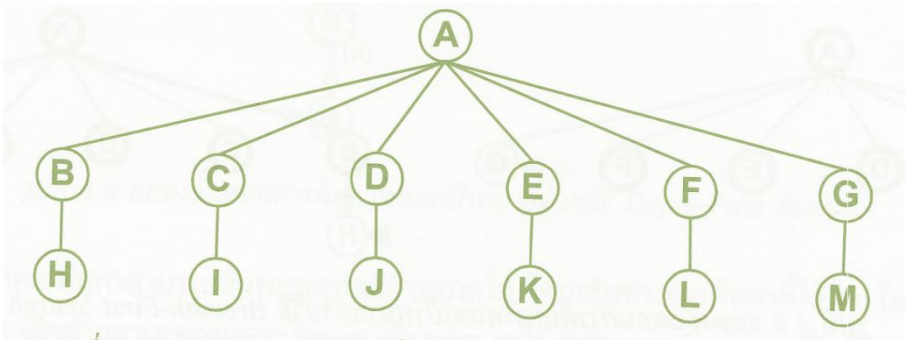
ภาพที่ 2.2 ปริภูมิสถานะของปัญหาเรื่องกล่อง
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 84)

จากภาพ 2.2 เป็นการนำมาแปลงเป็นโครงสร้างต้นไม้เพื่อให้ค้นหาง่าย ดังภาพที่ 2.3



ภาพที่ 2.3 โครงสร้างต้นไม้ที่ประกอบด้วยสถานะต่างๆ สำหรับหารค้นหา
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 85)

จากภาพที่ 2.3 นำสถานะต่างๆ มาแทนเป็นโหนดของต้นไม้ ดังนั้นโหนด A จะเป็นโหนดเริ่มต้น และโหนด H จะเป็นโหนดเป้าหมาย จะมีลักษณะดังภาพที่ 2.4

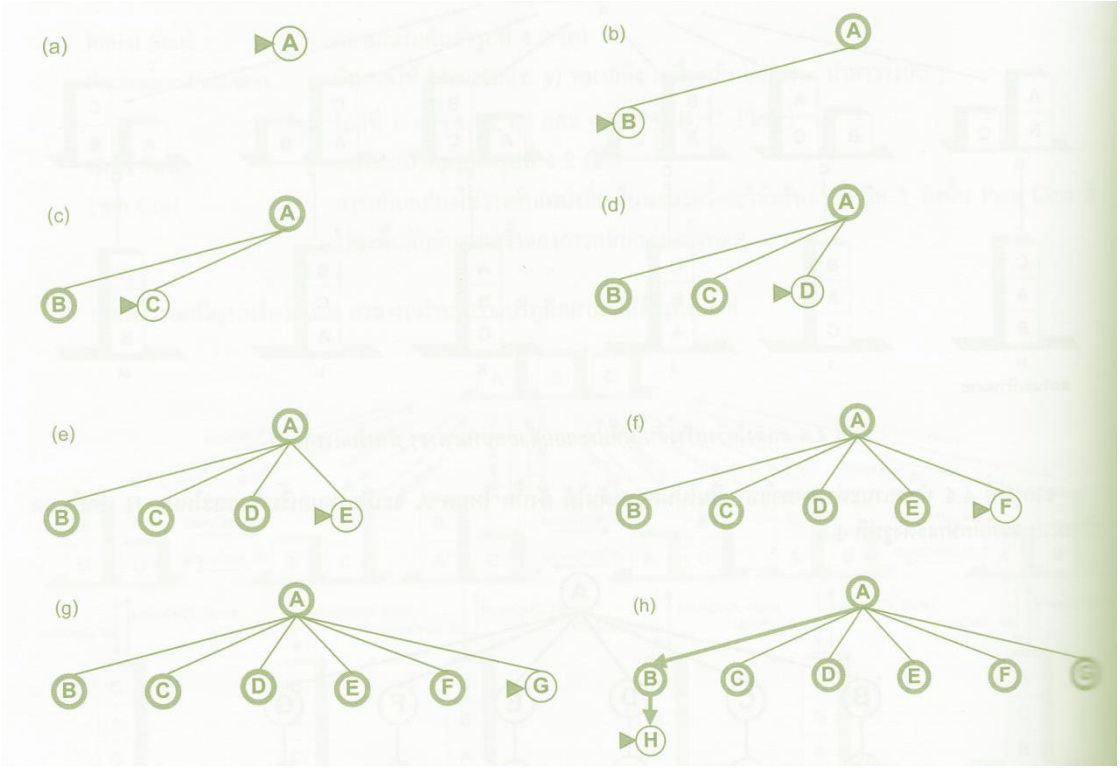


ภาพที่ 2.4 โครงสร้างต้นไม้ที่ประกอบด้วยสถานะต่างๆ สำหรับหารค้นหา
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 85)

1. การนำ Breadth-First Search มาประยุกต์กับเกม

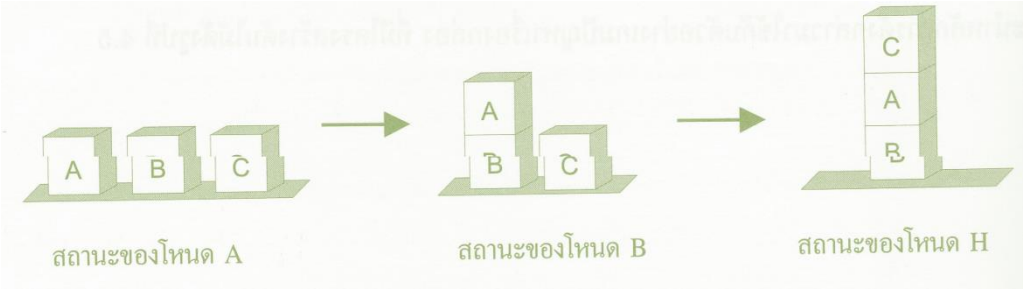
Breadth-First Search เป็นวิธีการค้นหาแบบแนวกว้าง จะค้นหาแต่ละโหนดจากซ้ายไปขวาทีละระดับของต้นไม้จากบนลงล่างหากยังไม่พบโหนดเป้าหมายให้ลงไปอีกหนึ่งระดับและค้นหาทีละโหนด ทำเช่นนี้ไปทีละระดับจนกว่าจะพบโนดเป้าหมายที่ต้องการ โดยจะนำหลักการดังกล่าวมาใช้กับตัวอย่าง เกมปัญหาเรื่องกล่อง ที่มีโครงสร้างต้นไม้ดังภาพที่ 2.4

ขั้นตอนการค้นหาโหนดเป้าหมาย โดยใช้วิธีการค้นหาแบบ Breadth-First Search มีลักษณะดังภาพที่ 2.5



ภาพที่ 2.5 ขั้นตอนการค้นหาโหนดเป้าหมายด้วยวิธี Breadth-First Search
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 86)

จากภาพที่ 2.5 พบว่าการค้นหาสามารถค้นพบสถานะเป้าหมายได้ โดยเส้นทางการค้นหาที่ได้คือ โหนด A โหนดB และ โหนดH ตามลำดับ ทำให้ได้ค่า Path Cost ใน Problem Formulation ของผลลัพธ์ตัวอย่างนี้เป็น 2 (เนื่องจากการหยิบกล่องไปวางบนตำแหน่งอื่น ในแต่ละครั้งจะใช้ทรัพยากรเป็น 1) ดังนั้น เมื่อนำสถานะของแต่ละโหนดมาพิจารณาจะได้ลำดับการแก้ไขปัญหามีลักษณะดังภาพที่ 2.6

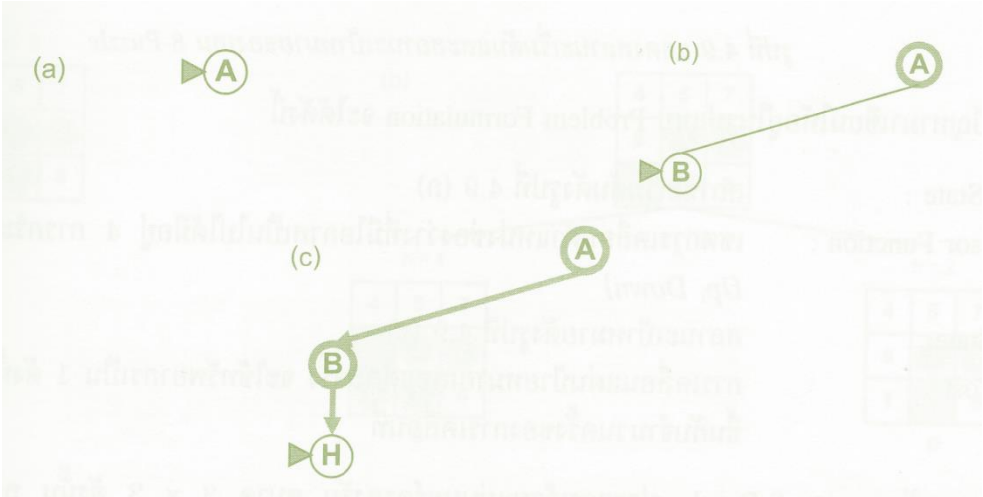


ภาพที่ 2.6 ลำดับการแก้ไขปัญหาเรื่องกล่อง
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 86)

2. การนำ Depth-First Search มาประยุกต์กับเกม

Depth-First Search เป็นวิธีการค้นหาแบบแนวลึก จะค้นหาทีละโหนดจากบนลงล่างหากกไม่พบให้ลงไปอีกหนึ่งระดับแล้วพิจารณาโหนดลูกที่อยู่ด้านซ้ายก่อน ทำเช่นนี้ไปเรื่อยๆ จนกว่า จะพบโหนดเป้าหมายที่องการหากพิจารณาจนถึงโหนดใบไม้ แล้วยังไม่พบ ให้ย้อนกลับขึ้นไปหนึ่งระดับไปยังโหนดพ่อแม่ที่ผ่านมาล่าสุด เพื่อมองหาเส้นทางไปยังโหนดที่เหลือ แล้วพิจารณาโหนดเช่นเดิม เมื่อพิจารณาแต่ละโหนดที่เป็นลูกของโหนดพ่อแม่ยังไม่พบโหนดที่ต้องการ ให้ย้อนกลับขึ้นไปอีกหนึ่งระดับและทำเช่นนี้ไปเรื่อยๆ จนกว่าจะพบโหนดที่ต้องการ โดยจะนำหลักการดังกล่าวมาใช้กับตัวอย่างเกมปัญหาเรื่องกล่อง

ขั้นตอนการค้นหาโหนดเป้าหมาย โดยใชข้วิธีการค้นหาแบบ Depth-First Search มีลักษณะดังภาพที่ 2.7



ภาพที่ 2.7 ขั้นตอนการค้นหาโหนดเป้าหมายด้วยวิธี Depth-First Search
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 87)

จากภาพที่ 2.7 พบว่าการค้นหาสามารถค้นพบสถานะเป้าหมายได้ โดยเส้นทางการค้นหาที่ได้คือ โหนด A, โหนด B และโหนด H ตามลำดับ ดังนั้น เมื่อนำสถานะแต่ละโหนดมาพิจารณาจะได้ลำดับการแก้ไขปัญหาเป็นลักษณะเหมือนกับภาพที่ 2.6

เกมกับการประยุกต์ด้วย Heuristic Search Technique

ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. (2552: 87-93) อธิบายไว้ว่า การประยุกต์โดยใช้เทคนิคการค้นหาแบบ Heuristic Search Technique ร่วมกับเกมต่างๆ เป็นเทคนิคการค้นหาแบบมีข้อมูลมาประกอบการพิจารณา โดยผู้เขียนโปรแกรมจะกำหนดข้อมูลดังกล่าวขึ้นมาเองตัวอย่างอัลกอริทึมที่นำมาประยุกต์ใช้กับเกมหัวข้อนี้คือ Greedy Best First Search และ A* Search

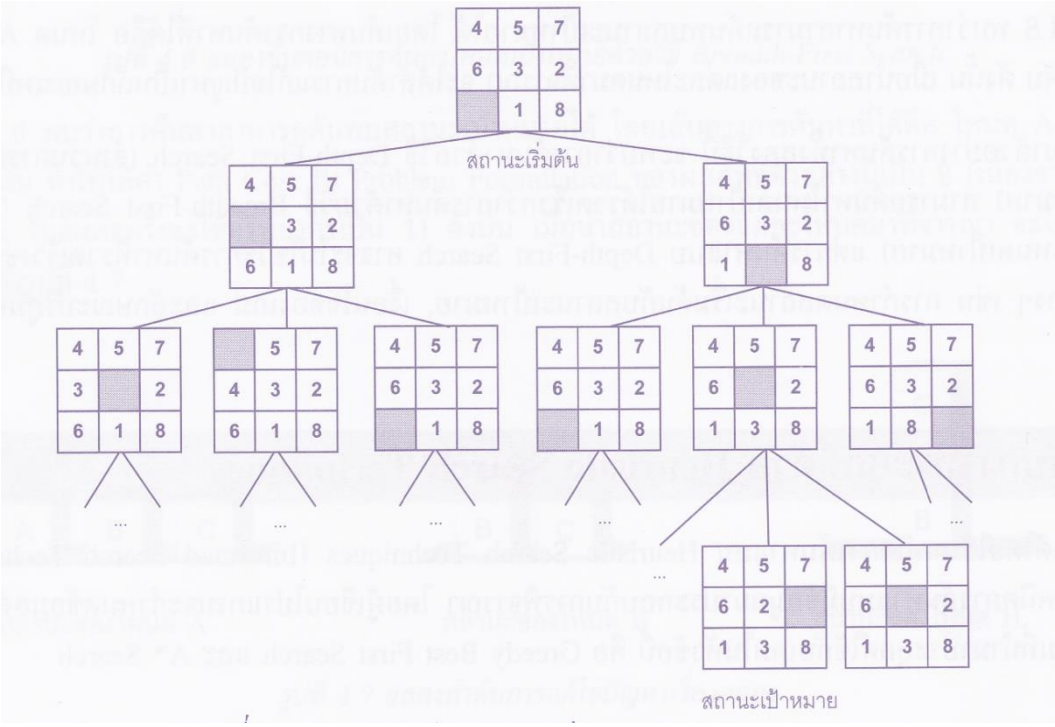
ตัวอย่างของเกมที่นำมาใช้พิจารณา คือ เกม 8-Puzzle ประกอบด้วยแผ่นบอร์ดรองรับขนาด 3x3 ภายในบรรจุแผ่นป้ายขนาด 1x1 ที่มีหมายเลขกำหนดตั้งแต่ “1” ถึง “8” และมี 1 ช่องว่าง แผ่นป้ายที่อยู่ติดกับช่องว่างสามารถเคลื่อนมาแทนที่ได้ ปัญหาของเกม 8-Puzzle คือ เลื่อนตำแหน่งแผ่นป้ายหมายเลขสถานะเริ่มต้นดังภาพที่ 2.8 ให้อยู่ในรูปแบบสถานะเป้าหมาย

ภาพที่ 2.8 สถานะเริ่มต้นและสถานะเป้าหมายของเกม 8-Puzzle
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 88)

หากนำปัญหามาเขียนอยู่ในรูปแบบ Problem Formulation จะได้ดังนี้

Inittial State:	สถานะเริ่มต้น ดังภาพที่ 2.8
Successor Function:	เซตการเคลื่อนตำแหน่งช่องว่างที่มีโอกาสเป็นไปได้มีอยู่ 4 การกระทำ คือ {Left, Right, Up, Down}
Goal State:	สถานะเป้าหมายดังภาพที่ 2.8
Path Cost:	การเคลื่อนแผ่นป้ายหมายเลขในแต่ละครั้งจะใช้ทรัพยากรเป็น 1 ดังนั้นPath Cost ที่จะใช้ขึ้นอยู่กับจำนวนครั้งของการเคลื่อนที่

หากพิจารณาปัญหาเกม 8-Puzzle ประกอบด้วยแผ่นบอร์ดรองรับขนาด 3x3 ดังนั้นสถานะที่เกิดขึ้นได้ทั้งหมด (3x3)! = 362,880 สถานะ สามารถนำมาสร้างปริภูมิสถานะได้ดังภาพที่ 2.9

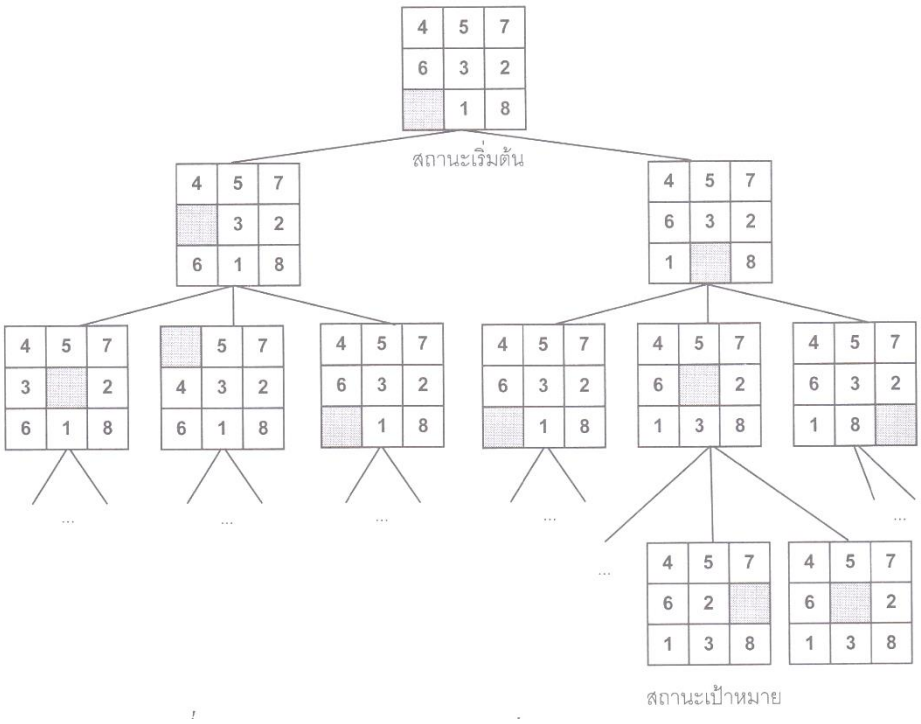


ภาพที่ 2.9 ปริภูมิสถานะบางส่วนของเกม 8-Puzzle
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 88)

1. การนำ Greedy Best First Search มาประยุกต์กับเกม

Greedy Best First Search เป็นวิธีการเลือกเส้นทางที่ดีที่สุดก่อน เพื่อค้นหาเป้าหมายได้อย่างรวดเร็ว โดยพิจารณาเลือกเส้นทางที่เป็นไปได้ และเส้นทางนั้นจะต้องใช้ทรัพยากรตั้งแต่ตำแหน่งที่พิจารณาจนถึงตำแหน่งเป้าหมาย (ค่า Heuristic: *h(n)*) น้อยที่สุด โดยต้องพิจารณาทุกตำแหน่งที่มองเห็น (ทุกตำแหน่งของโหนดใบไม้ขณะนั้น) ว่าตำแหน่งใดใช้ทรัพยากรน้อยที่สุด ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะพบเป้าหมายที่ต้องการ โดยจะนำหลักการดังกล่าวมาใช้กับตัวอย่างเกม 8-Puzzle

กำหนดให้จำนวนแผ่นป้ายของสถานะปัจจุบันที่อยู่ผิดตำแหน่งจากสถานะเป้าหมายเป็น ค่า Heuristic: *h(n)* ดังนั้นขั้นตอนการค้นหาสถานะเป้าหมาย โดยใช้วิธีการค้นหาแบบ Greedy Best First Search จะมีลักษณะดังภาพที่ 2.10 (กำหนดให้  แทนแผ่นป้ายที่มีอยู่เดิม)



ภาพที่ 2.10 ขั้นตอนการค้นหาสถานะเป้าหมายด้วยวิธี Greedy Best First Search
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 89)

จากภาพที่ 2.10 พบว่า การค้นหาด้วยวิธี Greedy Best First Search สามารถค้นพบสถานะเป้าหมายได้ขั้นตอนการค้นหามีดังนี้

1) เริ่มค้นหาที่สถานะ A (สถานะเริ่มต้น) พบว่ามีค่า *h*(A) เป็น 3
2) สถานะที่สามารถเกิดขึ้นได้ จากสถานะ A มีอยู่ 2 สถานะ คือ
   - สถานะ B: เลื่อนช่องว่างขึ้น ทำให้สถานะนี้มีค่า *h*(B) = 4
   - สถานะ C: เลื่อนช่องว่างไปทางขวา ทำให้สถานะนี้มีค่า *h*(C) = 2

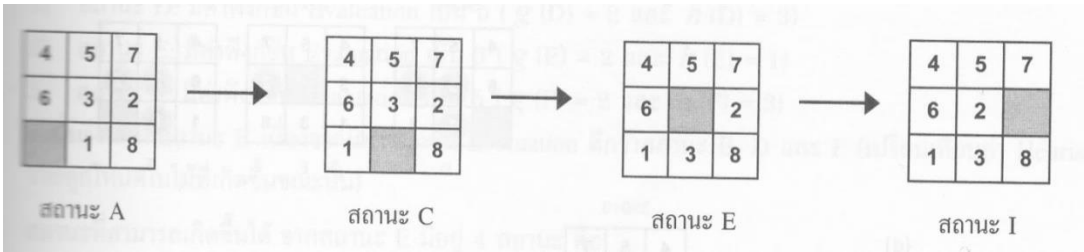
   ดังนั้นจึงเลือกสถานะ C เนื่องจากมีค่า Heuristic ดีกว่าสถานะ B
3) สถานะที่สามารถเกิดขึ้นได้ จากสถานะ C มีอยู่ 3 สถานะ คือ
   - สถานะ D: เลื่อนช่องว่างไปทางซ้าย ทำให้สถานะนี้มีค่า *h*(D) = 3
   - สถานะ E: เลื่อนช่องว่างขึ้น ทำให้สถานะนี้มีค่า *h*(E) = 1
   - สถานะ F: เลื่อนช่องว่างไปทางขวา ทำให้สถานะนี้มีค่า *h*(F) = 3

   ดังนั้นจึงเลือกสถานะ E เนื่องจากมีค่า Heuristic ดีกว่าสถานะ B,D และ F (เปรียบเทียบค่า Heuristic จากทุกโหนดใบไม้ที่เกิดขึ้นขณะนั้น)

4) สถานะที่สามารถเกิดขึ้นได้ จากสถานะ E มีอยู่ 4 สถานะ คือ
   - สถานะ G: เลื่อนช่องว่างไปทางซ้าย ทำให้สถานะนี้มีค่า *h*(G) = 2
   - สถานะ H: เลื่อนช่องว่างขึ้น ทำให้สถานะนี้มีค่า *h*(H) = 2
   - สถานะ I: เลื่อนช่องว่างไปทางขวา ทำให้สถานะนี้มีค่า *h*(I) = 0
   - สถานะ J: เลื่อนช่องว่างลง ทำให้สถานะนี้มีค่า h(J) = 2

   ดังนั้นจึงเลือกสถานะ I เนื่องจากเป็นสถานะเป้าหมายและมีค่า Heuristic ดีกว่าสถานะ B,D,F,G,H และ J

   ดังนั้นเส้นทางการค้นหาที่ได้คือ สถานะ A, สถานะ C สถานะ E และสถานะ I ตามลำดับ ทำให้ได้ค่า Path Cost ใน Problem Formulation ของผลลัพธ์ตัวอย่างนี้เป็น 3 (เนื่องจากการเคลื่อนย้ายแผ่นป้ายหมายเลขแต่ละครั้งจะใช้ทรัพยากรเป็น 1) จะได้ลำดับการแก้ไขปัญหามีลักษณะดังภาพที่ 2.11

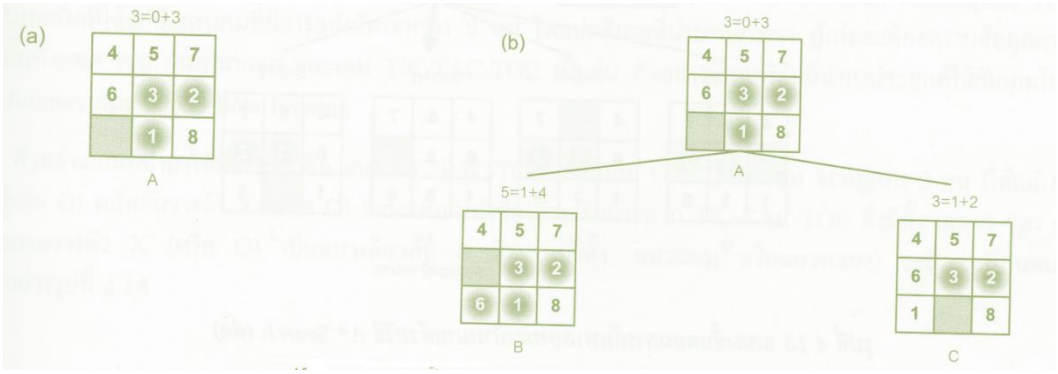


ภาพที่ 2.11 ลำดับการแก้ไขปัญหาเกม 8-Puzzle
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 91)

2. การนำ A* Search มาประยุกต์กับเกม

A* Star Search เป็นการค้นหาที่พิจารณาถึงค่า Heuristic (*h(n)*) รวมกับค่าของทรัพยากรที่ใช้จริงตั้งแต่โหนดเริ่มต้นจนถึงโหนด n หรือโหนดที่กำลังพิจารณาขณะนั้น (g*(n)*) เพื่อให้ๆได้ เป็นข้อมูลการประมาณค่าใช้จ่ายทั้งหมด หรือที่เรียกว่าค่าฟังก์ชัน Evaluation (*f(n)*) ที่สามารถนำข้อมูลนี้ไปใช้ตัดสินใจเลือกเส้นทางไปสู่เป้าหมายได้ รูปแบบสมการของ A* Search คือ *f(n)=g(n)+h(n)* โดยจะนำหลักการนี้มาใช้กับตัวอย่างเกม 8-Puzzle

กำหนดให้วิธีการหาค่า Heuristic (*h(n)*) เหมือนกับหัวข้อที่ 1 และค่า g(n) แทนการเคลื่อนแต่ละครั้งจะใช้ทรัพยากรทีละ 1 ดังนั้น ขั้นตอนการค้นหาสถานะเป้าหมายโดยใช้วิธีการค้รหาแบบ A*Search มีลักษณะดังภาพที่ 2.12 กำหนดให้  แทนแผ่นป้ายที่มีวางอยู่ผิดตำแหน่ง



ภาพที่ 2.12 ขั้นตอนการค้นหาสถานะเป้าหมายด้วยวิธี A*Search
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 91-92)

จากภาพที่ 2.12 พบว่าการค้นหาด้วยวิธี A*Search สามารถค้นพบสถานะเป้าหมายได้ ขั้นตอนการค้นหามีดังนี้

1) เริ่มค้นหาที่สถานะ A (สถานะเริ่มต้น) มีค่าฟังก์ชัน Evaluation เป็น 3 (g(A)=0) (ยังไม่มีการเปลี่ยนสถานะ)และค่า *h*(A) =3 (แทนจำนวนของแผ่นป้ายของสถานะปัจจุบันที่อยู่ผิดตำแหน่งจากสถานะเป้าหมาย)
2) สถานะที่สามารถเกิดขึ้นได้ จากสถานะ A มีอยู่ 2 สถานะ คือ
   - สถานะ B มีค่าฟังก์ชัน Evaluation เป็น 5 (g(B)=1) และค่า *h*(B) =4
   - สถานะ C มีค่าฟังก์ชัน Evaluation เป็น 3 (g(C)=1) และค่า *h*(C) =2

   ดังนั้นจึงเลือกสถานะ C เนื่องจากค่าฟังก์ชัน Evaluation ดีกว่าสถานะ B
3) สถานะที่สามารถเกิดขึ้นได้ จากสถานะ C มีอยู่ 3 สถานะ คือ
   - สถานะ D มีค่าฟังก์ชัน Evaluation เป็น 5 (g(D)=2) และค่า *h*(D) =3
   - สถานะ E มีค่าฟังก์ชัน Evaluation เป็น 3 (g(E)=2) และค่า *h*(E) =1
   - สถานะ F มีค่าฟังก์ชัน Evaluation เป็น 5 (g(F)=2) และค่า *h*(F) =3

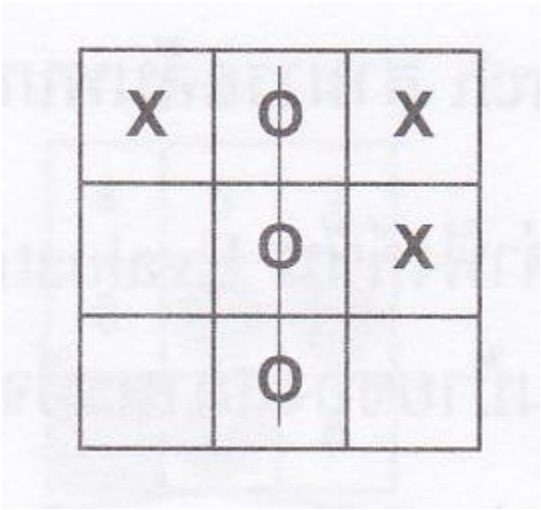
   ดังนั้นจึงเลือกสถานะ E เนื่องจากค่าฟังก์ชัน Evaluation ดีกว่าสถานะ B,D และ F (เปรียบเทียบค่า Heuristic จากทุกโหนดใบไม้ที่เกิดขึ้นขณะนั้น)
4) สถานะที่สามารถเกิดขึ้นได้ จากสถานะ E มีอยู่ 4 สถานะ คือ
   - สถานะ G มีค่าฟังก์ชัน Evaluation เป็น 5 (g(G)=3) และค่า *h*(G) =2
   - สถานะ H มีค่าฟังก์ชัน Evaluation เป็น 5 (g(H)=3) และค่า *h*(H) =2
   - สถานะ I มีค่าฟังก์ชัน Evaluation เป็น3 (g(I)=3) และค่า *h*(I) =0
   - สถานะ J มีค่าฟังก์ชัน Evaluation เป็น 5 (g(J)=3) และค่า *h*(J) =2

   ดังนั้นจึงเลือกสถานะ I เนื่องจากค่าฟังก์ชัน Evaluation ดีกว่าสถานะ B,D,F,H และ J (เปรียบเทียบค่า Heuristic จากทุกโหนดใบไม้ที่เกิดขึ้นขณะนั้น)

ดังนั้นเส้นทางการค้นหาที่ได้คือ สถานะ A สถานะ C สถานะ E และสถานะ I ตามลำดับ จะได้ลำดับการแก้ไขปัญหาลักษณะเหมือนกับภาพที่ 2.11

เกมกับการประยุกต์ด้วย Adversarial Search Techniques

ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. (2552: 93-37) อธิบายไว้ว่า การประยุกต์ใช้เทคนิคการค้นหาแบบ Adversarial Search Techniques ร่วมกับเกมจะเป็นเทคนิคที่ใช้แก้ปัญหาเกมที่มีการแข่งขันระหว่าง 2 คน โดยปกติเกมที่นำมาพิจารณา ผู้เล่นจะต้องทราบข้อมูลภายในเกมขณะนั้นทั้งหมด เช่น เกมหมากรุก และเกม TIC-TAC-TOE หรือเกม OX จะมีผู้เล่น 2 คน ได้แก่ผู้เล่น X และผู้เล่น O ผลัดกันวางตัว X หรือ ตัว O ลงบนตำแหน่งช่องว่างของกระดาน TIC-TAC-TOE ซึ่งมีทั้งหมด 9 ช่อง ถ้าผู้เล่นใดสามารถวางตัวX หรือ ตัว O เป็นแนวเดียวกัน 3 ตัว (แนวตั้ง,แนวนอน หรือ แนวทแยง)จะถือว่าผู้เล่นคนนั้นชนะ ดังตัวอย่างภาพที่ 2.13

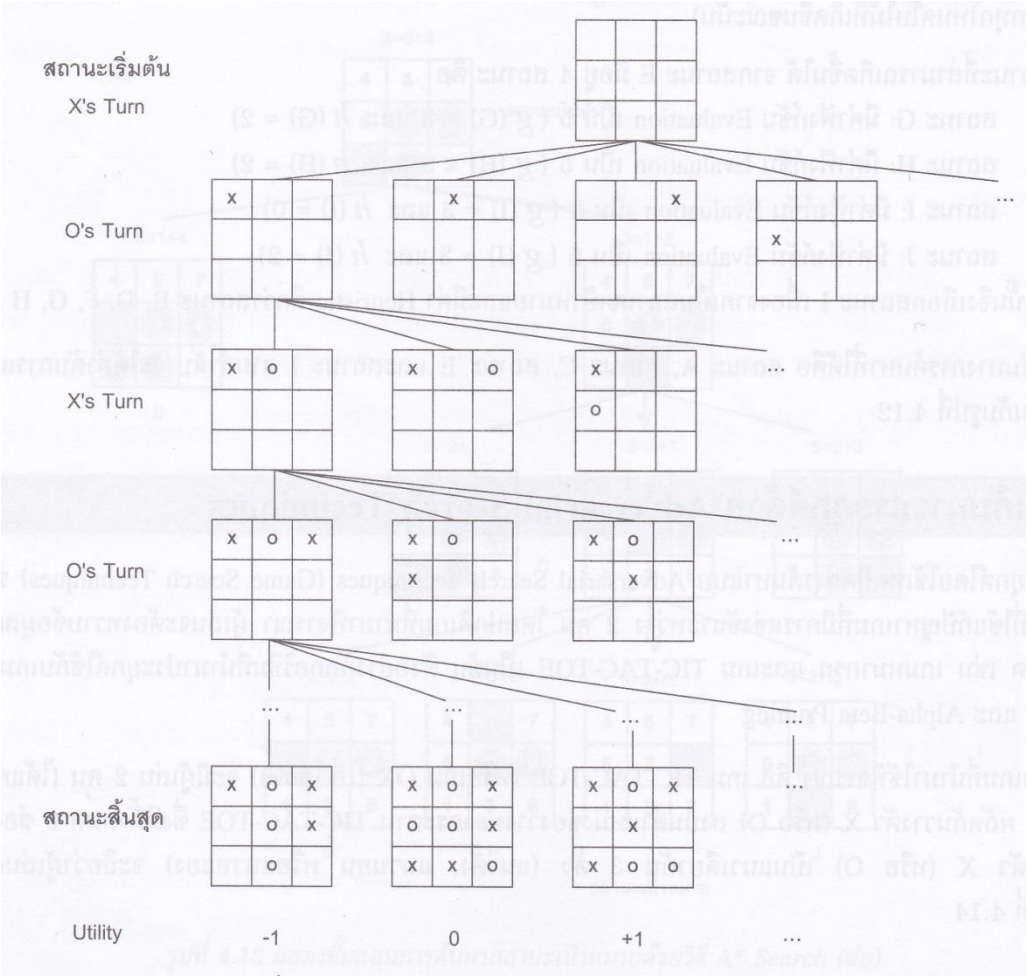


ภาพที่ 2.13 ตัวอย่างผู้เล่น O เป็นผู้ชนะในเกม
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 94)

หากนำปัญหามาเขียนในรูปแบบ Problem Formulation จะได้ดังนี้

Inittial State:	ไม่มีตัว X หรือตัว O อยู่บนกระดาน TIC-TAC-TOE
Successor Function:	ผลัดกันวาง ตัวX หรือ ตัว O ทีละตัวลงบนตำแหน่งช่องว่างของกระดาน TIC-TAC-TOE
Goal State:	ตัวX หรือ ตัว O ถูกวางเป็นแนวเดียวกัน 3 ตัว (แนวตั้ง,แนวนอน หรือ แนวทแยง)
Path Cost:	จำนวนการผลัดกันเล่น

หากพิจารณาปัญหาของเกม TIC-TAC-TOE จะพบว่าช่องกระดานมี 9 ช่อง แต่ละช่องสามารถใส่ “X”, “O” หรือไม่ใส่เลยก็ได้ ดังนั้น สถานะที่เกิดขึ้นได้ทั้งหมดมี 3^9 = 19,683 สถานะ สามารถสร้างปริภูมิสถานะได้ ดังภาพที่ 2.14

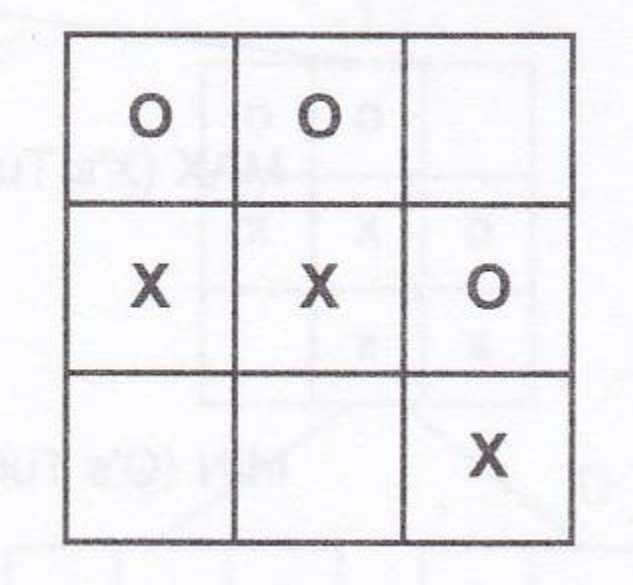


ภาพที่ 2.14 ปริภูมิสถานะบางส่วนของเกม TIC-TAC-TOE
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 94)

การนำ Minimax มาประยุกต์กับเกม

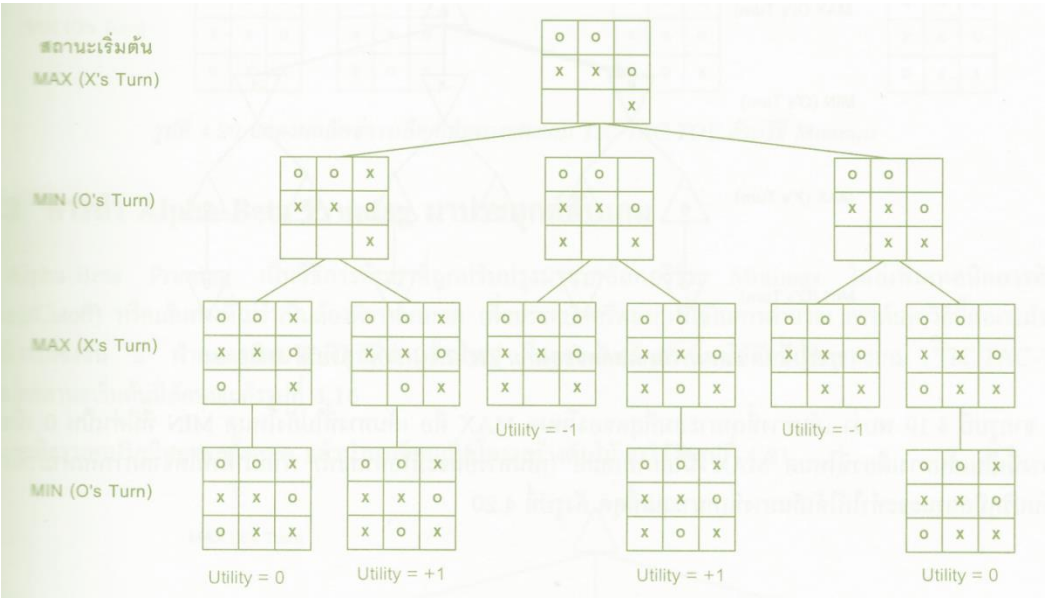
Minimax มีลักษณะการค้นหาแบบ Depth-First Search โยพิจารณาค่าต่างๆ จากโหนดใบไม้ และลือกค่าที่เหมาะสมขึ้นมายังโหนดด้านบน โดยจะทำเช่นนี้ไปจนกว่าโหนดรากจะได้รับค่า โดยจะนำหลักการนี้มาใช้กับเกม TIC-TAC-TOE

เกม TIC-TAC-TOE จะมีขนาดปริภูมิสถานะค่อนข้างใหญ่ ดังภาพ 2.14 จึงกำหนดตัวอย่างให้สถานะเริ่มต้นของเกมถูกเล่นมาสักระยะหนึ่งแล้วเพื่อความสะดวกต่อการทำความเข้าใจ โดยมีลักษณะดังรูป และกำหนดให้ผู้เล่นตัว “X” เป็นผู้เริ่มเล่น



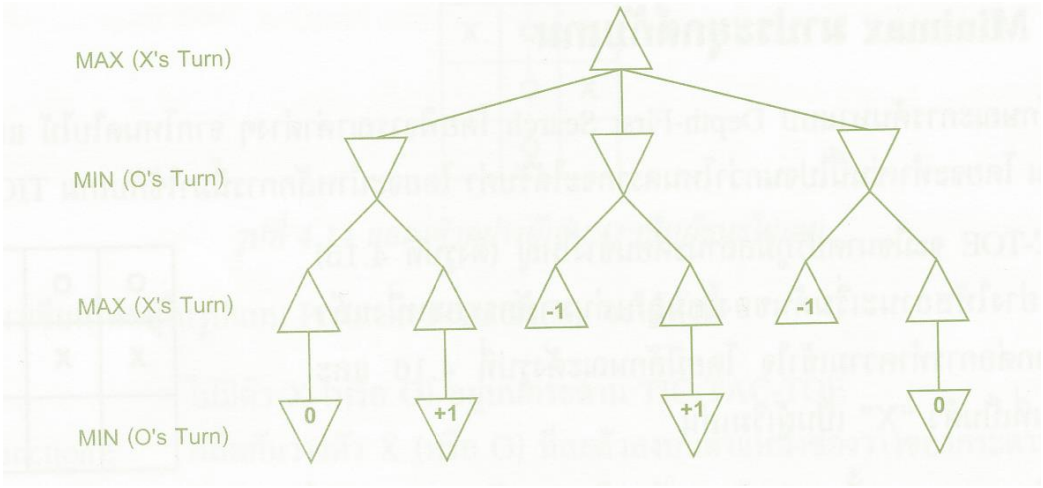
ภาพที่ 2.15 สถานะเริ่มต้นของเกม TIC-TAC-TOE
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 94)

พิจารณาความเป็นไปได้ของสถานะต่างๆ ที่อาจเกิดขึ้นในปัญหานี้ จะได้ปริภูมิสถานะดังภาพที่ 2.16



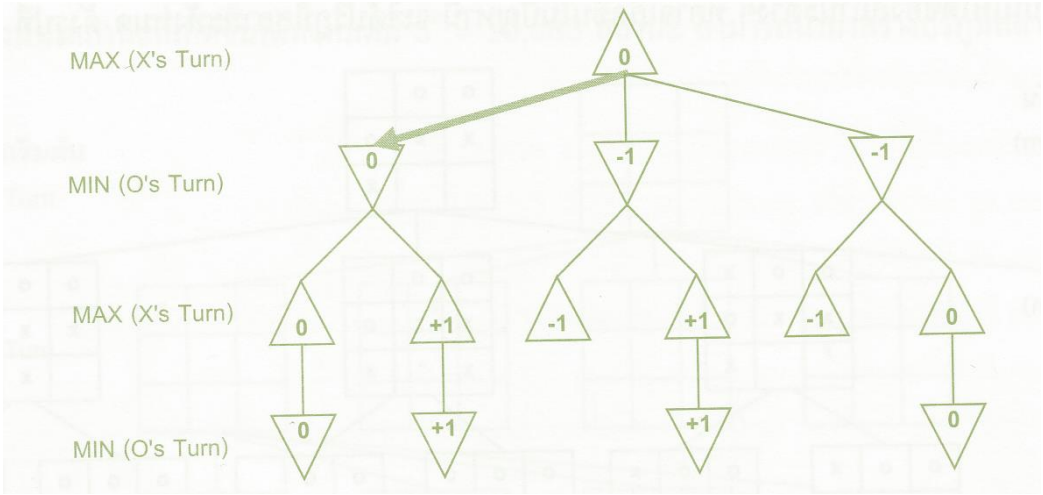
ภาพที่ 2.16 ปริภูมิสถานะของเกม TIC-TAC-TOE
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 95)

จากภาพ 2.17 Utility Function ของแต่ละ Terminal State ในปริภูมิสถานะจะประกอบด้วย -1 หมายถึง ผู้เล่นเป็นตัว “O” ชนะ, +1 หมายถึง ผู้เล่นเป็นตัว “X” ชนะ และ 0 หมายถึง เสมอ สามารถนำมาเขียนในรูปแบบโครงสร้างของต้นไม้ได้ โดยนำค่า Utility ต่างๆ ที่ได้จากปริภูมิสถานะของปัญหา มาใช้ในการพิจารณาการค้นหาเส้นทางแบบ Minimax ดังภาพที่ 2.17



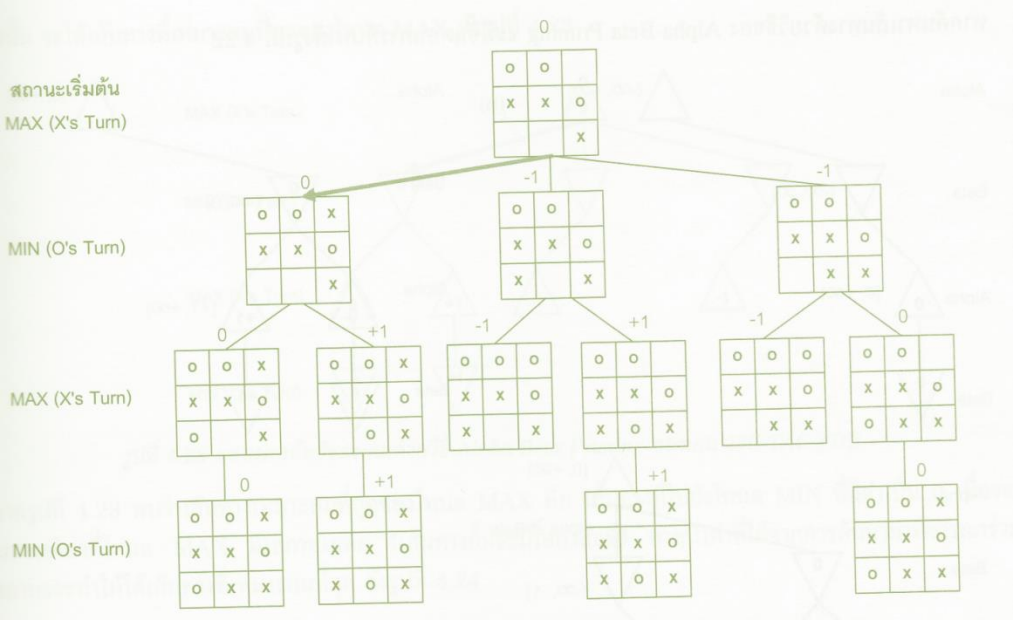
ภาพที่ 2.17 โครงสร้างต้นไม้จากปริภูมิสถานะของเกม TIC-TAC-TOE
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 96)

หากใช้วิธี Minimax จะสามารถค้นหาเส้นทางที่เหมาะสมที่สุดของโหนด MAX ได้ดังภาพที่ 2.18



ภาพที่ 2.18 การหาผลลัพธ์ของเกม TIC-TAC-TOE ด้วยวิธี Minimax
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 96)

จากภาพที่ 2.18 พบว่า เส้นทางที่เหมาะสมที่สุดของโหนด MAX คือ เส้นทางที่ไปยังโหนด MIN ที่มีค่าเป็น 0 เนื่องจากเส้นทางนี้เป็นเส้นทางเดียวที่โหนด MAX มีโอกาสเสมอ (เส้นทางอื่นจะมีโอกาสแพ้) หากนำค่ามราได้จากการค้นหามาพิจารณาร่วมกับปริภูมิสถานะจะทำให้ได้เส้นทางที่เหมาะสมที่สุด ดังภาพที่ 2.19



ภาพที่ 2.19 ผลลัพธ์การเลือกเส้นทางของเกม TIC-TAC-TOE ด้วยวิธี Minimax
ที่มา: (ณัฐพงษ์ วารีประเสริฐ และ ณรงค์ ล่ำดี. 2552: 97)

สรุป

ลำดับขั้นตอนการแก้ปัญหาเกมคือ จะต้องวิเคราะห์เกมให้อยู่ในรูปแบบ Problem Formulation เพื่อนำไปสร้างปริภูมิสถานะแล้วจึงนำเทคนิคการค้นหาที่เหมาะสมกับเกมมาช่วยค้นหาเป้าหมายที่ต้องการ การประยุกต์โดยใช้เทคนิคการค้นหาแบบ Blind Search Technique ร่วมกับเกมต่างๆ เป็นเทคนิคการค้นหาแบบไม่มีข้อมูลมาประกอบการพิจารณา ตัวอย่างโครงสร้างที่นำมาใช้อธิบายเป็นรูปแบบต้นไม้ ตัวอย่างอัลกอริทึมที่นำมาใช้คือ Breadth-First Search และ Depth-First Search การประยุกต์ใช้เทคนิคการค้นหาแบบ Adversarial Search Techniques ร่วมกับเกมจะเป็นเทคนิคที่ใช้แก้ปัญหาเกมที่มีการแข่งขันระหว่าง 2 คน โดยปกติเกมที่นำมาพิจารณาผู้เล่นจะต้องทราบข้อมูลภายในเกมขณะนั้นทั้งหมด และการประยุกต์โดยใช้เทคนิคการค้นหาแบบ Heuristic Search Technique ร่วมกับเกมต่างๆ เป็นเทคนิคการค้นหาแบบมีข้อมูลมาประกอบการพิจารณา โดยผู้เขียนโปรแกรมจะกำหนดข้อมูลดังกล่าวขึ้นมาเองตัวอย่างอัลกอริทึมที่นำมาประยุกต์ใช้กับเกมหัวข้อนี้คือ Greedy Best First Search และ A* Search

แบบฝึกหัด

1. จงยกตัวอย่างปัญญาประดิษฐ์ที่นำมาใช้กับการพัฒนาเกม
2. การประยุกต์โดยใช้เทคนิคการค้นหาแบบ Blind Search Technique
3. เทคนิคการค้นหาแบบใดเป็นเทคนิคการค้นหาแบบไม่มีข้อมูลมาประกอบการพิจารณา
4. จงอธิบายหลักการเทคนิค Breadth-First Search ที่นำมาประยุกต์ใช้กับเกม
5. จงอธิบายหลักการเทคนิค Depth-First Search ที่นำมาประยุกต์ใช้กับเกม
6. เทคนิคการค้นหาแบบใดเป็นเทคนิคการค้นหาแบบมีข้อมูลมาประกอบการพิจารณา
7. จงอธิบายหลักการเทคนิค Greedy Best First Search ที่นำมาประยุกต์ใช้กับเกม
8. จงอธิบายหลักการเทคนิค A* Search มาประยุกต์กับเกม ที่นำมาประยุกต์ใช้กับเกม
9. เทคนิคการค้นหาแบบใดเป็นเทคนิคที่ใช้แก้ปัญหาเกมที่มีการแข่งขันระหว่าง 2 คน
10. จงอธิบายหลักการเทคนิคการนำ Minimax ที่นำมาประยุกต์ใช้กับเกม

เอกสารอ้างอิง

ณัฐพงษ์ วารีประเสริฐ และณรงค์ ล่ำดี. (2552). *ปัญญาประดิษฐ์ (Artificial Intelligence).* กรุงเทพฯ: เคทีพี คอมพ์แอนด์ คอนซัลท์,

ศศลักษณ์ ทองขาว. (2549). *ปัญญาประดิษฐ์ (Artificial Intelligence).* กรุงเทพฯ: สำนักพิมพ์ภาพพิมพ์.