---------------------------------------

ReadMe File - DiceGame in Python

---------------------------------------

\*\* Purpose of the Document \*\*

---------------------------------

This readme file is prepared to explain how I designed my agent to solve dice game and provide details on my approach

and the decisions I have made throughout the design stage.

\*\* General Explanation on the Approach and Algorithm Selected \*\*

----------------------------------------------------------------

As we investigate the behaviour of the dice game, we can easily conclude that we will face with sequence of states with uncertainty in order to play the game. So, this implies that the dice game will be an application of sequential decision making under uncertainty and we can formulate this with Markov decision process (MDP).

In light of the above information, I have decided to implement value iteration algorithm for my agent to produce the best strategy for the dice game.

\*\* Algorithm Test/Development Phase\*\*

---------------------------------------

In my first attempt of the implementation on the algorithm, my agent was able to play the dice game successfully but the time it spent for the construction and takings the actions was considerably high. After a deep analysis on my algorithm, I realised that it was doing the construction phase for each of the actions so spending unnecessary time. So, I changed this, and my algorithm greatly improved.

I also implemented my code to support the extended rules, so my agent can play the game for different types of setup.

\*\* General Structure of the Code \*\*

------------------------------

My code is structured under one class called MyAgent which has the following functions.

\* Function 1: \_\_init\_\_ \*

------------------------------------

Under this function an attribute called self.Vopt is defined and initialized as a dictionary which is the state value function for each state. There is also one line of code here to call superclass constructer.

\* Function 2: Q \*

-----------------------------------------

This function calculates the expected utility of the chance node using the Bellman recursive equation implementation. It also detects terminal state (for any number of dice to support extended rules) and return the reward accordingly.

\* Function 3: iteration \*

-------------------------------

The algorithm in this function does an iteration by looping through all the states and actions iteratively in order to find the optimum utilities for each state using convergence and return them.

\* Function 4: policy \*

--------------------------------------

This function initiates the last operation in order to find the optimal policy for the current state and return the optimum policy.

\* Function 5: play \*

----------------------------

It is the function to return the optimum policy in order for my agent to play the in the best way.

\*\* References \*\*

----------------

Russell, S. and Norvig, P., 2016. Artificial Intelligence: A Modern Approach. 3rd Ed. Harlow: Pearson Education Limited.