CPSC 231 ASSIGNMENT #2

The Game of Pig

Pig is a very simple, but fun dice game invented by John Scarne in 1945.[1] It is normally played by two players, and has an interesting jeopardy decision component during each player's turn. A modern incarnation of the game called *Pass the Pigs* published by Winning Moves®, which uses a fun pair of custom pig dice rather than a standard six-sided die, is shown on the right.

The rules of Pig are very simple. Two players take turns trying to accumulate points, and the winner is the first player to reach 100 points. At the beginning of a player's turn, a separate turn score is started at zero, and the turn proceeds as follows:

1. The player makes a decision whether to *roll* (going to step 2) or to *hold* (going to step 3).

2. If the player decided to roll, he or she rolls a single six-sided die. If the roll is a

   1 – Pig out! The player scores nothing this turn, the turn ends, and the opponent's turn begins.

   2, 3, 4, 5, or 6 – The roll is added to the turn score, and the player continues his or her turn at step 1.

3. If the player decided to hold, the current turn score is added to the player's total score and his or her turn ends.

Players can toss a coin to decide who goes first. The first player to end their turn at or above 100 total points is the winner.

This assignment will be your first use of the Python programming language in this class to implement a version of the game that can be played within the console window on your computer. Your program will simulate rolling the die, keep track of the rules of the game, and even play a simple artificial intelligence (AI) against a human player.

Due Dates

Individual component	Friday, September 28, 11:59 PM
Paired component	Friday, October 5, 11:59 PM

[1] John Scarne. *Scarne on Dice*. Wilshire Book Co., 8th edition, 1992

Individual Component

The individual component of this assignment consists of a bunch of "warm-up" exercises for you to do get used to writing tiny programs in Python. You will likely find your solutions to these individual problems useful later on when you team up to develop the Pig game with your partner. You can use the exact same programming environment you set up for Assignment #1 to complete this assignment. Create and submit a separate Python program for each problem.

Problem 1: Rolling a Die

Write a Python program to simulate rolling a single six-sided die. Print the result of the roll to the terminal.

Example Problem 1 output:

```
- rolled a 3
```

Inputs: None.

Outputs: The result of your die roll.

Problem 2: Expected Value

If you study probability theory, you may think of the outcome of a die roll as a random variable. The *expected value* of the random variable long-term average or mathematical expectation of what the outcome would be if you repeated the random process many times. Write a Python program to compute the expected value of a six-sided die roll by simulating many rolls and calculating the average outcome. Use a command line argument to specify the number of simulated trials to perform.

Example Problem 2 invocation:

```
$ python3 my-p2.py 1000
```

Example Problem 2 output:

```
Rolling 1000 times...
Estimated expectation: 3.481
```

Inputs: The number of rolls to simulate as a command line argument.

Outputs: Your estimation of the expected value of a die roll, presented in a form similar to that shown on the right.

Problem 3: Expected Rolls

You might also wonder, on average, how many die rolls you can expect to make in a turn of Pig before you "pig out". Write a program to simulate rolling a die and counting the number of rolls before a 1 comes up. Then use the same simulation method from Problem 2 to estimate the expected number of turns before pigging out.

Example Problem 3 invocation:

```
$ python3 my-p3.py 500
```

Example Problem 3 output:

```
Simulating 500 turns...
Estimated expectation: 6.432
```

Inputs: A command line argument specifying the number of turns to simulate.

Outputs: Your estimation of the expected number of rolls before pigging out.

Problem 4: Pepys's Problem

One way of estimating the likelihood of a random event occurring is to observe how many times that particular event occurs, then divide that by the total number of events. For example, if you want to estimate the probability of a coin toss landing head-side up, you could toss a coin 100 times, count how many times it comes up "heads", then divide that by 100.

Use this strategy to solve Creative Exercise 1.3.40 from your text, which states:

> In 1693 Samuel Pepys asked Isaac Newton which is more likely: getting a 1 at least once when rolling a fair die six times or getting 1 at least twice when rolling it 12 times. Compose a program that could have provided Newton with a quick answer.

Inputs: None.

Outputs: The answer to Pepys's question, and some evidence to support it. Your program may print something like this:

```
Estimated likelihood of 1 once in 6: ⟨some probability⟩
Estimated likelihood of 1 twice in 12: ⟨some probability⟩
Therefore, ⟨your conclusion⟩
```

Paired Component

We encourage you to work with a partner to complete your full implementation of the Pig game. You may choose your own partner, but remember that you must work with a *different partner* for each assignment in this class! If you are solving these problems as a pair, one submission is sufficient for both students.

The five problems in this component are designed as a progression to lead you toward the final, playable Pig game. Think of it like your previous exercises in stepwise refinement, but here we've helped you to do some of the top-down design already.[2] We're getting you to implement and test solutions to specific subproblems, from simple to complex, then assemble them to create your Pig game.

Again, create and submit a separate Python program for each problem. You will likely want to reuse fragments of code between your solutions, or use your solution for a problem as the starting point for the next. Both are good ideas, but if you find errors later on, remember to go back and fix your previous code before submitting.

[2] Although we do not require you to hand this in for your assignment, you may find it helpful to create a plan for your Pig game program using stepwise refinement. Look at the Pig game as a whole, and see if you can break it down into subproblems yourself. Doing this exercise may help you to see how you can organize all the pieces you've already developed into a concise and descriptive full program.

Problem 5: The Computer's Turn

First, we'd like to program the computer to be able to play a turn of pig. We wouldn't want the computer to keep rolling until it pigs out every turn of course, because it would never score any points!

A very simple and good strategy for Pig is to keep rolling until you accumulate 20 or more points in the turn, then hold and keep that as your turn score. (If you look at the problems from the individual component, you might guess why.) Write a program that simulates a single turn of Pig with the computer using this strategy to play.

Inputs: None.

Outputs: Print the outcome of each die roll as it happens, then print the final score for the computer player at the end of the turn.

Example Problem 5 output:

```
 - rolled a 5
 - rolled a 6
 - rolled a 6
 - rolled a 4
Turn score = 21
```

Another sample run:

```
 - rolled a 3
 - rolled a 2
 - rolled a 1
Pigged out!
Turn score = 0
```

Problem 6: The Computer's Strategy

We can make a little tweak to the strategy from Problem 4 to make it do a lot better. For example, if we start the turn with 98 total points, there's really no reason to try to get 20 points because we only need 2 points to win! Write a program that plays a turn of pig where, given the initial score, it uses the strategy of holding at 20 points or after accumulating enough points to reach 100. This will be our final computer AI's strategy.

Inputs: The current total score entered via a prompt.

Outputs: Print the outcome of each die roll as it happens, then print the turn score and total score for the computer player.

Example Problem 6 output:

```
Enter current score: 90
 - rolled a 2
 - rolled a 6
 - rolled a 6
Turn score = 14
New total score = 104
```

Problem 7: Solitaire Pig

With the computer AI complete, we can test it out by having it play a full game of Pig by itself. Write a program that simulates a full one-player game, played by the just the computer, starting at a total score of 0 and playing turn after turn until it reaches 100 points.

Inputs: None.

Outputs: A turn-by-turn transcript of the computer playing Pig by itself until it reaches the goal of 100 points.

Example Problem 7 output:

```
 - rolled a 6
 - rolled a 1
Pigged out!
Turn score = 0
New total score = 0
 - rolled a 1
Pigged out!
Turn score = 0
New total score = 0
 - rolled a 5

...

 - rolled a 4
Turn score = 23
New total score = 90
 - rolled a 4
 - rolled a 5
 - rolled a 5
Turn score = 14
New total score = 104
```

Problem 8: Computer vs. Computer

The game of Pig was really meant for two players, so we'll make that happen now. First, we'll have both players controlled by the computer, and both using the same strategy as before. Write a program that simulates two computer AIs playing Pig against each other. Name the players "Player One" and "Player Two", or other names of your choosing, to distinguish them.

Your program should randomly choose which player goes first. It should keep a total score for each player, and alternate turns between the computer players until one player ends its turn with a score of 100 or higher. Print the outcomes of each roll that occurred during

each turn, and at the end of each turn, print a summary of the total scores to that point in the game.

Inputs: None.

Outputs: A turn-by-turn transcript of two computer players playing Pig against each other.

Problem 9: Player vs. Computer

Finally, it's time to create the interactive game of Pig where you get to play a full game against the computer. Replace one of the computer players from your previous program with an interactive prompt for a human player's input. After each roll during the human player's turn, ask whether the player wants to `[r]`oll or `[h]`old.[3]

Print a full transcript of the game playing out, just as you did with the previous problem. Play a few games against your computer AI when you're done, just to make sure everything works. Can you beat the computer? Can you think of a better strategy for the computer?

Inputs: A roll or hold instruction from the human player via interactive prompt, after every roll during the player's turn.

Outputs: A turn-by-turn transcript of the game unfolding as you play Pig against your computer artificial intelligence.

Bonus Problem: Big Pig

Big Pig is a high-stakes variation on the original game of Pig.[4] The play proceeds as described at the beginning of this handout, but a player rolls two six-sided dice instead of one. The turn is scored according to the following roll outcomes:

Either die (but not both) shows a 1: Pig out! The player scores nothing this turn, the turn ends, and the opponent's turn begins.

Both dice show a 1: Twenty-five (25) points are added to the turn score, and the player continues.

Both dice show 2, 3, 4, 5, or 6: Rolling a double fetches double the points. Twice the sum of the dice are added to the turn score (*e.g.* two 3's will score 12 points), and the player continues.

Neither die shows a 1: The sum of the dice is added to the turn score and the player continues.

The other major difference in the Big Pig variant is that the game is played in *innings*. An inning consists of each player playing a complete turn, always in the same order. A coin toss decides which of the two players will be the "first" player. Then the "top" half, or *frame*,

Example Problem 8 output:

```
Player One's score: 0
Player Two's score: 0
It's Player One's turn
 - rolled a 1
Pigged out!
Total turn score = 0

Player One's score: 0
Player Two's score: 0
It's Player Two's turn
 - rolled a 6
 - rolled a 2

...

Player One's score: 85
Player Two's score: 88
It's Player One's turn
 - rolled a 2
 - rolled a 6
 - rolled a 2
 - rolled a 4
 - rolled a 3
Total turn score = 17

Final score: 102 vs 88
Player One wins!
```

[3] Since the most frequent response will likely be to roll, you can make it so that an empty response (*i.e.* pressing enter) instructs the game to roll again.

[4] Complete rules for this variation, as described by Skip Frey, are presently available at `http://cs.gettysburg.edu/projects/pig/frey-rules.txt`

of the inning is the first player's turn, and the "bottom" half is the second player's turn.

Like the base game, the player that reaches 100 points first wins, but every Big Pig game must be played to the end of the inning. If at the end of the final inning, both players have 100 or more points, the player with the higher score wins. Thus, if the first player reaches a total score of 102 at the top half of an inning, the second player still has an opportunity to beat it, and if he or she scores 103 points or higher, then the second player wins the game. If the scores are tied at the end of the inning, extra innings are played until the tie is broken.

Write a program that allows you to play Big Pig interactively against a computer opponent with an AI strategy that you design. At minimum, the output should be similar to that of the basic Pig game from Problem 9. Feel free to embellish the interface as you'd like, to make for an improved playing experience.

To earn this bonus, *your computer opponent must be able to beat your Teaching Assistant consistently at Big Pig!* This bonus problem is more about designing an AI strategy for the computer to play Big Pig than it is about implementing the specific rules of the Big Pig variant itself. How you design this strategy is the interesting part of the problem, and is entirely up to you!

Inputs: A roll or hold instruction from the human player via interactive prompt, after every roll during the player's turn.

Outputs: A turn-by-turn transcript of the game unfolding as you play Big Pig against your computer artificial intelligence.

The Big Pig Tournament

If you've completed the Big Pig bonus, you can enter your computer AI into our CPSC 231 Big Pig Tournament to compete for prizes and even more bonus credit![5] We will play all eligible computer opponents against each other in a round-robin schedule. The entries with the most wins from the round robin games will then play in a single-elimination tournament to determine the 2018 Big Pig Champion.

[5] Odds of winning will depend on the total number of entries received.

To enter, submit a Python (`.py`) file with functions that match *exactly* the definitions shown in the listing below. You may include other functions needed to support your `roll_again()` function, but do not include any global code in this file (such as your Big Pig game from the previous problem) unless you need something to be run to initialize your AI algorithm. When your `roll_again()` function is called, it must make a decision (roll or hold) and return it within *five milliseconds* (0.005 s), or you may be disqualified for taking too long!

If, perhaps for your own convenience, you would like to submit the same file for both the Big Pig bonus problem and the tournament, you can use the Python convention of putting your global code inside a conditional block of the form

```
if __name__ == "__main__":
    # your global code (e.g. game simulation)
```

This technique is described in more detail in Chapter 2.2 of your text, and feel free to ask your instructors if you're unsure.

my-big-pig-entry.py

```
def my_name():
    """
    Informs the referee what your team or AI should be called.
    Please limit the name to 20 characters or less.

    :return: A string representing your name.
    """
    return "Player One"

def roll_again(turn_score, my_score, opponent_score, inning_frame):
    """
    Make a decision whether to roll or to hold, given the full
    state of a two-player Big Pig game. Meanings of parameters
    passed to you are documented below.

    :param turn_score: Points accumulated thus far in the turn.
    :param my_score: Your total score in the game so far.
    :param opponent_score: The opponent's total score so far.
    :param inning_frame: 0 for top of the inning, 1 for bottom.
    :return: True if you wish to roll again, False to hold.
    """
    # your code here to decide your computer player's strategy
    return False
```

Java Bonus

We'd like to encourage you to become a multi-lingual computer scientist, and to take a language-agnostic approach to computational problem solving. You may also know that you will be working in the Java programming language if you were to continue on to CPSC 233. Thus, we will award you a bonus if you submit correct Java programs for Problems 5–9 in this assignment, in additional to your Python programs.[6]

You may set up your Java programs however you'd like, as long as it is convenient enough for your TA to run them on the CPSC lab machines. There exists a version of our course text that uses the Java programming language, titled *Computer Science: An Interdisciplinary Approach*.[7] It has corresponding support libraries that can be found at https://introcs.cs.princeton.edu/java/home/. If you're not sure where to start with Java, you may find it most convenient to follow the instructions from that booksite. You may also use functionality from their Java Booksite Library if that helps you to create Java-language equivalents of your Python programs.

[6] Note that you should consider your Python programs to be your de facto solutions for this assignment. You will not get credit for solving these problems unless your Python solutions are correct.

[7] Robert Sedgewick and Kevin Wayne. *Computer Science: An Interdisciplinary Approach*. Addison-Wesley, 2016

Submission

After you have completed this assignment, you should have written a total of nine Python programs, saved as `.py` files: four for the individual component and five for the paired component. Please ensure that your files are named descriptively, with the problem number included, so that your TA can easily see which program is associated with each problem.

Use the University of Calgary Desire2Learn system[8] to submit your assignment work online. Log in using your UofC eID and password, then find our course, CPSC 231 L01 and L02, in the list. Then navigate to Assessments → Dropbox Folders, and find the folder for Assignment #2 here. Upload your programs for problems 1–4 in the individual folder, and your other programs in the paired folder (if your partner hasn't already done so). One submission will suffice for each pair of students.

[8] http://d2l.ucalgary.ca

If you are using one or more of your grace "late days" for this assignment, indicate how many you used in the note accompanying your submission. Remember that late days will be counted against *both* partners in the paired component. If you completed any of the bonus problems, please indicate that here as well.

Credits

The game of Pig was originally invented by John Scarne,[9] as previously indicated. To the best of our knowledge, the use of the Pig dice game to teach computer science was pioneered at Gettysburg College. The problems appearing in the paired component of this assignment are derived from originals posed by Todd Neller.[10]

[9] John Scarne. *Scarne on Dice*. Wilshire Book Co., 8th edition, 1992

[10] Todd W. Neller, Clifton G. M. Presser, Ingrid Russell, and Zdravko Markov. Pedagogical possibilities for the dice game Pig. *Journal of Computing Sciences in Colleges*, 21(6):149–161, 2006