卒業研究報告書

題目

pythonで作るコネクト4

指導教員

石水 隆　講師

報告者

15-1-037-0039

中塚　恭右

近畿大学理工学部情報学科

令和3年2月1日提出

概要

近年,人工知能は様々な分野において急速に導入が進められている。人工知能の中でもとりわけ注目されているのがディープラーニングによる機械学習である．囲碁や将棋など、以前は莫大な手数を推測することが難しく,従来の探索を用いたゲーム AI はアマチュア程度のレベルに留まっていた．しかし現在ではディープラーニングの登場により,莫大な情報を処理できる事で推測が容易になり,将棋や囲碁の AI が間界のプロに何度も勝利を収めている.一方より単純なゲームでは，従来の探索による方法でも充分強い AI を作ることができる．つまり，対象の複雑さにより探索を用いた場合とディープラーニング場合とで，どちらが効率の良い AI となるかは変わる.

そこで本研究ではコネクト 4 と呼ばれる重力付き四目並べと言うボードゲームを題材に python 言語で AI を開発し，従来の探索を用いたコネクト 4 の AI との性能差を検証する.

目次

1. 序論

1.1. 本研究の背景

近年,人工知能の技術が急速に進化していく中で機械学習 と呼ばれる人間の学習能力をコンピュータに実現させる技術のうち,ディープラーニング(深層学習)と呼ばれる手法が注目されている.ディープラーニングは画像処理で用いられたり，将棋やチェスなどにも用いられたりしている.

1.2. ディープラーニング

ディープラーニングは(深層学習)[1]は,機械学習と呼ばれる手法の一つである.ディープラーニングはいろいろな分野において成果を上げている.その中でも画像認識では写真に映る顔を認識し感情を読み取る技術や似たような特徴を持つグループを作ったり，照合したりすることができる[1]. また，製造業などでは、本来人間が「傷」や「故障」を目で見ていたものを同等の技術を有する画像処理により，高い品質管理が行えるようになった[1].

1.3. ディープラーニングによるゲーム AI

ディープラーニングは,先述の通り画像処理等の分野に使われているが,囲碁や将棋といったゲームにも深く関わっている.2016 年 3 月に韓国のトップレベルの囲碁棋士に，ディープラーニングを用いた「AlphaGo（アルファ碁)」[10]が勝利(4 勝 1 敗)した.AlphaGo は元々「教師あり学習」という盤面状況と打ち手のデータを学習する手と言う手法で膨大なデータから学習していたが学習不足であった.そこで,次に強化学習を導入した.強化学習とは,試行錯誤をする中で,報酬や罰を与えることで,その高度を強化し,自らその行動を学習するという学習方法である.将棋では,山本一成が 2008 年に開発をした ponaza[20]が挙げられる.人間界のプロと対局する将棋電王戦では ponaza は第 2 回(2013 年),第 3 回(2014 年),FINAL(2015 年)に登場し,いずれも勝利を収めている.また,2016 年と 2017 年の電王戦でも勝利を収めている[19].

1.4. コネクト 4

「コネクト４」とは縦 6 マス,横 7 マスのボードと各 21 枚の 2 色の石を用いた 2 人用ゲームである[4]。コネクト 4 のボードは縦に設置され，プレイヤーはボードの上部から石を投入することができる．投入された石はボードの一番下のマスか，すでに置かれた石の上のマスに配置される.各プレイヤーは交互に石を投入していき，縦・横・斜め・のいずれか先に 4 つの石を並べた方の勝利となる.

コネクト 4 は J.D.Allen により先手必勝出あることが示されており,双方最善手を打った場合,41 手で先手の勝ちとなる[4].また,コネクト 4 の既存のＡＩとしては,$\alpha\beta$ 法により着手選択する Connect 4 Solver がある[12]．図 1 にコネクト 4 の最善手を示す.

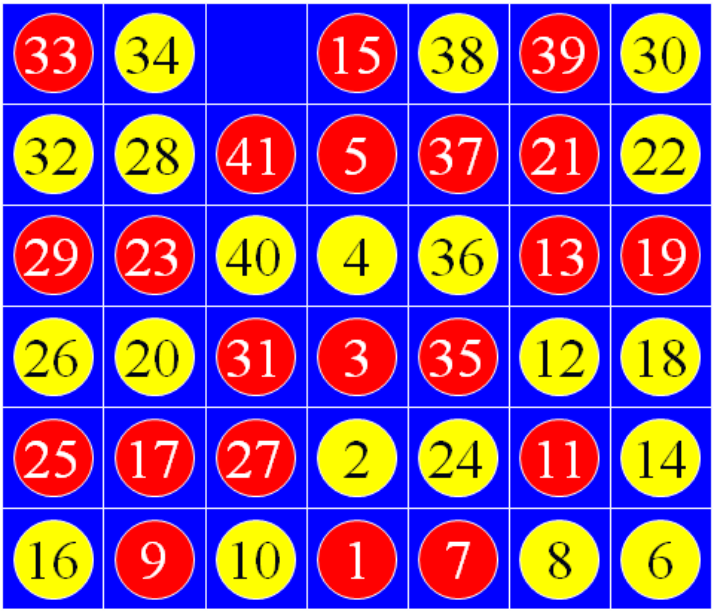

図 1 コネクト 4 の最善手[4]

1.5. 本研究の目的

前節で述べたように，コネクト 4 には既存の AI として Connect 4 Solver が存在する．Connect 4 Solver のプログラムは比較的シンプルに書かれており，効率よく着手選択できる．これはコネクト 4 の各手番の選択肢が最大

7 通りしかなく， $\alpha\beta$ 法でも短い時間で最適解が得られるためである．そこで，コネクト 4 のような選択肢の少ないゲームでもディープラーニングを用いた AI が有効であるかを検証するために，本研究では python を用いてコネクト 4 の AI を作成し，機械学習により強い AI を目指す．将棋や囲碁では，プロ棋士の棋譜を学習データとして用いることができる．しかし，コネクト 4 にはそのようなデータは無いため，まずランダムに石を落とすプログラムと AI を繰り返し対戦させることで学習させ，学習後に AI 同士の自己対戦でさらに学習させていく．また機械学習で最適解に辿り着けるか検証していく．

1.6. 本報告書の構成

本報告の構成は以下の通り出ある．まず第 2 章で本研究の対象であるコネクト 4 について述べる．第 3 章ではディープラーニングで学習を行うために本研究で作成した対戦用プログラムについて述べ，最後に第 5 章で結論 ，今後の課題について述べる．

2. コネクト 4

本章ではコネクト 4 について述べる．

「コネクト 4」[4]とは縦 6 マス，横 7 マスのボードと各 21 枚の 2 色の石を用いた 2 人用ゲームである．コネクト 4 のボードは縦に設置され，プレイヤーはボードの上部から石を投入することができる．投入された石はボードの一番下のマスか，すでに置かれた石の上のマスに配置される．各プレイヤーは交互に石を投入していき，縦・横・斜め・のいずれか先に 4 つの石を並べた方の勝利となる．また，今研究では置けないところに石を投入すると反則負けとなり，盤面全てが埋まっても勝敗がつかない場合は、引き分けとなる。

図 2 にコネクト 4 の盤面を示し，図 3, 4, 5 に縦，横，斜めでの勝利の例を示す．

図 6, 7 は反則負けと引き分けの例を示す．

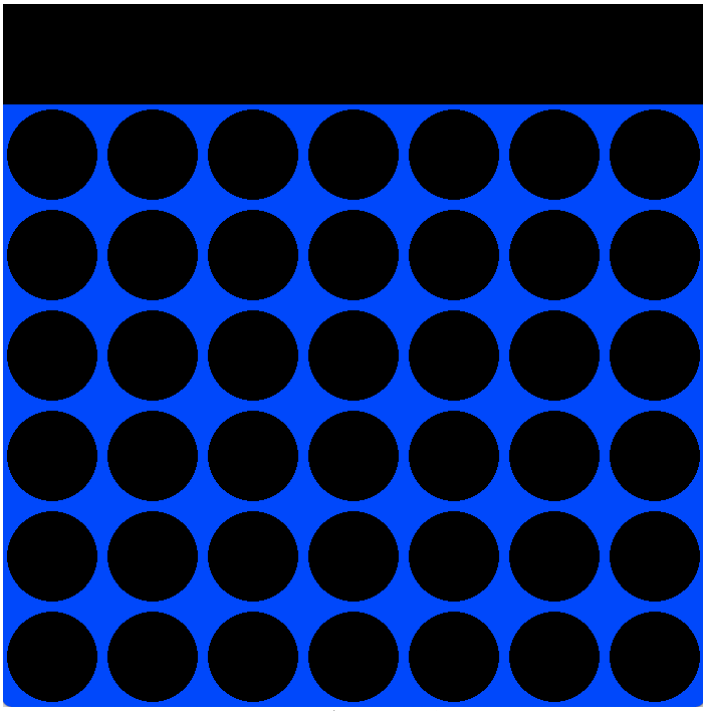

図 2　コネクト 4 盤面

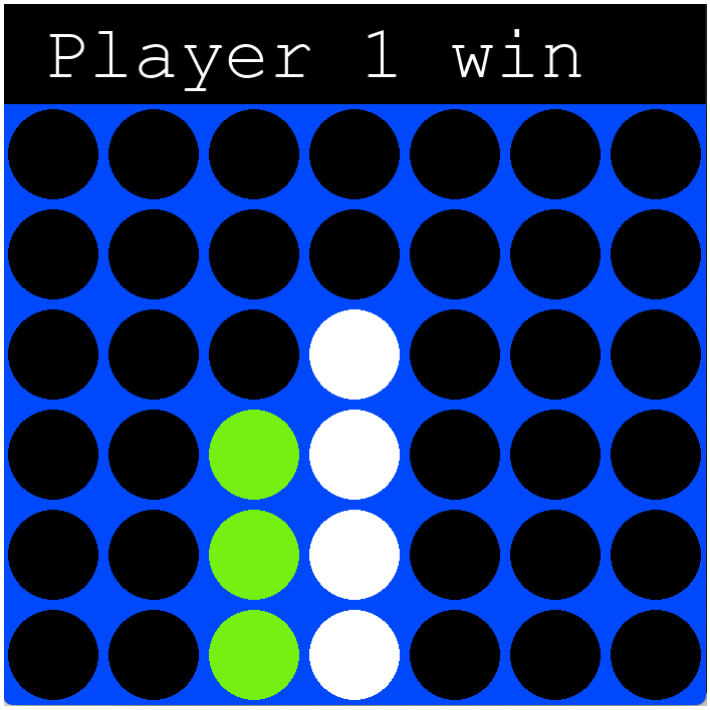

図 3　縦の列で勝利の例

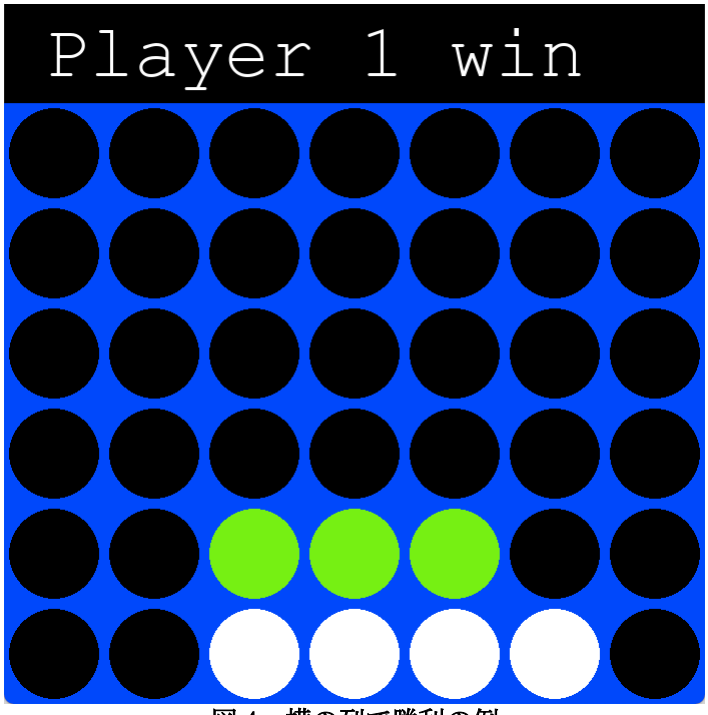

図 4　横の列で勝利の例

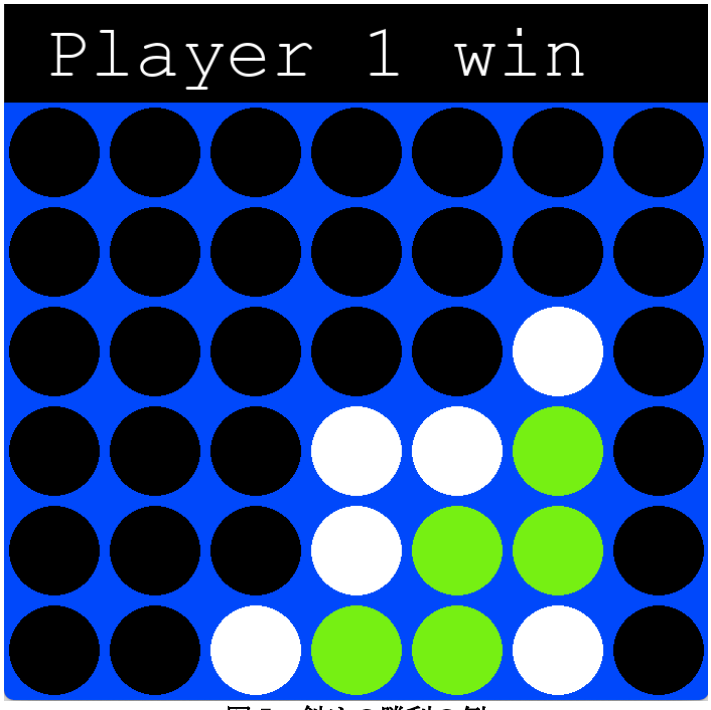

図５　斜めの勝利の例

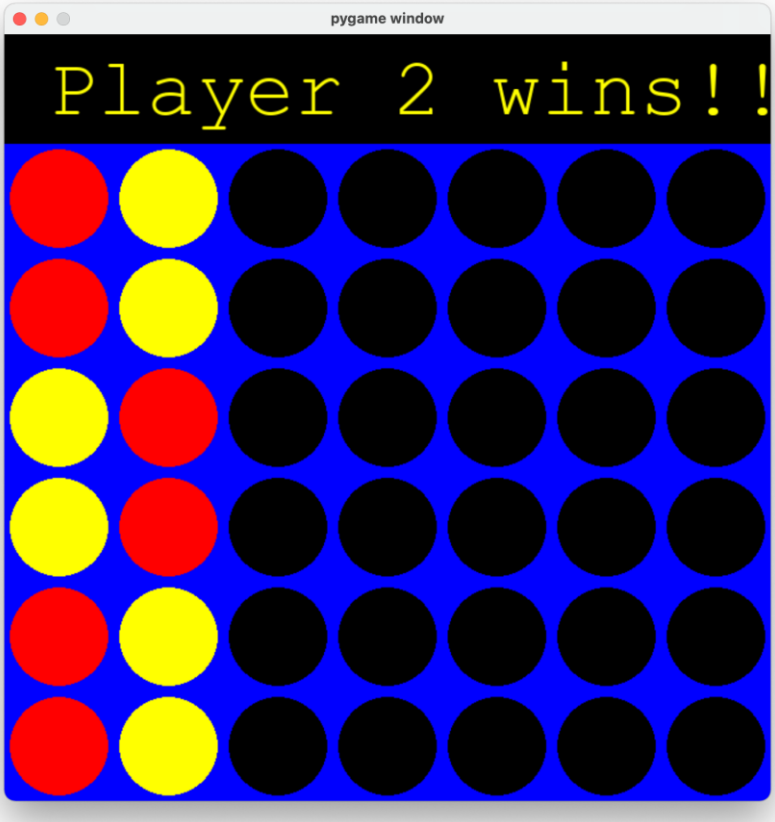

図６ プレイヤー１の反則負けの例

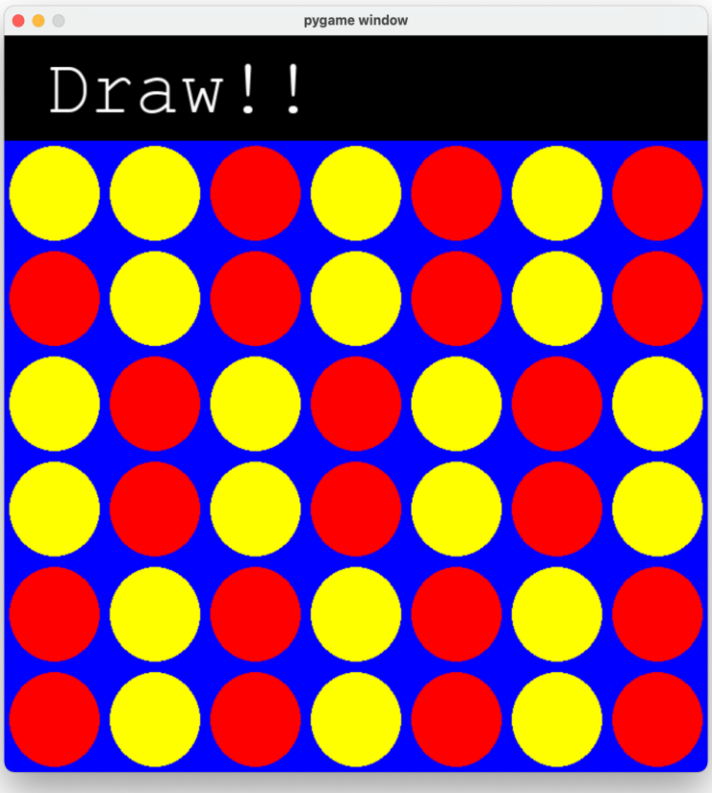

図 7　盤面がすべて埋まり引き分けとなった例

3.　学習用プログラムの作成

コネクト 4 の AI をディープラーニングを用いて作成するにあたって,学習を行うための対戦相手を必要とする.ディープラーニングで学習をするためには，学習度合に応じて異なる強さの対戦相手が必要である．しかし，既存の Connext 4 Silver は強すぎるため，初期の学習には適さない．そこで,本研究では,対戦相手として用いることができるプログラムを作成する.本研究で作成するプログラムは，Keith Galili 氏のコネクト 4 対人プログラム[21]を参考に作成している.

3.1. 学習用プログラムの戦略

ディープラーニングを用いて学習するにあたって、弱い対戦相手で学習を行い,学習が進むにつれて対戦相手を強くしていく必要がある.そこで本研究では,以下の戦略を用いて学習用プログラムを作成する.

- 戦略 1：ランダム

戦略 1 のプログラムは投入できる場所であれば,どこにでもランダムに石を投入する仕様である.

- 戦略 2：リーチ時に勝つ

戦略 2 のプログラムは自らの石が 3 つ並んでおり,かつ石が 4 つ並ぶ位置に石を置ける場合に,次の自分のターンで 4 つ目が揃う位置に石を投入する.また,リーチではない時はランダムに投入する.

- 戦略 3：相手のリーチを防ぐ

戦略 3 のプログラムは相手の石が 3 つ並んでおり,かつ石が 4 つ並ぶ位置に石を置ける場合に,それを防ぐ位置に自分の石を投入する.また,自分がリーチでかつ相手もリーチの場合は自分が有利となる所に石を置き,それ以外場合の時はランダムに投入する.

3.2. 学習用プログラム

本研究では python を用いて学習用プログラムを作成した.付録に本研究で作成したプログラムのソースを示す.
以下に本研究で作成したプログラムについて説明する.

- RandomAI.py

RandomAI.py は 3.1 節で述べた戦略 1 のプログラムである．RandomAI.py は石を投入できる場所からランダムに一つを選んで投入する．表 1 に RandomAI のメソッドを示す.

表 1 RandomAI.py のメソッド

create_board()	盤面の作成
dorp_drop_piece(board,col,row,piece)	石を投入
is_varid_location(board,row)	その列が全て石で埋まっているか
get_next_open_row(board)	その列で打てるマスの中で最下段を指す

- ReachAI1.py

ReachAI1.py は 3.1 節で述べた戦略 2 のプログラムである．ReachAI1.py は自らがリーチの状態であれば次のターンに必ず勝利するように投入する．表 2 に ReachAI1.py 独自のメソッドを示す．またこの条件以外は戦略 1 の通りに投入する.

表 2 ReachAI1.py 独自のメソッド

reach_attack1(board,piece)	相手のリーチを阻止する

- ReachAI2.py

ReachAI2.py は 3.2 節で述べた戦略 2 のプログラムである．ReachAI2.py は相手がリーチの場合にそれを阻止するように投入する表 3 に ReachAI2.py 独自のメソッドを示す．またこの条件以外は戦略 1 の通りに投入する.

表 3 ReachAI2.py 独自のメソッド

reach_attack2(board,piece)	相手のリーチを阻止する

4. プログラムの性能評価および考察

本研究で作成したプログラムの性能を評価するために，作成したプログラム同士の対戦，および既存の AI との対戦をそれぞれ 100 回ずつ行った．表 4 に各プログラムの勝利回数を示す.

表４　各プログラムの勝利回数(試行回数 100 回)

	RandomAI.py	ReachAI1.py	ReachAI2.py	connect4_with_ai.py
RandomAI.py		0	0	0
ReachAI1.py	100		4	0
ReachAI2.py	100	96		0
connect4_with_ai.py	100	100	100	

表 4 の結果から RandomAI.py が一度も勝利していない事がわかる.これは,他の AI では石がリーチした場合に次の手で勝利できるがこの AI では他の場所に置いてしまう為である.ReachAI1.py と ReachAI2.py では

ReachAI1.py は相手のリーチを防ぐ術を持っていない為,勝つ事が少なかった.ReachAI1.py が勝てたケースでは，ReachAI1.py がリーチ時にたまたまダブルリーチだった場合である. 1.4 節で述べた通りコネクト 4 は先手必勝であり[4]，connect4_with_ai.py は最善手を打つため，先手ならば必ず勝つ．また，connect4_with_ai.py が後手であっても，ReachAI2.py と connect4_with_ai.py では,connect4_with_ai.py がリーチするたびに ReachAI2.py は防ぐ石を投入するが,最終的に connect4_with_ai.py はダブルリーチを作ることができるため ReachAI2.py は必ず負けてしまう.

5. 結論・今後の課題

5.1. 結論

本研究ではディープラーニングを用いたゲーム AI を作成する予定であった為、ディープラーニング用の学習データが必要であった.ディープラーニングを用いて学習させるためには学習度合いに応じて対戦相手の強さを変える必要がある．しかし，コネクト 4 には非常に弱いランダム AI と,非常に強い connect4_with_ai はあるがその中間の AI が存在しない為,適度な強さを持つ AI を作成した．本研究では，学習用 AI の作成まではできたが，本来の目的であるディープラーニングを用いた AI を作成し，その AI に学習させるまでには至らなかった.

5.2. 今後の課題

ディープラーニングを用いることができなかった原因として, 自身の計画性と AI に対する知識が足りていなかったと感じられ, 今後の課題としてディープラーニングを用いたコネクト 4AI を作る. また, α β 法の AI とすぐに対戦すると, 勝つ事がないので学習が進まない為, 徐々に強い相手と対戦させる必要がある. 最善手を数手までうち, その後戦略 3 の動きをする AI などを追加し徐々に学習させていくことで, 既存の α β 法を用いた AI と最終的に同じ動きをするプログラムになるのか, またしないのならば, どういった結果になるのかを検証することが今後の課題としてあげられる.

謝辞

本研究を行うにあたって石水隆講師から開発に関する相談や資料のなどの提供など様々なご指導を受けました。ここに感謝の意を表します.

参考文献

[1] 藤田一弥，高原歩, 実装ディープラーニング, オーム社 (2016).

[2] 斎藤康毅, ゼロから作る Deep Learning - Python で学ぶディープラーニングの理論と実装，オラ イリージャパン（2016）

[3] James Dow Allen, The Complete Book of Connect 4, Puzzle Wright Press (2010)

[4] Victor Allis, A Knowledge-based Approach of Connect-Four, The Game is Solved: White Wins, Master Thesis, Department of Mathematics and Computer Science Vrije Universiteit (1988) http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf

[5] Yoshiaki Yamaguchi, Kazunori Yamaguchi, Tetsuro Tanaka, and Tomoyuki Kaneko, Infinite Connect-Four Is Solved: Draw, Advances in Computer Games, pp. 208-219 (2011)

[6] Yoshiaki Yamaguchi, Kazunori Yamaguchi, Tetsuro Tanaka, Cylinder-Infinite-Connect-Four except for Widths 2, 6, and 11 is Solved: Drawn, The 8th International Conference on Computers and Games (2013).

[7] Yoshiaki Yamaguchi, Todd W. Neller, First Player's Cannot-Lose Strategies for CylinderInfinite-Connect-Four with Widths 2 and 6, Advances in Computer Games, pp.113-121 (2015) http://cs.gettysburg.edu/~tneller/papers/acg2015.pdf

[8] 王銘宛, 棋士と AI：アルファ碁から始まった未来, 岩波新書, (2018).

[9] 斉藤康己, アルファ碁はなぜ人間に勝てたのか, ベスト新書, ベストセラーズ, (2016).

[10] 「グーグル囲碁ＡＩ「4 勝 1 敗」は人類の敗北か．プロ棋士から見た勝負の評価」，ニュースイッ チ，2016 年 3 月 6 日，日刊工業新聞，(2016) https://newswitch.jp/p/3971

[11] James Dow Allen, Expert Play in Connect-Four, (1990), http://tromp.github.io/c4.html

[12] Pascal Pons, Connect 4 Solver. https://connect4.gamesolver.org/

[13] Pascal Pons. Solving Connect 4：How To Build A Perfect AI, (2019) http://blog.gamesolver.org/

[14] 「共に電王戦出場、世界最強の"同僚"――コンピュータ将棋ソフト開発者 一丸貴則さん・山本 一 成 さ ん（ 前 編 ）」， 2014 年 4 月 25 日 ， ね と ら ぼ ， (2014) http://nlab.itmedia.co.jp/nl/articles/1404/25/news016.html

[15] 「 第 2 回 将 棋 電 王 戦 ／ 五 番 勝 負 」， 日 本 将 棋 連 盟 ， (2013) , https://www.shogi.or.jp/match/denou/2/index.html

[16] 「 第 3 回 将 棋 電 王 戦 ／ 五 番 勝 負 」， 日 本 将 棋 連 盟 ， (2014) https://www.shogi.or.jp/match/denou/3/index.html

[17] 「 将 棋 電 王 戦 FINAL ／ 五 番 勝 負 」， 日 本 将 棋 連 盟 ， (2015) , https://www.shogi.or.jp/match/denou/4/index.html

[18] 「 第 1 期 電 王 戦 ／ 二 番 勝 負 」 ， 日 本 将 棋 連 盟 ， (2016) , https://www.shogi.or.jp/match/denou/01/index.html

[19] 「 電 王 戦 ｜ 棋 戦 ｜ 日 本 将 棋 連 盟 」， 日 本 将 棋 連 盟 ， (2017) , https://www.shogi.or.jp/match/denou/

[20] 山本一成, 人工知能はどのようにして「名人」を超えたのか？：最強将棋 AI ポナンザの開発者が 教える機械学習・深層学習・強化学習の本質, ダイヤモンド社, (2017)

[21] Connect4-Python/connect4.py at master ・KeithGalli/Connect4-Python・GitHub, GitHub, (2017), https://github.com/KeithGalli/Connect4-Python/blob/master/connect4.py

ソースプログラム

本研究で作成したプログラムのソースファイルの一部を以下に示す

- RandamAI.py

```
import numpy as np
import random
import pygame
import sys
import math

BLUE = (0,0,255)
BLACK = (0,0,0)
GREEN = (0,255,0)

ROW_COUNT = 6
COLUMN_COUNT = 7

AI = 1


def create_board():
    board = np.zeros((ROW_COUNT,COLUMN_COUNT))
    return board

def drop_piece(board, row, col, piece):
    board[row][col] = piece

def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0

def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r

def print_board(board):
    print(np.flip(board, 0))

def winning_move(board, piece):

    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] ==
piece:
                return True


    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3][c] ==
piece:
                return True


    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and board[r+3][c+3]
== piece:
return True
```

```
    for c in range(COLUMN_COUNT-3):
        for r in range(3, ROW_COUNT):
            if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and board[r-3][c+3]
== piece:
                return True


def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE, SQUARESIZE))
            pygame.draw.circle(screen,                    BLACK,                    (int(c*SQUARESIZE+SQUARESIZE/2),
int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == AI_PIECE:
                pygame.draw.circle(screen,         GREEN,         (int(c*SQUARESIZE+SQUARESIZE/2),         height-
int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
    pygame.display.update()


board = create_board()
print_board(board)
game_over = False


pygame.init()

SQUARESIZE = 100

width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE

size = (width, height)

RADIUS = int(SQUARESIZE/2 - 5)

screen = pygame.display.set_mode(size)
draw_board(board)
pygame.display.update()

myfont = pygame.font.SysFont("monospace", 75)

while not game_over:

    for event in pygame.event.get():


    if turn == AI and not game_over:


        col = random.randint(0, COLUMN_COUNT-1)

        if is_valid_location(board, col):
            pygame.time.wait(500)
            row = get_next_open_row(board, col)
```

```
            drop_piece(board, row, col, 1)
            if winning_move(board, 1):
                label = myfont.render("Player 2 wins!!", 1, GREEN)
                screen.blit(label, (40,10))
                game_over = True

            print_board(board)
            draw_board(board)

            turn += 1
      if turn >= 42:
        label = myfont.render("Draw!!", 1, WHITE)
        screen.blit(label, (40,10))
        game_over = True
      print_board(board)
      draw_board(board)

    if game_over:
        pygame.time.wait(3000)
```

- ReachAI1.py

```
import numpy as np
import random
import pygame
import sys
import math

BLUE = (0,0,255)
BLACK = (0,0,0)

GREEN = (0,255,0)

ROW_COUNT = 6
COLUMN_COUNT = 7

AI = 1

EMPTY = 0

AI_PIECE = 2

WINDOW_LENGTH = 4

def create_board():
    board = np.zeros((ROW_COUNT,COLUMN_COUNT))
    return board

def drop_piece(board, row, col, piece):
    board[row][col] = piece

def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0

def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r
```

```
def print_board(board):
    print(np.flip(board, 0))

def winning_move(board, piece):

    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):

            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r]          [c+3]
== piece:
                return True


    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3]          [c]
== piece:
                return True


    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if  board[r][c]  ==  piece  and  board[r+1][c+1]  ==  piece  and  board[r+2][c+2]  ==  piece  and
board[r+3][c+3] == piece:
                return True


    for c in range(COLUMN_COUNT-3):
        for r in range(3, ROW_COUNT):
            if  board[r][c]  ==  piece  and  board[r-1][c+1]  ==  piece  and  board[r-2][c+2]  ==  piece  and
board[r-3][c+3] == piece:
                return True

def evaluate_window(window, piece):
        score =0
        opp_piece = PLAYER_PIECE
        if piece == PLAYER_PIECE:
                opp_piece = AI_PIECE

        if window.count(piece) == 4:
                score += 100
        elif window.count(piece) == 3 and window.count(EMPTY) == 1:
                score += 10
        elif window.count(piece) == 2 and window.count(EMPTY) == 2:
                score += 5
        if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
                score -= 80
        return score

def score_position(board, piece):
        score = 0
        for r in range(ROW_COUNT):
                row_array = [int(i) for i in list(board[r,:])]
                for c in range(COLUMN_COUNT-3):
                        window = row_array[c:c+WINDOW_LENGTH]
                        if window.count(piece) == 4:
                score += 100
            elif window.count(piece) == 3 and window.count(EMPTY) == 1:
                score += 10
```

```
        return score


def get_valid_locations(board):
        valid_locations = []
        for col in range(COLUMN_COUNT):
                if is_valid_location(board, col):

                valid_locations.append(col)
                return valid_locations

def reach_attack1(board, piece):
        valid_locations = get_valid_locations(board)
        best_score = -10000
        best_col = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
            emp_board = board.copy()
            drop_piece(temp_board, row, col, piece)
            score = score_position(temp_board, piece)
                if score > best_score:
                    best_score = score
                    best_col = col

        return best_col


def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE,
SQUARESIZE))
            pygame.draw.circle(screen, BLACK, (int(c*SQUARESIZE+SQUARESIZE/2),
int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == PLAYER_PIECE:
                pygame.draw.circle(screen, RED, (int(c*SQUARESIZE+SQUARESIZE/2), height-i
nt(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            elif board[r][c] == AI_PIECE:
                pygame.draw.circle(screen, YELLOW, (int(c*SQUARESIZE+SQUARESIZE/2), height-
int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            pygame.display.update()


board = create_board()
print_board(board)
game_over = False


pygame.init()

SQUARESIZE = 100

width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE
```

```
size = (width, height)

RADIUS = int(SQUARESIZE/2 - 5)

screen = pygame.display.set_mode(size)
draw_board(board)
pygame.display.update()

myfont = pygame.font.SysFont("monospace", 75)

while not game_over:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

        if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == PLAYER:
                pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)

        pygame.display.update()



            # # Ask for Player 2 Input
    if turn == AI :

        col = reach_attack1(board, AI_PIECE)

        if is_valid_location(board, col):
            pygame.time.wait(500)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, AI_PIECE)

            if winning_move(board, AI_PIECE):
                label = myfont.render("Player 2 wins!!", 1, GREEN)
                screen.blit(label, (40,10))
                game_over = True

            print_board(board)
            draw_board(board)

            turn += 1
     if turn >= 42:
        label = myfont.render("Draw!!", 1, WHITE)
        screen.blit(label, (40,10))
        game_over = True
      print_board(board)
      draw_board(board)
    if game_over:
        pygame.time.wait(3000)
```

- ReachAI2.py

```
import numpy as np
import random
import pygame
```

```
import sys
import math

BLUE = (0,0,255)
BLACK = (0,0,0)
GREEN = (0,255,0)

ROW_COUNT = 6
COLUMN_COUNT = 7
AI = 1

EMPTY = 0

AI_PIECE = 2

WINDOW_LENGTH = 4

def create_board():
    board = np.zeros((ROW_COUNT,COLUMN_COUNT))
    return board

def drop_piece(board, row, col, piece):
    board[row][col] = piece

def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0

def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r

def print_board(board):
    print(np.flip(board, 0))

def winning_move(board, piece):
    # Check horizontal locations for win
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r][c+3] ==
piece:
                return True

    # Check vertical locations for win
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3]         [c]
== piece:
                return True

    # Check positively sloped diaganols
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if  board[r][c]  ==  piece  and  board[r+1][c+1]  ==  piece  and  board[r+2][c+2]  ==  piece  and
board[r+3][c+3] == piece:
                return True

    # Check negatively sloped diaganols
    for c in range(COLUMN_COUNT-3):
```

```
        for r in range(3, ROW_COUNT):
            if board[r][c] == piece and board[r-1][c+1] == piece and board[r-2][c+2] == piece and
board[r-3][c+3] == piece:
                return True

def evaluate_window(window, piece):
        score =0
        opp_piece = PLAYER_PIECE
        if piece == PLAYER_PIECE:
                opp_piece = AI_PIECE

        if window.count(piece) == 4:
                score += 100
        elif window.count(piece) == 3 and window.count(EMPTY) == 1:
                score += 10
        elif window.count(piece) == 2 and window.count(EMPTY) == 2:
                score += 5
        if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
                score -= 80
        return score

def score_position(board, piece):
        score = 0
        for r in range(ROW_COUNT):
                row_array = [int(i) for i in list(board[r,:])]
                for c in range(COLUMN_COUNT-3):
                        window = row_array[c:c+WINDOW_LENGTH]
                        score += evaluate_window(window, piece)


        for c in range(COLUMN_COUNT):
                col_array = [int(i) for i in list(board[:,c])]

                for r in range(ROW_COUNT-3):
                        window = col_array[r:r+WINDOW_LENGTH]
                        score += evaluate_window(window, piece)

        for r in range(ROW_COUNT-3):
                for c in range(COLUMN_COUNT-3):
                        window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
                        score += evaluate_window(window, piece)
        for r in range(ROW_COUNT-3):
                for c in range(COLUMN_COUNT-3):
                        window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)]
                        score += evaluate_window(window, piece)
        return score


def get_valid_locations(board):
        valid_locations = []
        for col in range(COLUMN_COUNT):
                if is_valid_location(board, col):
                        valid_locations.append(col)
        return valid_locations

def reach_attak2(board, piece):
        valid_locations = get_valid_locations(board)
        best_score = -10000
        best_col = random.choice(valid_locations)
```

```
        for col in valid_locations:
                row = get_next_open_row(board, col)
                temp_board = board.copy()
                drop_piece(temp_board, row, col, piece)
                score = score_position(temp_board, piece)
                if score > best_score:
                        best_score = score
                        best_col = col

        return best_col



def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE, SQUARESIZE,
SQUARESIZE))
            pygame.draw.circle(screen, BLACK, (int(c*SQUARESIZE+SQUARESIZE/2),
                int(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == PLAYER_PIECE:
                pygame.draw.circle(screen, RED, (int(c*SQUARESIZE+SQUARESIZE/2), height-
int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            elif board[r][c] == AI_PIECE:
                pygame.draw.circle(screen, YELLOW, (int(c*SQUARESIZE+SQUARESIZE/2), height-
int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
    pygame.display.update()


board = create_board()
print_board(board)
game_over = False


pygame.init()

SQUARESIZE = 100

width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE

size = (width, height)

RADIUS = int(SQUARESIZE/2 - 5)

screen = pygame.display.set_mode(size)
draw_board(board)
pygame.display.update()

myfont = pygame.font.SysFont("monospace", 75)


while not game_over:

    for event in pygame.event.get():
```

```
    if turn == AI and not game_over:


        col = Reach_attack2(board, AI_PIECE)

        if is_valid_location(board, col):
            pygame.time.wait(500)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, AI_PIECE)

            if winning_move(board, AI_PIECE):
                label = myfont.render("Player 2 wins!!", 1, GREEN)
                screen.blit(label, (40,10))
                game_over = True

            print_board(board)
            draw_board(board)

            turn += 1

     if turn >= 42:
        label = myfont.render("Draw!!", 1, WHITE)
        screen.blit(label, (40,10))
        game_over = True
      print_board(board)
      draw_board(board)
    if game_over:
        pygame.time.wait(3000)
```

● connect_with_ai

```
import numpy as np
import random
import pygame
import sys
import math

BLUE = (0,0,255)
BLACK = (0,0,0)
RED = (255,0,0)
YELLOW = (255,255,0)

ROW_COUNT = 6
COLUMN_COUNT = 7

PLAYER = 0
AI = 1

EMPTY = 0
PLAYER_PIECE = 1
AI_PIECE = 2

WINDOW_LENGTH = 4

def create_board():
    board = np.zeros((ROW_COUNT,COLUMN_COUNT))
    return board
```

```
def drop_piece(board, row, col, piece):
    board[row][col] = piece

def is_valid_location(board, col):
    return board[ROW_COUNT-1][col] == 0

def get_next_open_row(board, col):
    for r in range(ROW_COUNT):
        if board[r][col] == 0:
            return r

def print_board(board):
    print(np.flip(board, 0))

def winning_move(board, piece):
    # Check horizontal locations for win
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT):
            if board[r][c] == piece and board[r][c+1] == piece and board[r][c+2] == piece and board[r]         [c+3]
== piece:
                return True

    # Check vertical locations for win
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c] == piece and board[r+2][c] == piece and board[r+3]         [c]
== piece:
                return True

    # Check positively sloped diaganols
    for c in range(COLUMN_COUNT-3):
        for r in range(ROW_COUNT-3):
            if board[r][c] == piece and board[r+1][c+1] == piece and board[r+2][c+2] == piece and
            board[r+3][c+3] == piece:
                return True

    # Check negatively sloped diaganols
    for c in range(COLUMN_COUNT-3):
        for r in range(3, ROW_COUNT):
            if  board[r][c]  ==  piece  and  board[r-1][c+1]  ==  piece  and  board[r-2][c+2]  ==  piece  and
board[r-3][c+3] == piece:
                return True

def evaluate_window(window, piece):
    score = 0
    opp_piece = PLAYER_PIECE
    if piece == PLAYER_PIECE:
        opp_piece = AI_PIECE

    if window.count(piece) == 4:
        score += 100
    elif window.count(piece) == 3 and window.count(EMPTY) == 1:
        score += 5
    elif window.count(piece) == 2 and window.count(EMPTY) == 2:
        score += 2

    if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
        score -= 4
```

```
    return score

def score_position(board, piece):
    score = 0

    ## Score center column
    center_array = [int(i) for i in list(board[:, COLUMN_COUNT//2])]
    center_count = center_array.count(piece)
    score += center_count * 3

    ## Score Horizontal
    for r in range(ROW_COUNT):
        row_array = [int(i) for i in list(board[r,:])]
        for c in range(COLUMN_COUNT-3):
            window = row_array[c:c+WINDOW_LENGTH]
            score += evaluate_window(window, piece)

    ## Score Vertical
    for c in range(COLUMN_COUNT):
        col_array = [int(i) for i in list(board[:,c])]
        for r in range(ROW_COUNT-3):
            window = col_array[r:r+WINDOW_LENGTH]
            score += evaluate_window(window, piece)

    ## Score posiive sloped diagonal
    for r in range(ROW_COUNT-3):
        for c in range(COLUMN_COUNT-3):
            window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)

    for r in range(ROW_COUNT-3):
        for c in range(COLUMN_COUNT-3):
            window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)

    return score

def is_terminal_node(board):
    return winning_move(board, PLAYER_PIECE) or winning_move(board, AI_PIECE) or len(get_valid_locations(board))
== 0

def minimax(board, depth, alpha, beta, maximizingPlayer):
    valid_locations = get_valid_locations(board)
    is_terminal = is_terminal_node(board)
    if depth == 0 or is_terminal:
        if is_terminal:
            if winning_move(board, AI_PIECE):
                return (None, 100000000000000)
            elif winning_move(board, PLAYER_PIECE):
                return (None, -10000000000000)
            else: # Game is over, no more valid moves
                return (None, 0)
        else: # Depth is zero
            return (None, score_position(board, AI_PIECE))
    if maximizingPlayer:
        value = -math.inf
        column = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
```

```
            b_copy = board.copy()
            drop_piece(b_copy, row, col, AI_PIECE)
            new_score = minimax(b_copy, depth-1, alpha, beta, False)[1]
            if new_score > value:
                value = new_score
                column = col
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return column, value

    else: # Minimizing player
        value = math.inf
        column = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, PLAYER_PIECE)
            new_score = minimax(b_copy, depth-1, alpha, beta, True)[1]
            if new_score < value:
                value = new_score
                column = col
            beta = min(beta, value)
            if alpha >= beta:
                break
        return column, value

def get_valid_locations(board):
    valid_locations = []
    for col in range(COLUMN_COUNT):
        if is_valid_location(board, col):
            valid_locations.append(col)
    return valid_locations

def draw_board(board):
    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):

            pygame.draw.rect(screen, BLUE, (c*SQUARESIZE, r*SQUARESIZE+SQUARESIZE,
             SQUARESIZE, SQUARESIZE))
            pygame.draw.circle(screen,            BLACK,            (int(c*SQUARESIZE+SQUARESIZE/2),            I
nt(r*SQUARESIZE+SQUARESIZE+SQUARESIZE/2)), RADIUS)

    for c in range(COLUMN_COUNT):
        for r in range(ROW_COUNT):
            if board[r][c] == PLAYER_PIECE:
                pygame.draw.circle(screen,          RED,          (int(c*SQUARESIZE+SQUARESIZE/2),          height-i
nt(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            elif board[r][c] == AI_PIECE:
                pygame.draw.circle(screen,  YELLOW,  (int(c*SQUARESIZE+SQUARESIZE/2),                          height-
int(r*SQUARESIZE+SQUARESIZE/2)), RADIUS)
            pygame.display.update()

board = create_board()
print_board(board)
game_over = False

pygame.init()
```

```
SQUARESIZE = 100

width = COLUMN_COUNT * SQUARESIZE
height = (ROW_COUNT+1) * SQUARESIZE

size = (width, height)

RADIUS = int(SQUARESIZE/2 - 5)

screen = pygame.display.set_mode(size)
draw_board(board)
pygame.display.update()

myfont = pygame.font.SysFont("monospace", 75)

turn = random.randint(PLAYER, AI)

while not game_over:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

        if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == PLAYER:
                pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)

        pygame.display.update()

        if event.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            #print(event.pos)
            # Ask for Player 1 Input
            if turn == PLAYER:
                posx = event.pos[0]
                col = int(math.floor(posx/SQUARESIZE))

                if is_valid_location(board, col):
                    row = get_next_open_row(board, col)
                    drop_piece(board, row, col, PLAYER_PIECE)

                    if winning_move(board, PLAYER_PIECE):
                        label = myfont.render("Player 1 wins!!", 1, RED)

                        screen.blit(label, (40,10))
                        game_over = True

                    print_board(board)
                    draw_board(board)


    # # Ask for Player 2 Input
    if turn == AI and not game_over:

        col, minimax_score = minimax(board, 5, -math.inf, math.inf, True)

        if is_valid_location(board, col):
```

```
        #pygame.time.wait(500)
        row = get_next_open_row(board, col)
        drop_piece(board, row, col, AI_PIECE)

        if winning_move(board, AI_PIECE):
            label = myfont.render("Player 2 wins!!", 1, YELLOW)
            screen.blit(label, (40,10))
            game_over = True

        print_board(board)
        draw_board(board)

        turn += 1
       if turn >= 42:
    label = myfont.render("Draw!!", 1, WHITE)
    screen.blit(label, (40,10))
    game_over = True
  print_board(board)
  draw_board(board)

if game_over:
    pygame.time.wait(3000)
```