Hog

● Graded

Group

Seong Jae Ahn
Sangwon Ji
View or edit group

Total Points

25 / 24 pts

Autograder Score

25.0 / 24.0

Autograder Results

```
=====================================================================
Assignment: Project 1: Hog
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

---------------------------------------------------------------------
Question 0
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Question 1
Passed: 3
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Question 2
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Question 3
Passed: 2
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Question 4
Passed: 1
```

Failed: 0
[ooooooooook] 100.0% passed

Question 5
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

Question 6
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

Question 7
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

Question 8
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

Question 9
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

Question 10
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

Question 11
Passed: 1
Failed: 0
[oooooooooook] 100.0% passed

Question 12
Passed: 0
Failed: 0
[k.........] 0.0% passed

Point breakdown
Question 0: 0.0/0
Question 1: 2.0/2

Question 2: 2.0/2
Question 3: 2.0/2
Question 4: 2.0/2
Question 5: 4.0/4
Question 6: 2.0/2
Question 7: 2.0/2
Question 8: 2.0/2
Question 9: 2.0/2
Question 10: 2.0/2
Question 11: 2.0/2
Question 12: 0.0/0

Score:
Total: 24.0

Cannot backup when running ok with --local.

Final Score:25.0
Early Submission. Bonus Point Included

Submitted Files

hog.py Download

```
"""The Game of Hog."""

from dice import six_sided, make_test_dice
from ucb import main, trace, interact

GOAL = 100  # The goal of Hog is to score 100 points.

######################
# Phase 1: Simulator #
######################


def roll_dice(num_rolls, dice=six_sided):
    """Simulate rolling the DICE exactly NUM_ROLLS > 0 times. Return the sum of
    the outcomes unless any of the outcomes is 1. In that case, return 1.

    num_rolls:  The number of dice rolls that will be made.
    dice:       A function that simulates a single dice roll outcome.
    """
    # These assert statements ensure that num_rolls is a positive integer.
    assert type(num_rolls) == int, 'num_rolls must be an integer.'
    assert num_rolls > 0, 'Must roll at least once.'
    # BEGIN PROBLEM 1
    "*** YOUR CODE HERE ***"
    sum=0
    temp = False # consider as a switch if dice hits 1. True : Dice hits 1 or not it remains False
    while 1 :
        value=dice()
        num_rolls=num_rolls-1
        if value!=1 :
            sum = sum + value
            #print("this is sum ")
            #print(sum)
        if value ==1:
            temp=True # made variable 'temp' as dice hit 1. So at the end return 1

        if num_rolls==0 and temp ==True:
            return 1

        if num_rolls==0 and temp ==False:
            return sum


"""
>>> print(roll_dice(3,make_test_dice(4,1,2,6)))
>>> print(roll_dice(1,make_test_dice(4,1,2,6)))

"""

```

```
50
51  #print(roll_dice(2,make_test_dice(4,2,3,1)))
52  #print(roll_dice(1,make_test_dice(4,2,3,1)))
53
54
55      # END PROBLEM 1
56
57
58  def boar_brawl(player_score, opponent_score):
59      """Return the points scored by rolling 0 dice according to Boar Brawl.
60
61      player_score:     The total score of the current player.
62      opponent_score:   The total score of the other player.
63          >>> boar_brawl(21312,4231)
64          >>> boar_brawl(21,12)
65          >>> boar_brawl(100,9)
66
67      """
68      # BEGIN PROBLEM 2
69      "*** YOUR CODE HERE ***"
70      players_first_digit = -1
71      opponent_second_digit=-1
72      players_first_digit=player_score % 10
73      opponent_second_else=opponent_score//10
74
75      def op_second_digit_check_func(x):
76
77          if x==0:
78              return x
79          elif x <10:
80              return x
81          else :
82              return x %10
83
84      opponent_second_digit=op_second_digit_check_func(opponent_second_else)
85
86
87      if opponent_second_digit != players_first_digit %10:
88          return 3 * abs(opponent_second_digit -players_first_digit)
89
90      else :
91       return 1
92
93      # END PROBLEM 2
94
95
96  def take_turn(num_rolls, player_score, opponent_score, dice=six_sided):
97      """Return the points scored on a turn rolling NUM_ROLLS dice when the
98      player has PLAYER_SCORE points and the opponent has OPPONENT_SCORE points.
99
100     num_rolls:       The number of dice rolls that will be made.
101     player_score:    The total score of the current player.
```

```
    opponent_score:  The total score of the other player.
    dice:            A function that simulates a single dice roll outcome.
    """
    # Leave these assert statements here; they help check for errors.
    assert type(num_rolls) == int, 'num_rolls must be an integer.'
    assert num_rolls >= 0, 'Cannot roll a negative number of dice in take_turn.'
    assert num_rolls <= 10, 'Cannot roll more than 10 dice.'
    # BEGIN PROBLEM 3
    "*** YOUR CODE HERE ***"
    if num_rolls ==0:
        return boar_brawl(player_score, opponent_score)
    else :
        return roll_dice(num_rolls, dice)



    # END PROBLEM 3


def simple_update(num_rolls, player_score, opponent_score, dice=six_sided):
    """Return the total score of a player who starts their turn with
    PLAYER_SCORE and then rolls NUM_ROLLS DICE, ignoring Fuzzy Factors.
    """
    if num_rolls==0:
        score = player_score + boar_brawl(player_score, opponent_score) #activates boar_bowl
        return score
    else :
        return player_score + roll_dice(num_rolls,dice)


def hog_gcd(x, y):
    """Return the greatest common divisor between X and Y"""
    # BEGIN PROBLEM 4
    "*** YOUR CODE HERE ***"
    if(y == 0):
        return abs(x)

    else:
        return hog_gcd(y, x % y)




    # END PROBLEM 4


def fuzzy_points(score):
    """Return the new score of a player taking into account the Fuzzy Factors rule.
    """
    # BEGIN PROBLEM 4
    "*** YOUR CODE HERE ***"
    if hog_gcd(score,100) > 10: # gcd greater than 10 is fuzzy number
```

```
154
155         return score+((hog_gcd(score,100)//10 )%10)*2
156
157     elif hog_gcd(score,100) <= 10:
158         return score
159
160
161     # END PROBLEM 4
162
163
164 def fuzzy_update(num_rolls, player_score, opponent_score, dice=six_sided):
165     """Return the total score of a player who starts their turn with
166     PLAYER_SCORE and then rolls NUM_ROLLS DICE, *including* Fuzzy Factors.
167     """
168     # BEGIN PROBLEM 4
169     "*** YOUR CODE HERE ***"
170     """if player_score==fuzzy_points(player_score):
171         return roll_dice(num_rolls,dice)+player_score
172     else:
173         return player_score + roll_dice(num_rolls,dice)+fuzzy_points(player_score)
174     """
175     player_score=simple_update(num_rolls, player_score, opponent_score, dice)
176
177     player_score=fuzzy_points(player_score)
178
179     return player_score
180
181     """if num_rolls==0: #previous one
182         #temp=simple_update(num_rolls, player_score, opponent_score, dice)
183         temp1= fuzzy_points(player_score)
184         return temp1
185
186     elif fuzzy_points(player_score)==player_score:
187         return simple_update(num_rolls, player_score, opponent_score, dice)
188
189     elif fuzzy_points(player_score)!=player_score:
190          return simple_update(num_rolls, player_score, opponent_score, dice)
    +fuzzy_points(player_score)
191     """
192
193     # END PROBLEM 4
194
195
196 def always_roll_5(score, opponent_score):
197     """A strategy of always rolling 5 dice, regardless of the player's score or
198     the opponononent's score.
199     """
200     return 5
201
202
203 def play(strategy0, strategy1, update,
204          score0=0, score1=0, dice=six_sided, goal=GOAL):
```

```
    """Simulate a game and return the final scores of both players, with
    Player 0's score first and Player 1's score second.

    E.g., play(always_roll_5, always_roll_5, fuzzy_update) simulates a game in
    which both players always choose to roll 5 dice on every turn and the Fuzzy
    Factors rule is in effect.

    A strategy function, such as always_roll_5, takes the current player's
    score and their opponent's score and returns the number of dice the current
    player chooses to roll.

    An update function, such as fuzzy_update or simple_update, takes the number
    of dice to roll, the current player's score, the opponent's score, and the
    dice function used to simulate rolling dice. It returns the updated score
    of the current player after they take their turn.

    strategy0: The strategy for player0.
    strategy1: The strategy for player1.
    update:    The update function (used for both players).
    score0:    Starting score for Player 0
    score1:    Starting score for Player 1
    dice:      A function of zero arguments that simulates a dice roll.
    goal:      The game ends and someone wins when this score is reached.
    """
    who = 0  # Who is about to take a turn, 0 (first) or 1 (second)
    # BEGIN PROBLEM 5
    "*** YOUR CODE HERE ***"
    while score0 < goal and score1 < goal:
        if who%2==0: #player 0 turn
            #num_rolls=strategy0(score0,score1)
            #score0=simple_update(strategy0(score0,score1),score0,score1,dice)
            score0 = update(strategy0(score0,score1),score0,score1,dice)

        else : #who%2==1 : #player 1 turn
            #num_rolls=strategy1(score1,score0)
            #score1=simple_update(strategy1(score1,score0),score1,score0,dice)
            score1 = update(strategy1(score1,score0),score1,score0,dice)
                #score1=fuzzy_update(simple_update(strategy1(score1,score0),dice))
                #score1=simple_update(strategy1(score1,score0),score0,score1,dice)
        who+=1
    return score0, score1


    # END PROBLEM 5

"""if update == fuzzy_update:
            if who==0: #player 0 turn
                score0=fuzzy_update(strategy0(score0,score1),score0,score1,dice)
                who=who+1

            elif 1-who==1 : #player 1 turn
                score1=fuzzy_update(strategy1(score1,score0),score0,score1,dice)
```

```
            who=who-1"""

######################
# Phase 2: Strategies #
######################


def always_roll(n):
    """Return a player strategy that always rolls N dice.

    A player strategy is a function that takes two total scores as arguments
    (the current player's score, and the opponent's score), and returns a
    number of dice that the current player will roll this turn.

    >>> strategy = always_roll(3)
    >>> strategy(0, 0)
    3
    >>> strategy(99, 99)
    3
    """
    assert n >= 0 and n <= 10
    # BEGIN PROBLEM 6
    "*** YOUR CODE HERE ***"


    def strategy(a,b): # get two arguments

        return n

    return strategy


    # END PROBLEM 6


def catch_up(score, opponent_score):
    """A player strategy that always rolls 5 dice unless the opponent
    has a higher score, in which case 6 dice are rolled.

    >>> catch_up(9, 4)
    5
    >>> strategy(17, 18)
    6
    """
    if score < opponent_score:
        return 6  # Roll one more to catch up
    else:
        return 5


def is_always_roll(strategy, goal=GOAL):
    """Return whether STRATEGY always chooses the same number of dice to roll
```

```
    given a game that goes to GOAL points.

    >>> is_always_roll(always_roll_5)
    True
    >>> is_always_roll(always_roll(3))
    True
    >>> is_always_roll(catch_up)
    False
    """
    # BEGIN PROBLEM 7
    "*** YOUR CODE HERE ***"
    first_play = strategy(0,0) # we start we 0,0
    for score in range(goal): # possibility of score of player
        for opponent_score in range(goal): # possibility of score of opponents
            # now we check if stratey value is not equal to first game, whenever they aren't same then False, to catch up.

            if strategy(score,opponent_score) != first_play: # such as catch up.
                return False
    return True # always True until goal, for example (99,99) still go on

    # END PROBLEM 7


def make_averaged(original_function, total_samples=1000):
    """Return a function that returns the average value of ORIGINAL_FUNCTION
    called TOTAL_SAMPLES times.

    To implement this function, you will have to use *args syntax.

    >>> dice = make_test_dice(4, 2, 5, 1)
    >>> averaged_dice = make_averaged(roll_dice, 40)
    >>> averaged_dice(1, dice)  # The avg of 10 4's, 10 2's, 10 5's, and 10 1's
    3.0
    """
    # BEGIN PROBLEM 8
    "*** YOUR CODE HERE ***"
    # END PROBLEM 8
    # i=0
    # total=0
    # def averaged_dice(dice,total_samples):
    #     while i<total_samples:
    #         total+=dice()

    #     return total/total_samples
    # return averaged_dice(make_test_dice,total_samples)

    def averaged_function(*args):
        average_total = 0
        for x in range(total_samples): # For loop,
            average_total = average_total + original_function(*args)
        return average_total / total_samples
```

```
360
361     return averaged_function
362
363
364 def max_scoring_num_rolls(dice=six_sided, total_samples=1000):
365     """Return the number of dice (1 to 10) that gives the highest average turn score
366     by calling roll_dice with the provided DICE a total of TOTAL_SAMPLES times.
367     Assume that the dice always return positive outcomes.
368
369     >>> dice = make_test_dice(1, 6)
370     >>> max_scoring_num_rolls(dice)
371     1
372     """
373     # BEGIN PROBLEM 9
374     "*** YOUR CODE HERE ***"
375     max_averaged_score=0
376     max_num_roll_location=0
377     for num_rolls in range(1,11):
378         averaged_rolls = make_averaged(roll_dice, total_samples)
379         averaged_score = averaged_rolls(num_rolls,dice)
380         if averaged_score > max_averaged_score:
381             max_averaged_score = averaged_score
382             max_num_roll_location =num_rolls  # the maximum roll happens on num_rolls in the for
    loop, the location should be updated
383     return max_num_roll_location
384
385
386     # END PROBLEM 9
387
388
389 def winner(strategy0, strategy1):
390     """Return 0 if strategy0 wins against strategy1, and 1 otherwise."""
391     score0, score1 = play(strategy0, strategy1, fuzzy_update)
392     if score0 > score1:
393         return 0
394     else:
395         return 1
396
397
398 def average_win_rate(strategy, baseline=always_roll(6)):
399     """Return the average win rate of STRATEGY against BASELINE. Averages the
400     winrate when starting the game as player 0 and as player 1.
401     """
402     win_rate_as_player_0 = 1 - make_averaged(winner)(strategy, baseline)
403     win_rate_as_player_1 = make_averaged(winner)(baseline, strategy)
404
405     return (win_rate_as_player_0 + win_rate_as_player_1) / 2
406
407
408 def run_experiments():
409     """Run a series of strategy experiments and report results."""
410     six_sided_max = max_scoring_num_rolls(six_sided)
```

```
    print('Max scoring num rolls for six-sided dice:', six_sided_max)

    print('always_roll(6) win rate:', average_win_rate(always_roll(6)))  # near 0.5
    print('catch_up win rate:', average_win_rate(catch_up))
    print('always_roll(3) win rate:', average_win_rate(always_roll(3)))
    print('always_roll(8) win rate:', average_win_rate(always_roll(8)))

    print('boar_strategy win rate:', average_win_rate(boar_strategy))
    print('fuzzy_strategy win rate:', average_win_rate(fuzzy_strategy))
    print('final_strategy win rate:', average_win_rate(final_strategy))
    "*** You may add additional experiments as you wish ***"


def boar_strategy(score, opponent_score, threshold=12, num_rolls=6):
    """This strategy returns 0 dice if Boar Brawl gives at least THRESHOLD
    points, and returns NUM_ROLLS otherwise. Ignore score and Fuzzy Factors.
    """
    # BEGIN PROBLEM 10
    if boar_brawl(score,opponent_score) >= threshold:
        return 0
    else:
        return num_rolls

     # Remove this line once implemented.
    # END PROBLEM 10


def fuzzy_strategy(score, opponent_score, threshold=12, num_rolls=6):
    """This strategy returns 0 dice when your score would increase by at least threshold."""
    # BEGIN PROBLEM 11
    temp = score
    if fuzzy_update(0,score,opponent_score)-temp >= threshold :
        return 0
    else :
        return num_rolls  # Remove this line once implemented.

    # END PROBLEM 11


def final_strategy(score, opponent_score):
    """Write a brief description of your final strategy.

    *** YOUR DESCRIPTION HERE ***
    """
    # BEGIN PROBLEM 12
    return 6  # Remove this line once implemented.
    # END PROBLEM 12


##########################
# Command Line Interface #
##########################
```

```
463
464 # NOTE: The function in this section does not need to be changed. It uses
465 # features of Python not yet covered in the course.
466
467 @main
468 def run(*args):
469     """Read in the command-line argument and calls corresponding functions."""
470     import argparse
471     parser = argparse.ArgumentParser(description="Play Hog")
472     parser.add_argument('--run_experiments', '-r', action='store_true',
473                         help='Runs strategy experiments')
474
475     args = parser.parse_args()
476
477     if args.run_experiments:
478         run_experiments()
479
```