Lab 4: Minimax Search

Nim

Starting with N objects (e.g., sticks), each player alternates turns, taking 1, 2, or 3 objects per turn. The goal is to take the last object.

Player 1 Moves

Player 2 Moves

Take 3

Take 3

Take 1

Take 1

Take 2

Mancala

- Many variations. Our version: goal is to get as many stones in your house (end to your right)

Image from https://punchboard.co.uk/how-to-play-mancala/

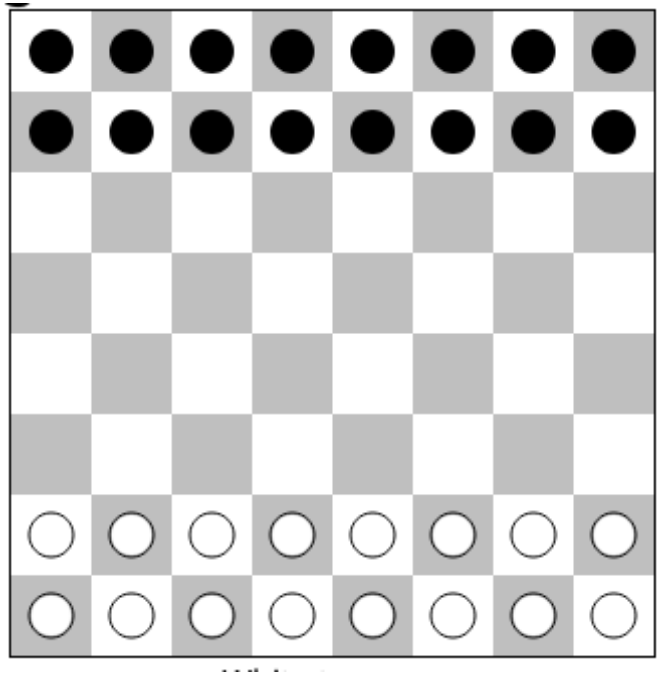
Moves

- Pick up stones from one pocket and distribute one in each pocket going counter-clockwise.

Image from https://punchboard.co.uk/how-to-play-mancala/

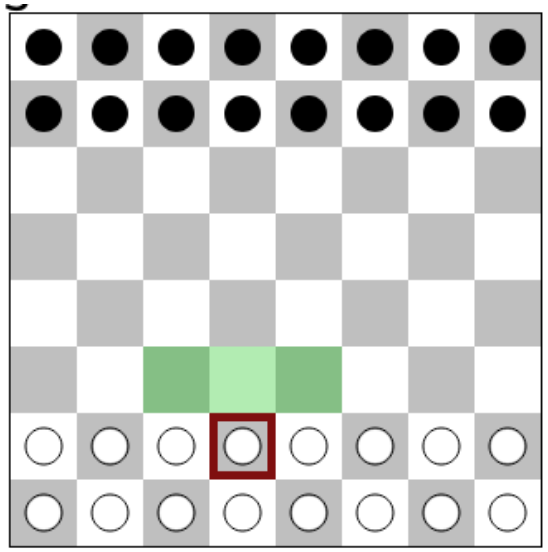
Breakthrough

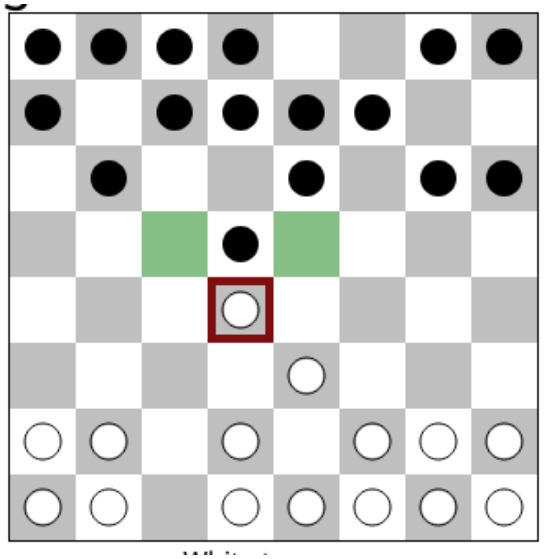
- Goal: be first to reach other side

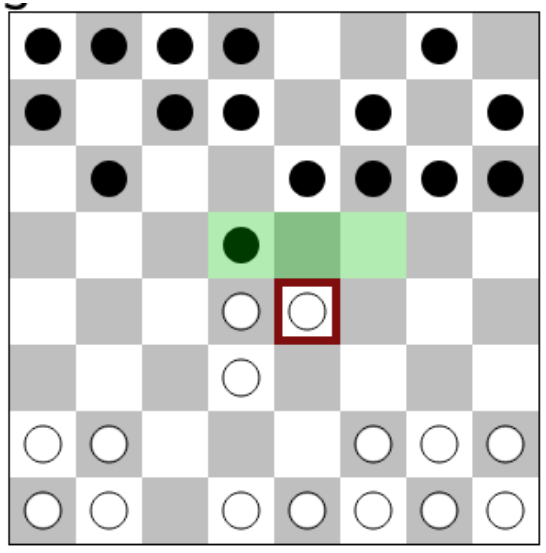


Moves

- Can only ever move forward (toward other side) to an adjacent square
- For unoccupied spaces: straight, diagonal-left, diagonal-right
- For occupied spaces: cannot move into a space you occupy, and cannot move straight into an opponent space
- Capture: can capture an opponent space with diagonal move only







Code Base

Games are defined in:

- Nim.py
- Mancala.py
- Breakthrough.py

Player Agents:

- BasicPlayers.py
- MinMaxPlayers.py

Overall scripts and heuristics:

- PlayGame.py
- StaticEvaluators.py

Code Base

Games are defined in:

- Nim.py
- Mancala.py
- Breakthrough.py

Player Agents:

- BasicPlayers.py
- **MinMaxPlayers.py**

Overall scripts and heuristics:

- PlayGame.py
- **StaticEvaluators.py**

Bolded files involve implementation, rest are provided code

But start by studying all games and agent classes

Infinity

Several ways to represent infinity in Python. The easiest is:

```
bestVal = float('inf') # or 'infinity'
bestVal = float('-inf')
```

Inheritance and Polymorphism

- The `Game` class is abstract – defines an interface without implementation

```python
class Game:
    def __repr__(self):⋯


    def makeMove(self, move):
        """Returns a new game instance in which move has been played.
        """
        raise NotImplementedError("Interface class, should not be instanciated")

    @property
    def availableMoves(self):
        """List of legal moves for the current player."""
        raise NotImplementedError("Interface class, should not be instanciated")

    @property
    def isTerminal(self):
        """Boolean indicating whether the game has ended."""
        raise NotImplementedError("Interface class, should not be instanciated")

    @property
    def winner(self):
        """+1 if the first player (maximizer) has won. -1 if the second player
        (minimizer) has won. 0 if the game is a draw. Raises an AttributeError
        if accessed on a non-terminal state."""
        raise NotImplementedError("Interface class, should not be instanciated")
```

Inheritance and Polymorphism

- Behavior is dynamic – depends on the actual underlying object (game)
- But code is general
  reusable
  extensible
  flexible

```
def play_game(game, player1, player2, show=False):
    """Plays a game then returns the final state."""
    while not game.isTerminal:
        if show:
            print(game)
        if game.turn == 1:
            m = player1.getMove(game)
            if show:
                print("player1 made move", m)
        else:
            m = player2.getMove(game)
            if show:
                print("player2 made move", m)
        if m not in game.availableMoves:
            raise Exception("invalid move: " + str(m))
        game = game.makeMove(m)
    if show:
        print(game, "\n")
        if game.winner != 0:
            print("player", game._print_char(game.winner), "wins")
        else:
            print("it's a draw")
    return game
```