A Little Guidance on Programming Assignment #3

The State of the Game

Zombie Dice has an *observable* environment. The state of the game is known by both players. This information is all that is available for each player to make specific action decisions. The state of the game in Zombie Dice has many components. These include:

- the number of brains eaten by each player
- which player is currently taking a turn
- the number of brains collected this turn by the current player
- the number of blasts collected this turn by the current player
- if the current player has decided to roll again or end the turn
- the dice remaining in the cup (including their colors)

This information is all contained in member variables in the `State` class.

The `State` class also offers a `payoff` method for evaluating states. This method returns the *utility* of the current state if that state is a *terminal state* (i.e., the game is over). This utility value is bounded:

$$(-\texttt{State.win payoff} = -100) \leq U(s) \leq (\texttt{State.win payoff} = +100)$$

If the given state is *not* a terminal state, then the `payoff` method calls a *heuristic* evaluation function of your design, and the value that the function returns is taken as an estimated expected utility of the non-terminal state.

When performing game tree search, there is not sufficient time to search all possible futures all the way to the end of the game. Instead, the provided implementation of the *minimax algorithm* searches the game tree to a fixed depth, and then calls the `payoff` method on the states corresponding to the game tree nodes at that depth.

In this way, the `payoff` method provides the (estimated) utility of nodes at the maximum depth. The minimax algorithm must then back these values up the tree to calculate the *expected utility* of intermediate nodes in the tree. Once these values are backed all the way up to the root of the tree, the algorithm will be able to choose the action that corresponds to the *maximum expected utility.*

Roll Outcome Probabilities

There are two sources of randomness in Zombie Dice. First, there is a random draw of dice from the cup. Calculating the expected utility over the random outcomes of this drawing action is already performed by provided code. Second, the rolling of the dice produces random roll outcomes. Calculating the expected utility over the random outcomes of rolls is what you need to do.

Immediately after dice have been drawn from the cup, the state, s, consists of three dice in the hand: {`die1`,`die2`,`die3`}. After rolling, each die can come up one of three ways: {`Brain`,`Blast`,`Feet`}.

You have been provided with code that calculates the conditional probability that the dice roll will result in a given outcome, given the dice in the hand – the current state of the game.

$$P(\texttt{die1, die2, die3 | state = } s).$$

For example, this code could be used to compute:

$$P(\texttt{die1=Brain, die2=Blast, die3=Feet | state = } s),$$

This probability calculation is performed in a method of the `State` class.

The Expected Utility of a State

The *expected utility* of a game state is the sum of the utilities of all of the potential next states, weighted by the probabilities that those states will be next. For the expected utility that you are asked to compute, the game state will always be one in which a handful of (three) dice are about to be rolled. Thus, the possible next states involve all of the possible outcomes of the dice roll. Formally, the expected utility for a state, s, is ...

$$\mathbb{EU}[s] = \sum_{s'} P(s' | \texttt{current state = } s)\ U(s'),$$

... where s' is a game state that results when, starting at game state s, a particular roll outcome (i.e., values for each of {`die1`,`die2`,`die3`}) occurs. Thus, in the equation above, s' iterates over all possible outcomes when rolling the three dice.

Designing a Heuristic Utility Function

The heuristic function that you are to design evaluates a given state of the game, s. That state has many features, as described earlier, including the number of brains that have been eaten by each player. The heuristic function is to provide an educated guess of how likely it is that each player will win the game, communicated as an expected utility. If it is very likely that the computer player will win, then this value should be close to the maximum possible payoff value. If it is very likely that the human player will win, then this value should be close to the minimum possible payoff value. If both players have about an equal chance of winning, at state s, the heuristic value should be close to zero.

The heuristic function must be fast to compute, so it can't involve searching over possible future states from s. How can we estimate how good the state is for each player without looking ahead?

In general, the state is better for the computer player when the computer player has eaten many brains. Symmetrically, the state is better for the human player when the human player has eaten many brains.

If it is the computer player's turn, and the computer player is close to winning, then the expected value of the payoff is quite positive. If, at that point, the computer player has collected no blasts, the expected payoff is greater than if the computer player has collected a couple of blasts. Similarly, if there are mostly green dice left in the cup, it is likely that the computer player will draw dice that will roll well, but if there are mostly red dice left in the cup, the expected payoff is lower. These are examples of features of the state, s, that can tell us something about the expected payoff without performing any look-ahead search. There are many other aspects of the state that could be considered. Exactly which state features you use in your heuristic function and how you translate those features into a numeric expected payoff is up to you. State properties that are more predictive of who will win should have a greater effect on the heuristic value that is returned.

The value returned by the heuristic function should be bounded in the same way that the value returned by the `payoff` method is:

$$(-\texttt{State.win payoff} = -100) \leq U_h(s) \leq (\texttt{State.win payoff} = +100)$$

Can you find a way to quickly calculate a heuristic value, $U_h(s)$, that gives a good approximation of the expected value of the final game payoff, given the current state, s? In general, a heuristic function that provides a better estimate of the expected value of payoff should cause the computer player to make better decisions.