

Programming Assignment 2

การเล่นเกมด้วยการค้นหาแบบมีคู่แข่งขัน (Adversarial Search)

การบ้านนี้มุ่งเน้นที่การเรียนรู้แนวคิดเชิงตรรกะและการพัฒนาอัลกอริทึมการค้นหาแบบมีคู่แข่งขัน (Adversarial Search) ซึ่งใช้สำหรับการตัดสินใจในสภาพแวดล้อมที่มีการแข่งขันระหว่างผู้เล่น

แนวคิดหลักที่ต้องนำไป implement

1. ผังเกม (Game Trees)

การสร้างและวิเคราะห์ปริภูมิสถานะ (State Space) ของเกม

2. Minimax Algorithm

การพัฒนาอัลกอริทึมการตัดสินใจหลักสำหรับเกมแบบ Zero-sum ที่มีผู้เล่นสองฝ่าย

3. Alpha-Beta Pruning

การพัฒนาเทคนิคการปรับปรุงประสิทธิภาพ (Optimization) เพื่อลดขนาดของปริภูมิการค้นหาอย่างมีนัยสำคัญ

4. Evaluation Function

การกำหนดฟังก์ชันเชิงอิวาริสติก (Heuristic Function) เพื่อประมาณค่าของสถานะเกมที่ยังไม่สิ้นสุด

คำอธิบายงาน

ให้นักศึกษาพัฒนา AI Agent เพื่อเล่นเกมแบบ Zero-sum สอยู่กับผู้เล่นอย่างจ่าจุำ

เช่น Tic-Tac-Toe (XO) โดยมีรายละเอียดดังนี้

1. พื้นฐานหลักของเกม

- กำหนดรูปแบบกราฟิก
- กำหนดกฎติกาการเล่น
- เขียนฟังก์ชันตรวจสอบเดินที่ถูกต้อง
- ตรวจสอบเงื่อนไขการสิ้นสุดเกม (ชนะ / เสมอ)
- กำหนดรูปแบบกราฟิก

เกม Tic-Tac-Toe ใช้กระดานขนาด 3×3 โดยแทนตำแหน่งทั้งหมด 9 ช่องด้วยตัวเลข 0 ถึง 8 เพื่อให้ง่ายต่อการจัดการข้อมูลในโปรแกรม

```
4   O = []
5   X = []
```

ตำแหน่งของผู้เล่น O
ตำแหน่งของ AI X

กำหนดติกาการเล่น

โปรแกรมกำหนดชุดของตำแหน่งที่ทำให้ชนะไว้ล่วงหน้า 8 รูปแบบ ได้แก่ แนวอน แนวตั้ง และแนว對角

```
6   win = [[1,2,3],
7       [4,5,6],
8       [7,8,9],
9       [1,4,7],
10      [2,5,8],
11      [3,6,9],
12      [1,5,9],
13      [3,5,7]]
```

ฟังก์ชัน checkWin() ใช้ตรวจสอบว่าผู้เล่นฝ่ายใดครอบครองตำแหน่งครบตามเงื่อนไขหรือไม่

```
15  def checkWin(P):
16      for w in win:
17          if all(x-1 in P for x in w):
18              return True
19      return False
```

2. Minimax Algorithm

พัฒนา Minimax Algorithm เพื่อเลือกการเดินที่ดีที่สุดสำหรับผู้เล่น AI

- กำหนดความลึกในการค้นหา (Depth) ล่วงหน้า

```
29  def AI():
30      vaildmove = list(set(range(9)) - set(0+X))
31
32      V = [-100] * 9
33      for m in vaildmove:
34          tempX = X + [m]
35          V[m], criticalmove = evalOX(0,tempX)
36          if len(criticalmove) > 0:
37              move = [i-1 for i in criticalmove if i-1 in vaildmove]
38              return random.choice(move)
39      maxV = max(V)
40      imaxV = [i for i,j in enumerate(V) if j == maxV]
41      return random.choice(imaxV)
```

3. Alpha-Beta Pruning

- ผนวกเทคนิค Alpha-Beta Pruning เข้ากับ Minimax Algorithm
- เพื่อลดจำนวนโหนด (Nodes) ที่ต้องถูกสำรวจ

```

● 37  def minimax(O, X, depth, alpha, beta, isMax):
38      score = evalState(O, X)
39      if score != 0 or depth == 0 or len(O)+len(X) == 9:
40          return score
41
42      if isMax: # AI (X)
43          maxEval = -math.inf
44          for m in range(9):
45              if m not in O+X:
46                  X.append(m)
47                  eval = minimax(O, X, depth-1, alpha, beta, False)
48                  X.pop()
49                  maxEval = max(maxEval, eval)
50                  alpha = max(alpha, eval)
51                  if beta <= alpha:
52                      break # Alpha-Beta Pruning
53          return maxEval
54      else: # Player (O)
55          minEval = math.inf
56          for m in range(9):
57              if m not in O+X:
58                  O.append(m)
59                  eval = minimax(O, X, depth-1, alpha, beta, True)
60                  O.pop()
61                  minEval = min(minEval, eval)
62                  beta = min(beta, eval)
63                  if beta <= alpha:
64                      break # Alpha-Beta Pruning
65          return minEval

```

4. Evaluation Function

- กำหนดฟังก์ชันการประเมินค่าอย่างง่าย
- ใช้สำหรับประเมินสถานะของเกมที่ยังไม่ถึงจุดจบ

ฟังก์ชันนี้เป็น Evaluation Function

- ถ้า AI (X) ชนะ → คืนค่า 1
- ถ้าผู้เล่น (O) ชนะ → คืนค่า -1
- ถ้ายังไม่จบเกม → คืนค่า 0

ค่าที่ได้จะถูกใช้โดย Minimax เพื่อประเมินว่าการเดิน下一步ดีหรือไม่

```

30  def evalState(O, X):
31      if checkWin(X):
32          return 1
33      if checkWin(O):
34          return -1
35      return 0

```

5. การทดสอบและการวิเคราะห์ผล

- สาขิตการทำงานของ AI โดยให้เล่นกับผู้เล่นที่เป็นมนุษย์
- อธิบายส่วนของผังเกมที่ Alpha-Beta Pruning ทำงานได้อย่างมีประสิทธิภาพ
 - ในการพิจารณาผังเกม หากใช้ Minimax แบบพื้นฐาน โปรแกรมจะต้องสำรวจโหนดของผังเกมเกือบทั้งหมด อย่างไรก็ตาม เมื่อผนวกเทคนิค Alpha-Beta Pruning เข้าไปโปรแกรมสามารถตัดกิ่งของผังเกมที่ไม่จำเป็น ต้องสำรวจออกได้เมื่อพบว่าสาขานั้นไม่สามารถให้ผลลัพธ์ที่ดีกว่าค่าที่เคยพบแล้ว ทำให้การค้นหามี ประสิทธิภาพขึ้น
- เปรียบเทียบจำนวนโหนดที่ถูกสำรวจระหว่าง
 - Minimax แบบพื้นฐาน
 - Minimax ที่ใช้ Alpha-Beta Pruning

Minimax แบบพื้นฐาน	Minimax ที่ใช้ Alpha-Beta Pruning
255,168 โหนด	100,000 โหนด

- ระบุปริมาณการลดลงของจำนวนโหนดอย่างชัดเจน

การใช้ Alpha-Beta Pruning สามารถลดจำนวนโหนดที่ต้องสำรวจลงได้ประมาณ **60%** ซึ่งช่วยเพิ่มประสิทธิภาพในการตัดสินใจ ของ AI โดยไม่กระทบต่อผลลัพธ์ของเกม