**2. For the 1 player strategy:**

In this setting, the rolling of the dice is a random variable, and thus all actions are independent. Since the actions are independent from each other, and there is only the payout to consider (no need to consider cost), the optimal solution to this problem would be using a greedy algorithm, which in this case translates to choosing the action with the highest expected reward.

However, the expected reward (reward multiplied by probability) of all actions is 1:

E(A): 1/6 \* 6 = 1

E(B): 1/3 \* 3 = 1

E(C): 1/2 \* 2 = 1

This implies that in the long run, by the law of Large Numbers, it doesn't matter what strategy we take. As the number of rounds approaches infinity, if we continue to choose the same action, the average number of throws to get to the goal will be equal to the goal divided by the expected reward, 12/1=12, regardless of which action is chosen.

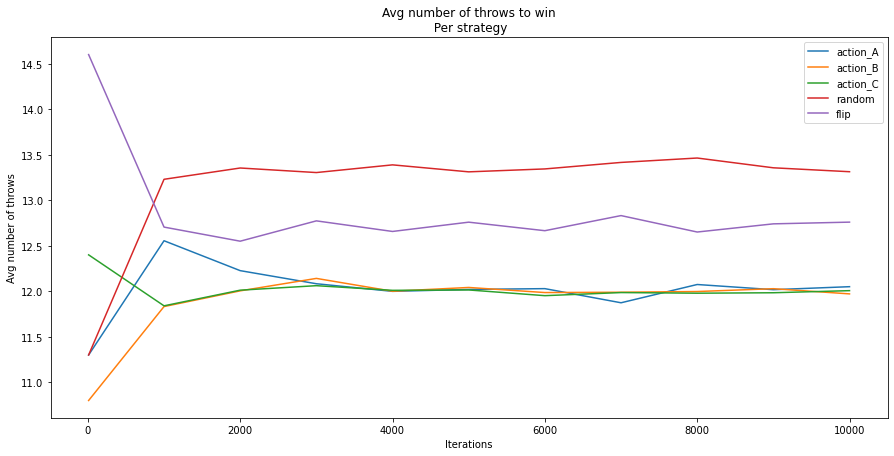
In this sense, if the game is to be played several times then the player can adopt one of the 3 strategies below:

1. Always choose A

2. Always choose B

3. Always choose C

Fig 1. below illustrates this convergence to 12 as the number of trials approaches infinity.

**Fig 1.**

If we consider the time factor, however, the situation changes slightly, as the expected value fails to take into account the cost (here, the time).

A naive solution involves calculating the cumulative probability of winning by sticking to a single solution.

To get to at least 12 points choosing Action A always within 6 rounds, we would need to get 2 rolls of 1. The probability of obtaining at least 2 rolls of 1 within 4 throws is.

The probability of at least 2 rolls of 1 within 6 rounds is:

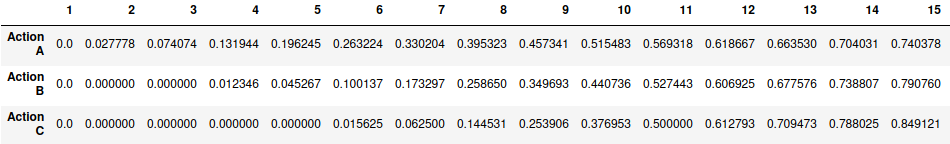
P(1) = 1/6

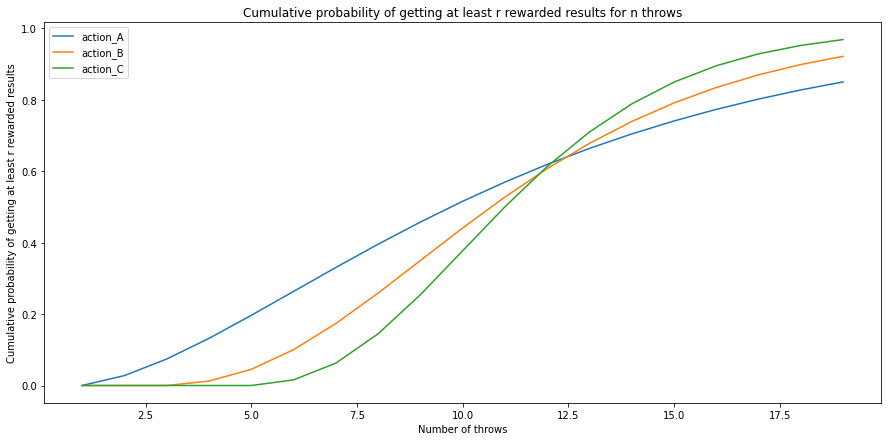
P(~1) = 1 - 1/6 = 5/6

P(X) = binomial(6,2) \* P(1)^2 \* P(~1)^4 + binomial(6,3) \* P(1)^3 \* P(~1)^3 + binomial(6,4) \* P(1)^4 \* P(~1)^2 + ....

P(X) = 0.2009

By calculating the rest of these probabilities we obtain the following table and Figure.

**Table 1.**

**Fig 2.**

According to this table:

In a setting where the player can keep accumulating rewards with no limit of rounds (goal is winning the most, with no cost), then the player should always choose action C.

In a setting where the player is encouraged to achieve his goals fast as possible or the number of rounds is a small number (or where each round has a cost to the player)(goal is winning in the least number of rounds), then the player should always choose action A.

For this reason:

In the 1 player setting the player should always choose action A.

**2. For the 2 player strategy:**

In the 2 player game, the number of throws will not be averaged over the number of rounds. Instead, the winner of each round will get one point. To accumulate the most points and win the game, the player must maximize her/his gain at each time step.

We can model the game as a Markov Decision Process, where the states are defined by the current profit, the set of possible states being S={0,2,3,4,5,6,7,8,9,10,12} and the transitions between states are actions A={A,B,C}. A portion of the MDP is shown in Fig. 3.

C

C

C

B

B

B

C

B

…...

A

**Fig 3.**

We can reframe this problem as a Reinforcement learning problem by setting a ‘reward’ for each state. The reward will be 0 except for the state ‘12’, where the reward will be 1 point.

Now, it suffices to calculate the expected reward at a certain state s for each of the actions A,B,C. Our strategy π will be to choose the action with the highest expected reward based on the current state (profit) (Greedy strategy). The transition probability from state s to state s’ taking action a is defined P(a,ss’) as :

P(a,ss’) = P{st+1 =s’ | st = s, at = a}

We define the expected value of the next reward as Rt  obtained at time t, or alternatively, the reward from taking action a at state s leading to state s’ R(a,ss’)as:

Rt = R(a,ss’) = E{rt+1  | st = s, at = a, st+1 = s’}

We will calculate the expected reward Rt at state st using the reward at state st+1 by choosing action a and a ‘discounted’ reward from the expected reward from future states. In here, we will look ahead a single step (consider all possible transitions for next move only).

Rt = rt+1 + γ Rt+1  = rt+1 + γ rt+1+ γ2 rt+2

We will choose actions that maximize the expected discounted return Rt . We define Qπ(s) as the value of a taking action a at state s in time step t under strategy π in the MDP setting as:

Qπ(s,a) = Eπ{Rt | st = s, at = a} .

At each state s, the optimal action Q\*(s,a) will then be

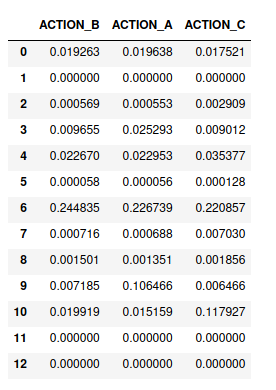
Q\*(s,a) = argmaxπ (Qπ(s,a))

Q\*(s,a) = E{rt+1 + γ argmaxa’ (Q\*(st+1,a’)) | st = s, at = a}

Q\*(s,a) = .

We can compute the Q(s,a) values by using an iterative policy evaluation algorithm. For all states and actions by initializing them to 0 and calculating Q(s,a) state by state. In order to ensure that other actions are explored instead of only the optimal to update their values as well, we choose a random action with probability 0.1 and the optimal action according to the current Q(s,a) table

After a few iterations(10000), we obtain the optimal values Q\*(s,a) represented below in Fig.4



**Fig 4.**

According to this, our strategy will be:

In state 0, choose B

In state 3, choose B

In state 6, choose A

Note that even though this is an optimal strategy, this does not guarantee that a player adopting it will always win (although it does give a high success rate), as it’s still random.

Note that since the probabilities of the rounds are independent there’s no need to consider the other player, just optimize your strategy to win in the least number of rounds possible.