The Game of Wordn’t

Joshua Cortez

We’ll introduce the Game of Wordn’t, and then describe how a Wordn’t bot can be implemented. We’ll exploit some properties of the Wordn’t game along the way.

What is Wordn’t?

What is Wordn’t? It’s a game invented at Cobena. (It was introduced by an intern and it caught on I guess…) It’s played with multiple people, usually 4-6 players, but there’s no hard limit on the number of players. Players take turns in a circle.

The first player starts by choosing any letter in the alphabet. For each succeeding turn, a player adds a letter either to the start or the end of the string. Players must be careful in their choice of letter and position. While adding letters, they should be careful not to form an actual word from the given string. They should also make sure that the string actually builds up to an actual word.

If however, the player makes any of those 2 violations, it’s the succeeding player’s responsibility to challenge the previous player. The challenge may take 2 forms.

1. The current string is an actual word.
    - The current string is checked against a given wordlist. This wordlist is set and pre-agreed among the players. This can be any wordlist, but a popular choice is the official Scrabble words. If the current string is actually in the wordlist, then the player who was challenged loses. Otherwise, the player who challenged loses.

2. The current string doesn’t build up to any word.
    - The player who was challenged must provide a word in the wordlist where he/she was intending to “build towards.” In other words, the current string should be a substring of the given word. If valid word is provided by the player who was challenged, then the player who challenged loses. Otherwise, the player who was challenged loses.

Let’s describe an example of a game with 4 players. Let’s say the first player chooses “A”. The second layer chooses C and puts it at the beginning, forming “CA”. The third player chooses “B” and puts it at the end, forming “CAB”. The fourth player then challenges the third player for forming a word. The third player loses since “CAB” is in the wordlist.

Here’s another example. Players are using the Scrabble wordlist. The first player chooses “Z”. The second player then puts “X” at the end, forming “ZX”. The third player then challenges the second player. The second player is forced to verify if there is a word in the wordlist that contains “ZX”. No matter how the second player responds, they list because there are no words in Scrabble that contain “ZX”.

A single *round* of Wordn’t ends when a player fails to respond to a challenge by the succeeding player. The player who failed to respond is the loser of the round. A single *game* of Wordn’t consists of multiple rounds. This could be any number, for example a game may consist of 100 rounds. For each round, the order of players (i.e. whose turn is first, second, etc.) is randomized for fairness. At the end of all the rounds, players are ranked based on who has the least number of total losses.

Considerations for a Wordn’t Bot

Let’s look at how the dynamics of the game change if we play the game with bots instead of with humans. When humans play Wordn’t, the fun in the game rests on knowledge of what words are possible. But when bots play the game, they can have full access to the entire wordlist.

Here are the possible actions of a player during his turn. With a bot, some actions become trivial.

1. Challenging that the current string is already a word.
    - This is very easy for a bot. They’ll just lookup the current string during their turn.
2. Challenging that the current string cannot build towards a word.
    - It’s still very easy for a bot. It can loop over all the words in the wordlist and check if the current string is a substring.
3. Add a letter to the current string.
    - This is not trivial. This is the most interesting aspect of developing a Wordn’t Bot. We’ll dive deeper on how to make a bot be good at this.

Overall Strategy

Let’s outline a general strategy for adding a letter to the current string. (Note that this includes the very first turn, and for that, the current string is the empty string.)

1. During my turn, could I find a way of adding a letter, such that I will **never be forced to form a word during my turn, or in any of my next turns?** Let’s call this a sure-win move.
    - How could I know if a sure-win move is possible? If it is possible, how could I know the letter and the position (whether at the start or at the end of the current string)?
2. If a sure-win move is not possible, could I find a way that maximizes the probability of not losing?

As it turns out, the best strategy for Wordn’t boils down to carefully optimizing for ratios. Two ratios are at heart of the Wordn’t bot: the Basis Word Ratio (BWR), and the Nonhalting Intersection Ratio (NIR). We’ll start by defining some building blocks of Wordn’t. We’ll use those building blocks to define the two ratios. Using those ratios, we’ll devise detailed strategies.

Building Blocks

We’ll start with describing the building blocks of BWR and NHI. We start with the fundamental building blocks of the game, and progressively derive more complex pieces.

Current String

This is the string received by the player during their turn.

Proposal String

One of 52 possible new strings by adding a letter either to the end of the current string or at the start of the current string. The 52 proposal strings are all the possible actions of the player, assuming that they don't challenge.

Wordlist

The list of words pre-agreed among the players. This can be any list of words, but a popular choice is the official Scrabble words.

Basis Words

Basis words are a subset of the word list that is a "basis" for a string (regardless of whether it's a current string or proposal string). They are the set of words that the player may be building towards. It's possible to check the basis words for any string (could be an empty set). More concretely, all the words that can have the string as a substring are basis words. A valid word (i.e. in the wordlist) is a basis word of itself.

For example, if the string is "BA", then possible basis words are "BAD", "BADMINTON", "BABY", and "LOWBALL". It's possible that some strings **don't** have basis words. If the current string during your turn has no basis words, you should challenge it!

Halting Words

Halting words are a subset of the word list that lead to almost certain loss (and consequently ending the game). This set of words depends on 3 things: the player's position number, the number of players, and the words in the wordlist.

Halting words are those whose length modulo the number of players is equal to the player's position number. More concretely, for player in position i, in a Wordn't game of n players, halting words words are those whose length l satisfies the equation $l \; mod \; n = i$.

Suppose you're the second player in a game of 6 player Wordn't where the maximum number of characters is 15 and there aren't 2 character words. Then, all words that are 8 characters long and 14 characters long are halting words.

Halting Basis Words

Here we construct something useful out of basis words and halting words. Let's remember strategy #1: *"During my turn, could I find a way of adding a letter, such that I will* ***never be forced to form a word during my turn, or in any of my next turns?"*** This is akin to checking all the possible basis words for all 52 proposal strings. The best proposal, if it exists, is one that will never force you to form a word during your turn, or in any of your next turns.

And it's from there that we define halting basis words. A halting basis word is a word you could build towards, but it can set you up to lose. If you choose a proposal with a halting basis word, there's a chance that the other players can build towards that. Consequently, in one of your next possible turns, you can be forced to lose.

Let’s be more precise in defining halting basis words. We can break it down into 3 main cases.

1. Exact Case
    - Basis word is also a halting word. By definition, this is a word that you could build towards but can lead set you up to lose.
    - This depends only on your position number.
2. Prefix Case
    - This occurs when the string is a prefix of a halting word, and the halting word is a prefix of a basis word. The basis word with these properties is a halting basis word.
    - In this case, the only way to build towards the basis word is by forming the halting word first.
    - Example: Let’s say you’re in the first position in a game of 6 player Wordn’t. This means that 7 letter words are halting words. VOLCANOES is a possible basis word for the proposal V, however, VOLCANO is a halting word. Since V is a prefix of VOLCANOES and VOLCANO is a prefix of VOLCANOES, then VOLCANOES is a halting basis word.
    - This case depends on both a position number and a string.
3. Suffix Case
    - This is the mirror image of the prefix case. This occurs when the string is a suffix of a halting word, and the halting word is a suffix of the basis word. The basis word with these properties is a halting basis word.
    - Just like in the prefix case, the only way to build towards the basis word is by forming the halting word first.
    - This case depends on both a position number and a string.

Nonhalting Basis Words

These are the kind of basis words that work to your advantage. Given a player position and a string, **nonhalting basis words are the basis words that are not halting basis words**. Basis words depend only on a string, but nonhalting basis words depend on both a string and the player position. This is because halting words depend on the player position.

For a given player position and a string, the set of halting basis words is the complement of the set of nonhalting basis words. The union of those 2 sets is the set of basis words.

Basis Word Ratio (BWR)

Now we’re ready to define the basis word ratio. Given a string s and player position i, the basis word ratio is defined as

$$BWR_{s,i} = \frac{\#\ of\ nonhalting\ basis\ words}{\#\ of\ basis\ words}$$

This metric ranges from 0 to 1. and it’s a good way of choosing the best proposal.

Strategy to use BWR

During your turn, compute for the BWR of all 52 proposals. There are 4 possible scenarios.

1. Sure-Win Proposal
   - You find a proposal with BWR = 1
   - This is a sure-win (or rather sure non-loss) because BWR = 1 means that all the basis words for that proposal are nonhalting. No matter what basis words the other players choose to build towards, you'll never be forced to lose.
2. Imminent Loss
   - All proposals have BWR = 0
   - This means you're heading towards a dead end of imminent loss. All basis words for all proposals are halting basis words. No matter what proposal you choose, the other players can only build towards words that will force you to lose.
   - The only way of not losing in this case is if any other player fumbles by giving a proposal that doesn't have any basis words. To take advantage of this possibility, we can implement a stalling strategy. We choose the proposal that has the highest average basis word length. The rationale is basis words with more characters, on average, means longer stalling time. With a longer stalling time, there's a higher chance of another player fumbling.
   - Note that this can occur the first time you have a turn, in which case your opponents already set you up to lose before you had a chance to respond. This makes some player positions more disadvantageous than others. There's no foolproof strategy in Wordn't.
3. Challenge Sure-Win
   - BWR is undefined for all proposals, because the number of basis words is always zero.
   - This means that the previous player has made a mistake by giving a proposal that doesn't have any basis words. Challenge the previous player (current string doesn't build towards a word), and they will surely lose.
4. Inconclusive Scenario
   - The highest BWR (among all proposals) is less than 1 but greater than 0.
   - There's no assurance of a win or a loss, and the game's outcome will depend on the other players' choices. This is an interesting case, and the appropriate strategy will depend on the assumptions of how the other players are playing. We shall tackle this in more detail in the next section.

Assumptions and Strategies for the Inconclusive Scenario

In the inconclusive scenario, the outcome will depend on the other players' choices. Let's explore some assumptions on the other players' behavior.

1. Naïve Opponents
   - In this assumption, opponents are only concerned if they are building towards a word in the wordlist. They are indifferent among proposals, as long as the proposal has basis words.

- With this assumption, the BWR can be interpreted as a probability of winning (or not losing). Following the proposal during your turn, the opponents are randomly traversing through basis words until the round ends. This is equivalent to a random draw among the basis words of your proposal. You can only hope that the random draw happens to be a nonhalting basis word.
- The best strategy in this case would be to choose the proposal with the highest BWR. You would have the highest probability of winning.

2. Careful Opponents
    - Here, opponents are aware of their halting basis words. They will prefer proposals where they don't have any halting basis words.
    - Using BWR won't be the best strategy. Using BWR won't be the best strategy. Why? For naïve opponents, the game becomes a random draw among the basis words of your proposal. But for a careful opponent, some of the basis words are more likely to be "drawn", because they are more advantageous for the opponent.

For naïve opponents, using the BWR is sufficient. The overall strategy is to maximize the BWR. For careful opponents however, the BWR is not enough. We'll need to define a metric called the average nonhalting intersection ratio (or AVG NIR). We'll describe the NIR, and AVG NIR in the next section. Then we'll describe how to use it along with the BWR to deal with careful opponents.

Nonhalting Intersection Ratio (NIR)

In the previous section, we were considering opponents who were aware of their halting basis words. Using NIR is about taking advantage of the opponents' behavior by using it against them.

Let's now define NIR. During player i's turn, he/she chooses a proposal string s. The next player, player $i+1$, can choose proposal string s'. The NIR is defined for i, s, and s'. We'll abbreviate nonhalting basis words as NHBW. Player i is abbreviated as P_i. With $|.|$ denoting the cardinality of a set (i.e. number of elements in the set).

$$NIR_{s,i,s'} = \frac{|(NHBW\ of\ P_i\ for\ proposal\ s) \cap (NHBW\ of\ P_{i+1}\ for\ proposal\ s')\ |}{|\ NHBW\ of\ P_{i+1}\ for\ proposal\ s'\ |}$$

The NIR is also a ratio that ranges from 0 to 1. The numerator contains an intersection of two sets. This intersection is the crux of the metric. If NIR is equal to 1, all of the next player's nonhalting basis words ***are also*** your nonhalting basis words. After the next player's turn, no matter what the other opponents propose, they can only choose to build towards any of your nonhalting basis words. In the *careful opponent* scenario, this is a sure-win for you!

But there's still something missing. In the NIR, we assume that we know the next player's proposal s'. But we don't know that in advance! This brings us to the AVG NIR.

Average Nonhalting Intersection Ratio (AVG NIR)

Unlike the NIR, which is defined for i, s, and s', the AVG NIR is defined for only i and s. This is more practical, because you don't have to know the next player's proposal s'. Instead of anticipating the possible s', you can enumerate all the possible s'.

The AVG NIR takes the average of the NIR across all the possible s'. Let's call the set of all possible s' as S'. The AVG NIR is defined as follows.

$$AVG\ NIR_{s,i} = \frac{1}{|S'|} \sum_{s' \in S'} NIR_{s,i,s'}$$

Note that S'should not include proposals whose basis words are all halting, because NIR will be undefined (i.e. have a zero denominator). A careful opponent will not choose such a proposal anyway. Since there are at most 52 possibilities, $|S'| \leq 52$.

There are several ways of defining S'. It may not necessarily just be the 52 possibilities excluding the proposals with undefined NIR. These different ways of defining S'correspond to different ways of assuming how the careful opponent behaves.

1. Sure-Win Proposals Only
    - Opponent will only choose among proposals where BWR = 1.
    - In other words, $S' = \{all\ s'\ such\ that\ BWR_{s',i+1} = 1\}$
    - Note however, that it's possible for $S'$to be empty. In that case, they can resort to the second case below.
2. Any Non-Losing Proposal
    - Opponent will only choose among proposals where BWR > 0.
    - In other words, $S' = \{all\ s'\ such\ that\ BWR_{s',i+1} > 0\}$
    - In the edge case that that $S'$is still empty, AVG NIR will be undefined.
3. Maximize BWR
    - Opponent will choose proposal/s that maximizes BWR.
    - In the case that there are multiple proposals that maximize BWR, the opponent will be indifferent to those s'.
4. Recursive Optimization
    - Opponent is choosing proposals also by computing for the AVG NIR. Opponent is also anticipating what the succeeding player will choose and will use it against them. Opponent is also choosing the best proposal based on BWR and NIR.
    - The level of recursion may vary. For example, the next player, P_{i+1}, may assume that P_{i+2}will be anticipating P_{i+3}, that P_{i+2} will be anticipating P_{i+3}, and so on. It can also be that the recursion stops at one level, with P_{i+1}anticipating P_{i+2}, but no further than that. This can get computationally expensive quickly.
    - The set S'will be dependent on the outcome of the recursion.

For our purposes, let's assume that the careful opponent acts in according to #1 and #2. That is, they choose sure-win proposals if there are any. If there aren't any sure-win proposals, they choose randomly among the non-losing proposals. Now S' is clearly defined as we outline our overall strategy.

Strategy for Careful Opponents

This summarizes all the steps during your turn.

1. Before anything else, compute for the BWR among all 52 possible proposals. The flow follows from the "Strategy to Use BWR" section.
    a. If there is a proposal with BWR = 1, choose that proposal, and you win the round.
    b. If all proposals are BWR = 0, then stall and hope for the best. Stall by choosing the proposal with the highest average basis word length.
    c. If BWR is undefined for all proposes (i.e. denominator is zero), then challenge the previous player.
    d. If the highest BWR among all proposals is less than 1 but greater than 0, then we will need to use NIR. This is the "inconclusive scenario".
2. If we're in the inconclusive scenario, we first need to know all next opponent's possible proposals. To accomplish that, for each of your proposals with nonzero BWR, do the following.
    a. Define all the 52 proposals for the next player.
    b. Compute for the BWR for all of the 52 proposals of the next player.
    c. Define S' using the careful opponents assumptions #1 and #2. S' is the set of all possible proposals of the next player.
        i. $S' = \{all\ s'\ such\ that\ BWR_{s',i+1} = 1\}$. S' may be empty.
        ii. If S' is empty, then $S' = \{all\ s'\ such\ that\ BWR_{s',i+1} > 0\}$
    d. Compute for the AVG NIR based on S'.
3. Choose the proposal with highest AVG NIR.
    a. What does it mean when AVG NIR = 1? All of the next player's options are also advantageous to you. It's a sure-win for both you and the next player.
    b. Even if the highest AVG NIR < 1, using the AVG NIR is still better than BWR because the careful opponent's behavior is baked into the AVG NIR.

A very simplified of the strategy is as follows: "Can I find a sure win proposal? And if I can't, can I "entice" the next player to choose a proposal that works for both of us?"

Appendix

Proposal Windows

A bot following the strategy above can perform the same action given the same current string. This is perfectly reasonable in some cases. For example, there might only be one proposal with BWR = 1 for current string s. However, there can be cases where there are multiple proposals with BWR = 1 for current string s. To keep the game interesting, we can introduce some randomness via proposal windows. There are 2 main cases.

1. There are multiple sure-win proposals.

- Occurs with BWR = 1, or when AVG NIR = 1.
- Your proposal window are all the sure-win proposals. Randomly choose among the sure-win proposals.

2. The highest ratio is less than 1 but greater than 0.
    - This applies to the BWR if naïve opponents, and to the AVG NIR if careful opponents.
    - Define a score threshold t. It should be reasonably low. For example, $t = 1\%$
    - Construct a proposal window by finding all proposals whose ratio is within t from the highest ratio. Randomly choose among these proposals.

Computational Considerations

Most of the computational heavy lifting in the Wordn't bot is in finding basis words and halting basis words. They both depend on substring matching. While there are several approaches, one approach is by using the Aho-Corasick algorithm. This algorithm is typically used for keyword matching. Given a set of keywords and input text (imagine a long string), the Aho-Corasick algorithm can simultaneously find the locations of the keywords (assuming there are matches), within the input text.

How can this algorithm be used in Wordn't? During player i's turn, he/she has 52 possible proposals $s_1, s_2, \dots s_{52}$. The player should also know their halting words. By treating the proposals as keywords and using the concatenated wordlist as the input text, we can enumerate the basis words for each of the $s_1, s_2, \dots s_{52}$ using the Aho-Corasick algorithm. Let's call the set of basis words for proposal s_j as B_j.

We can again apply the Aho-Corasick algorithm to find the halting basis words per proposal s_j. We can treat the halting words as keywords, and the concatenated elements of B_j as the input text. With this, we have the a set of candidate halting basis words per proposal s_j, let's denote this as C_j.

The C_j are only candidates because all the algorithm did was find basis words with a halting word substring. Halting basis words require more specific conditions for the substring match (refer to the exact, prefix, and suffix cases discussed above). However, the algorithm did most of the heavy lifting, so the additional compute for post-processing should be minimal. After additional post-processing on C_j, we have enumerated the halting basis words per proposal s_j.

Further work on a more efficient implementation can be done by using a game tree. A game tree is a directed graph where the nodes are positions and the edges are moves. In Wordn't, the game tree can be constructed at the start of the game, since the word list is already known.