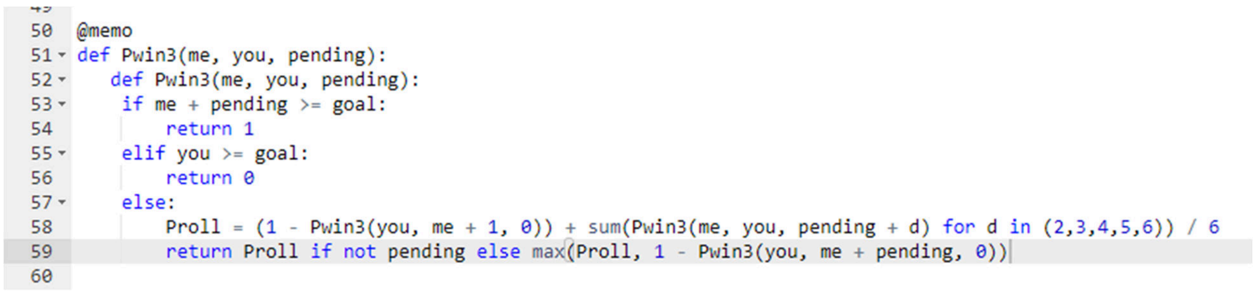
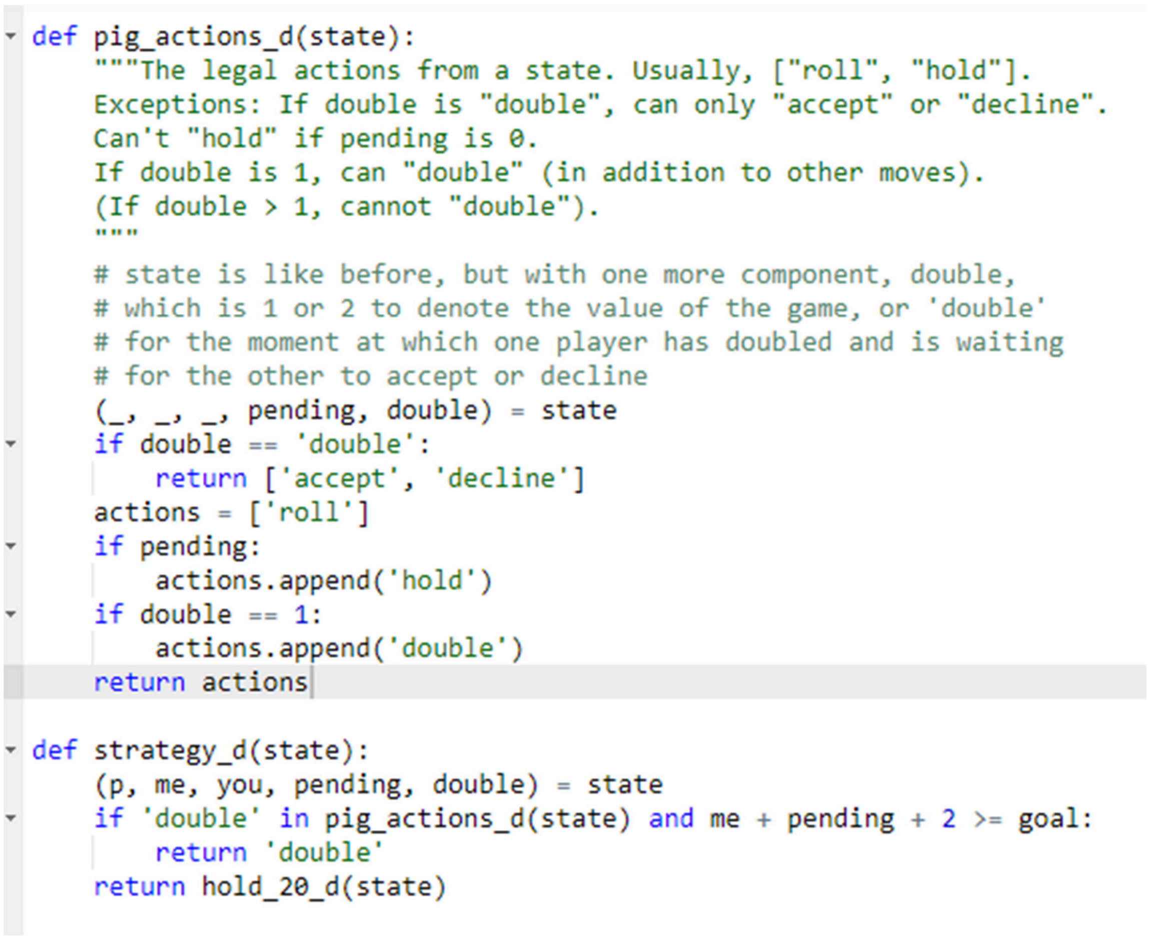
Improving Optimal



```
@memo
def Pwin3(me, you, pending):
    def Pwin3(me, you, pending):
     if me + pending >= goal:
         return 1
     elif you >= goal:
         return 0
     else:
         Proll = (1 - Pwin3(you, me + 1, 0)) + sum(Pwin3(me, you, pending + d) for d in (2,3,4,5,6)) / 6
         return Proll if not pending else max(Proll, 1 - Pwin3(you, me + pending, 0))
```

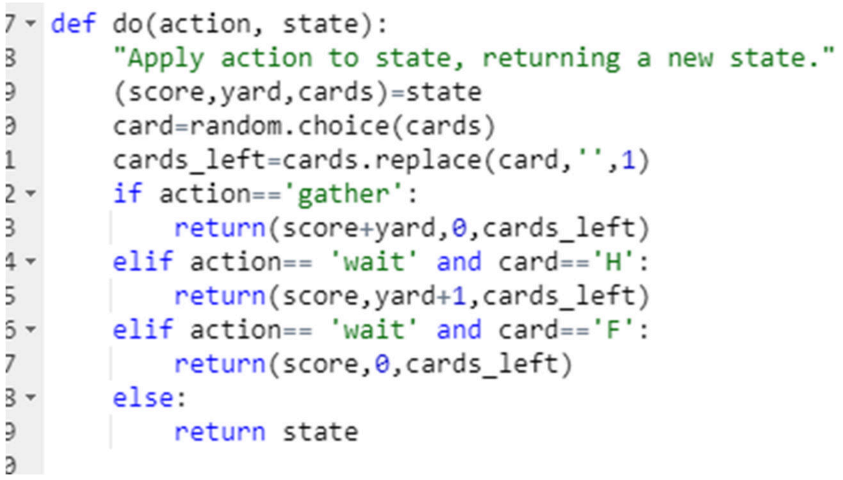
Doubling Pigs



```
def pig_actions_d(state):
    """The legal actions from a state. Usually, ["roll", "hold"].
    Exceptions: If double is "double", can only "accept" or "decline".
    Can't "hold" if pending is 0.
    If double is 1, can "double" (in addition to other moves).
    (If double > 1, cannot "double").
    """
    # state is like before, but with one more component, double,
    # which is 1 or 2 to denote the value of the game, or 'double'
    # for the moment at which one player has doubled and is waiting
    # for the other to accept or decline
    (_, _, _, pending, double) = state
    if double == 'double':
        return ['accept', 'decline']
    actions = ['roll']
    if pending:
        actions.append('hold')
    if double == 1:
        actions.append('double')
    return actions

def strategy_d(state):
    (p, me, you, pending, double) = state
    if 'double' in pig_actions_d(state) and me + pending + 2 >= goal:
        return 'double'
    return hold_20_d(state)
```

Foxes and Hens



```
def do(action, state):
    "Apply action to state, returning a new state."
    (score,yard,cards)=state
    card=random.choice(cards)
    cards_left=cards.replace(card,'',1)
    if action=='gather':
        return(score+yard,0,cards_left)
    elif action== 'wait' and card=='H':
        return(score,yard+1,cards_left)
    elif action== 'wait' and card=='F':
        return(score,0,cards_left)
    else:
        return state
```