Throughout this document, m = the number of known cards in the deck, n = the number of unknown cards in the opponent's deck, and s = (m! / ((m - n)! \* n!)), which is expressed as c(m, n).  
Also, Pi represents the initial probability of winning (initial). The total symbol should not be confused with the frequently used i lower limit.  
We find the probability of winning using the following equation:

Here, Pk does not represent the probability of the k-th state occurring, as per standard notation. It represents the probability of winning when we give the k-th combination of cards to the player. (I couldn't think of another method to show this.)  
This probability can be either 1 or 0.

Expected value

We need to treat each possible combination as independent events. In the end, since we want to calculate the expected value of drawing a card:

Vk represents the expected value of drawing a card in the assumed situation where the k-th combination is with the player. Vk is:

Here, Pj represents the probability of winning when we draw the j-th card, and Pi represents the probability of winning when no card is drawn. For both, the probability of winning can be either 1 or 0 (independently).

If we write these two equations in expanded form:

Example

Let the cards be dealt, and we have the 10,5 cards, with the player's open card being 10. As a result, the known deck is 2,4,6,7,8,10.  
In this case, s = 15. The k=1 combination is 2,4; the k=2 combination is 2,6; the k=3 combination is 2,7... indicating the cards.  
In V1, since the 2 and 4 cards are removed from the deck, the known deck consists of 6,7,8,10 cards.  
If we want to calculate V1, the upper limit is 6-2, which gives 4. Since Pi is 0 (because the player has 10,2,4 and we have 10,5), the result is 0.

When j=1, the first element of the current deck, which is 6, is considered. Pj is 1 because the player has 10,2,4 and we have 10,5,6.

After these calculations, the total V = 6. Since this result is positive, drawing a card is a reasonable and necessary move.

Bot Parameters:

1. The probability of placing a large bet when the probability of winning is less than 0.5 (0.5 is the critical threshold where losing is expected more than winning). (Pbluff)
2. When deciding to bluff, the money placed should be unpredictable, otherwise, bluffing becomes meaningless.

There is an infinite function that determines which bet amount to choose with which probability; it just needs to satisfy the following integral (mrmax represents the maximum bet increase that can be made):

There are infinitely many functions that satisfy this condition, but for our case, we can use the Gaussian distribution or normal distribution curve.

A graph with colored lines and numbers

Description automatically generated

1. To see the increase in bets, the minimum probability of winning for this parameter must be satisfied when the bet is increased by the player, and

must hold true. (

1. Bet divider : To avoid abuse of the equation explained in point 3, we need to imagine the maximum bet as a line and discretize this line. Otherwise, when the player increases the bet by the maximum possible amount, we need the probability of winning to be 1 to see this, which would be a rare situation.
2. Minimum Winning Probability for Bet Increase: In cases where the probability of winning is high, increasing the bet will be advantageous for us. ()

Game parameters

1. Starting money: The money given to each player at the start of the game. When multiplied by other parameters, it should be large enough not to involve fractions but small enough for easy calculations. (mstart)
2. Blind Bet Ratio The forced money placed at the beginning of each round. (Qblind)
3. Maximum Bet Increase Ratio: The maximum money that can be added to the blind bet. (Qmaxraise)

Game Parameters <---> Bot Parameters

Some bot parameters need to be calculated as a function of the game parameters, otherwise, we will only succeed in games with a single parameter set.

One idea is to calculate Pcmin in an infinite game as 1, in a single-round game as 0, and intermediate values can be calculated using a function.  
When both blind bet and maximum bet increase are 0, Pcmin = 1 (the game will take a long time, and this will work in our favor as it will lower the impact of luck over time), while when both add up to 1, Pcmin = 0 (the game will end quickly, and we certainly don't have the luxury of moving forward cautiously).  
Thus, we can use the function: Pcmin = 1 - (Qblind - Qmaxraise).

Pw  and trees

Another problem to solve is when the bot increases the bet, the opponent draws cards and changes Pw.  
For example, after the cards are dealt, Pw > Prmin might occur, leading to an increase in the bot's bet. However, the issue is that Pw might not stay constant. The player may draw extra cards and drastically reduce Pw.  
For this, we need to calculate Pw for 1, 2, and 3 card draws, but keep in mind that the probability of each player drawing cards might differ. In some deck and open card situations (these two can be referred to as the "state"), the likelihood of drawing 2 or 3 cards is high, whereas in other situations, even drawing 1 card may have a very low probability.  
Therefore, we need to examine how likely it is that additional cards will be drawn in different hands.  
A simple internet search provides us with the statistical analysis of what value players typically stop at. Can we use these values directly in our game? No.  
Because this game has mechanics very different from blackjack, we cannot expect players to behave the same way.  
One method that can be used is to calculate the average value of when the player stops or draws cards until that point. Unfortunately, this method also has flaws.  
If the opponent (the player) is counting cards, they consider not only the cards in their hand but also the current state of the deck.  
However, determining which player is counting cards or how well they can count is perhaps the most difficult or even impossible part of this program.

At this point, the problem that needs to be addressed is: which winning probability to use: the current winning probability or the one predicted according to human psychology. This dilemma is not easily solvable.  
If we act according to the current winning probability, our whole plan (bet increase, withdrawal, etc.) could fail if the opponent draws cards. In this case, acting according to human psychology might be the right approach, but the opponent is not a rational actor. Predicting their moves is very difficult and unreliable.  
So, is it logical to assume that the opponent is also a completely rational actor like us?

This approach also brings serious problems. Humans cannot calculate situations with millions of combinations. Assuming they can and will act accordingly will exaggerate the opponent's abilities and lead to losing many rounds by withdrawing when we could have won.

Still, since I can't think of another way, I accept that the opponent will draw cards with specific probabilities based on certain hand values as the only option.

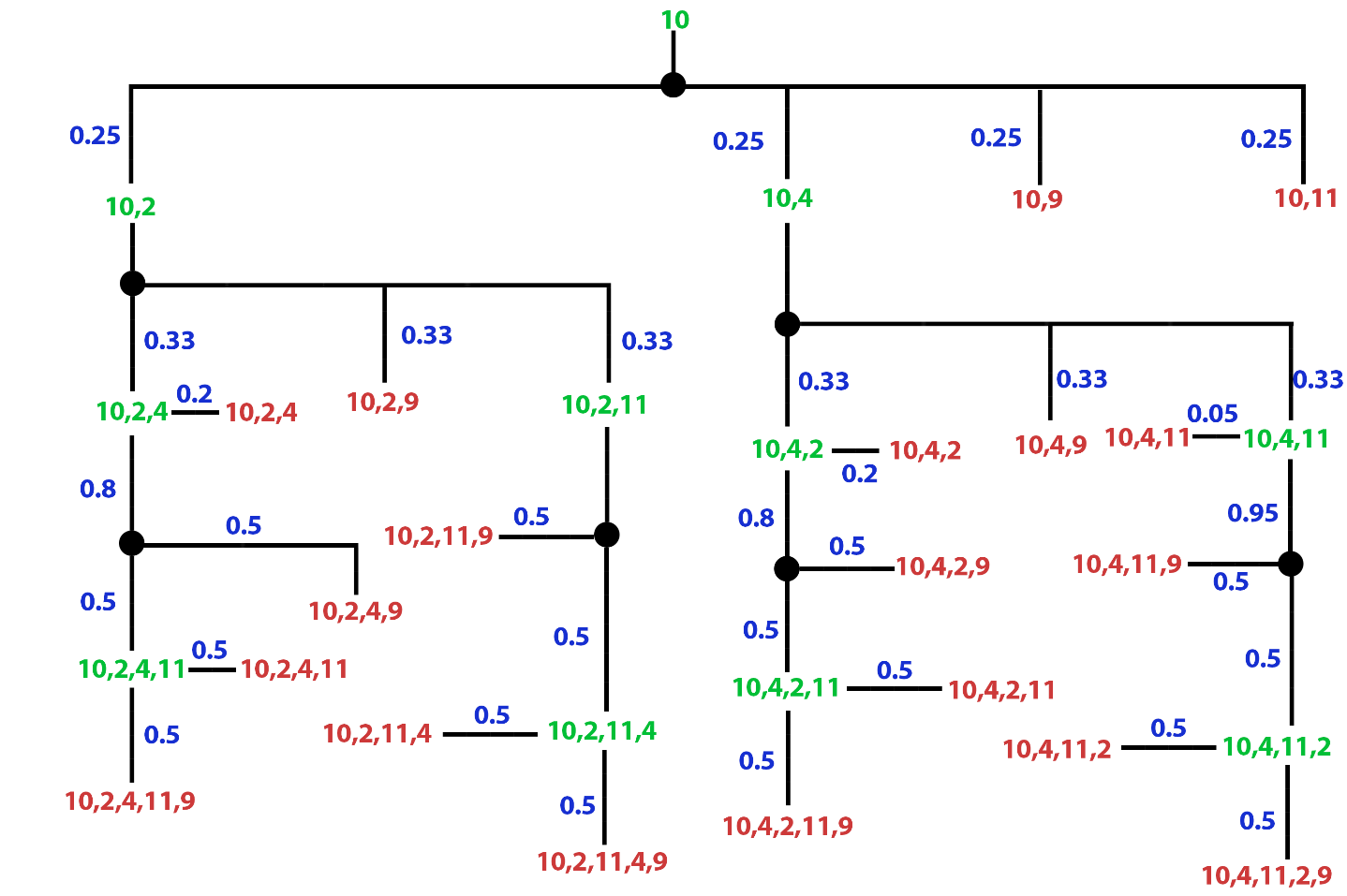
For this, we will discuss 3 different winning probabilities:

* Instant Winning Probability (Psimple)
* Predicted Winning Probability (Passumed)
* Complex Winning Probability (Pcomplex)

To calculate the complex winning probability, we need to determine the weight we assign to the predicted winning probability. Let’s call this weight Wassumed.  
Thus, our complex winning probability becomes: [(1 - Wassumed) \* Psimple + Wassumed \* Passumed].

Example:

In the game deck, there are cards 2,4,7,9,10,10,11 (Ace). The bot has the cards 10,7, and the player's open card is 10.  
To calculate the predicted winning probability, we first need to create a tree:



Thus, the calculated probability values are:

0 -> 0.305208  
15 -> 0.00416667  
16 -> 0.0333333  
17 -> 0.0739583  
18 -> 0  
19 -> 0.25  
20 -> 0  
21 -> 0.333333

Multiplying the draw scenario by 1/2, we get Passumed = 0.4164.  
If we want to calculate just the current winning probability, we need to find the possible combinations based on the known and unknown cards.  
10,2 10,4 10,9 10,11  
In this case, Psimple = 2/4, which is 1/2.  
Assuming Wassumed is 0.2, we get Pcomplex = 0.4832 for this scenario.