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**

   การกำหนดฟังก์ชันเชิงฮิวริสติก (Heuristic Function) เพื่อประมาณค่าของสถานะเกมที่ยังไม่สิ้นสุด

คำอธิบายงาน

ให้นักศึกษาพัฒนา **AI Agent** เพื่อเล่นเกมแบบ Zero-sum สองผู้เล่นอย่างง่าย

เช่น **Tic-Tac-Toe (XO)** โดยมีรายละเอียดดังนี้

1. **พื้นฐานหลักของเกม**

- กำหนดรูปแบบกระดานเกม
- กำหนดกติกาการเล่น
- เขียนฟังก์ชันตรวจสอบการเดินที่ถูกต้อง
- ตรวจสอบเงื่อนไขการสิ้นสุดเกม (ชนะ / เสมอ)
- กำหนดรูปแบบกระดานเกม

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

เพื่อให้ง่ายต่อการจัดการข้อมูลในโปรแกรม

กำหนดกติกาการเล่น

โปรแกรมกำหนดชุดของตำแหน่งที่ทำให้ชนะไว้ล่วงหน้า 8 รูปแบบ ได้แก่ แนวนอน แนวตั้ง และแนวทแยง

```
win = [[1,2,3],
       [4,5,6],
       [7,8,9],
       [1,4,7],
       [2,5,8],
       [3,6,9],
       [1,5,9],
       [3,5,7]]
```

ฟังก์ชัน checkWin() ใช้ตรวจสอบว่าผู้เล่นฝ่ายใดครอบครองตำแหน่งครบตามเงื่อนไขหรือไม่

```
def checkWin(P):
    for w in win:
        if all(x-1 in P for x in w):
            return True
    return False
```

2. **Minimax Algorithm**

   พัฒนา Minimax Algorithm เพื่อเลือกการเดินที่ดีที่สุดสำหรับผู้เล่น AI

- กำหนดความลึกในการค้นหา (Depth) ล่วงหน้า

```
def AI():
    vaildmove = list(set(range(9)) - set(O+X))

    V = [-100] * 9
    for m in vaildmove:
        tempX = X + [m]
        V[m], criticalmove = evalOX(O,tempX)
        if len(criticalmove) > 0:
            move = [i-1 for i in criticalmove if i-1 in vaildmove]
            return random.choice(move)
    maxV = max(V)
    imaxV = [i for i,j in enumerate(V) if j ==  maxV]
    return random.choice(imaxV)
```

3. **Alpha-Beta Pruning**

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

```python
def minimax(O, X, depth, alpha, beta, isMax):
    score = evalState(O, X)
    if score != 0 or depth == 0 or len(O)+len(X) == 9:
        return score

    if isMax:  # AI (X)
        maxEval = -math.inf
        for m in range(9):
            if m not in O+X:
                X.append(m)
                eval = minimax(O, X, depth-1, alpha, beta, False)
                X.pop()
                maxEval = max(maxEval, eval)
                alpha = max(alpha, eval)
                if beta <= alpha:
                    break   # Alpha-Beta Pruning
        return maxEval
    else:      # Player (O)
        minEval = math.inf
        for m in range(9):
            if m not in O+X:
                O.append(m)
                eval = minimax(O, X, depth-1, alpha, beta, True)
                O.pop()
                minEval = min(minEval, eval)
                beta = min(beta, eval)
                if beta <= alpha:
                    break   # Alpha-Beta Pruning
        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 เพื่อประเมินว่าการเดินนั้นดีหรือไม่

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

5. **การทดสอบและการวิเคราะห์ผล**

- สาธิตการทำงานของ AI โดยให้เล่นกับผู้เล่นที่เป็นมนุษย์
- อธิบายส่วนของผังเกมที่ Alpha-Beta Pruning ทำงานได้อย่างมีประสิทธิภาพ
    - ในการพิจารณาผังเกม หากใช้ Minimax แบบพื้นฐาน

      โปรแกรมจะต้องสำรวจโหนดของผังเกมเกือบทั้งหมด อย่างไรก็ตาม เมื่อผนวกเทคนิค **Alpha-Beta Pruning**

      เข้าไปโปรแกรมสามารถตัดกิ่งของผังเกมที่ไม่จำเป็นต้องสำรวจออกได้เมื่อพบว่าสาขานั้นไม่สามารถให้ผลลัพธ์ที่ดีกว่าค่าที่เคยพบแล้ว ทำให้การค้นหามีประสิทธิภาพขึ้น

- 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%**

ซึ่งช่วยเพิ่มประสิทธิภาพในการตัดสินใจของ AI โดยไม่กระทบต่อผลลัพธ์ของเกม

รายชื่อสมาชิก

1. 66200028 เกียรติยศ จันทร์เพ็ง

2. 66200250 ศตายุ ใสสว่าง

3. 66200357 ทินบดี สุมังคละสมบัติ