Jeremy Miesch

CSCI 3202

Spring 2025

AI Spymasters in Codenames

I. Project Overview

In my final project, I trained an AI spymaster for the word association board game codenames (rules here). The goal was to create a pygame popup where you can play on a team with an AI spymaster, while also providing an option to simulate many games and test the effectiveness of the AI spymaster vs a random spymaster.

I chose to implement the random spymaster by having it pick a random word on the codenames board and then find the n most similar words on the board based on cosine similarity (where n could be 2, 3, or 4). The practice did not place any consideration on which teams each word belonged to, and was not punished for incorrect guesses.

The AI spymaster used an alpha beta to determine the most optimal clues, with the utility function being the difference in scores between the two players after n plies or until the game finished. The score was calculated as follows:

1. +1 point to whichever team a guessed word belongs to
2. -0.5 point if a neutral word is guessed
3. -5 points if the death word is guessed

It randomly sampled a slew of potential clues it could give from a list of the words belonging to its team, and then determined the clue that would abide by the minimax. In an effort to improve time efficiency and accuracy, I also added a 'pre check' in the get_clue_combos method. This method checked whether each randomly sampled clue would likely be a failure due to proximity to the death word or an opponent's word, and only considered the best clue candidates in the actual alpha beta.

In codenames, the player who moves first is assigned an extra word to combat an advantage. In my implementation, I instead ensured that both players get the same number of moves every game, so, if player 1 wins, then player 2 gets an extra "redemption" round to balance the advantage.

II. Data and models

I used an online dataset of codenames words, called codenames_words.txt in my zip, for the program to draw from (extracted from here and slightly modified down to 389 words). It was important to have different vectorizer models for the spymaster and the guesser to avoid 'perfect' guessing accuracy (in simulations - in the pygame window the guesser is the user), and so I used glove-wiki-gigaword-100 for the spymaster and word2vec-google-news-300 for the guesser. I chose these models specifically because they are both well-established embeddings trained on large corpora, but they differ enough in their training data and embedding strategies to cause some miscommunication between the guesser and spymaster.

III. Results

At 5 plies, the program was too slow to simulate a full 5x5 board at any significant number of repetitions. However, I was able to get 500 games for 3x3 and 4x4 boards at 5 plies, as shown below.

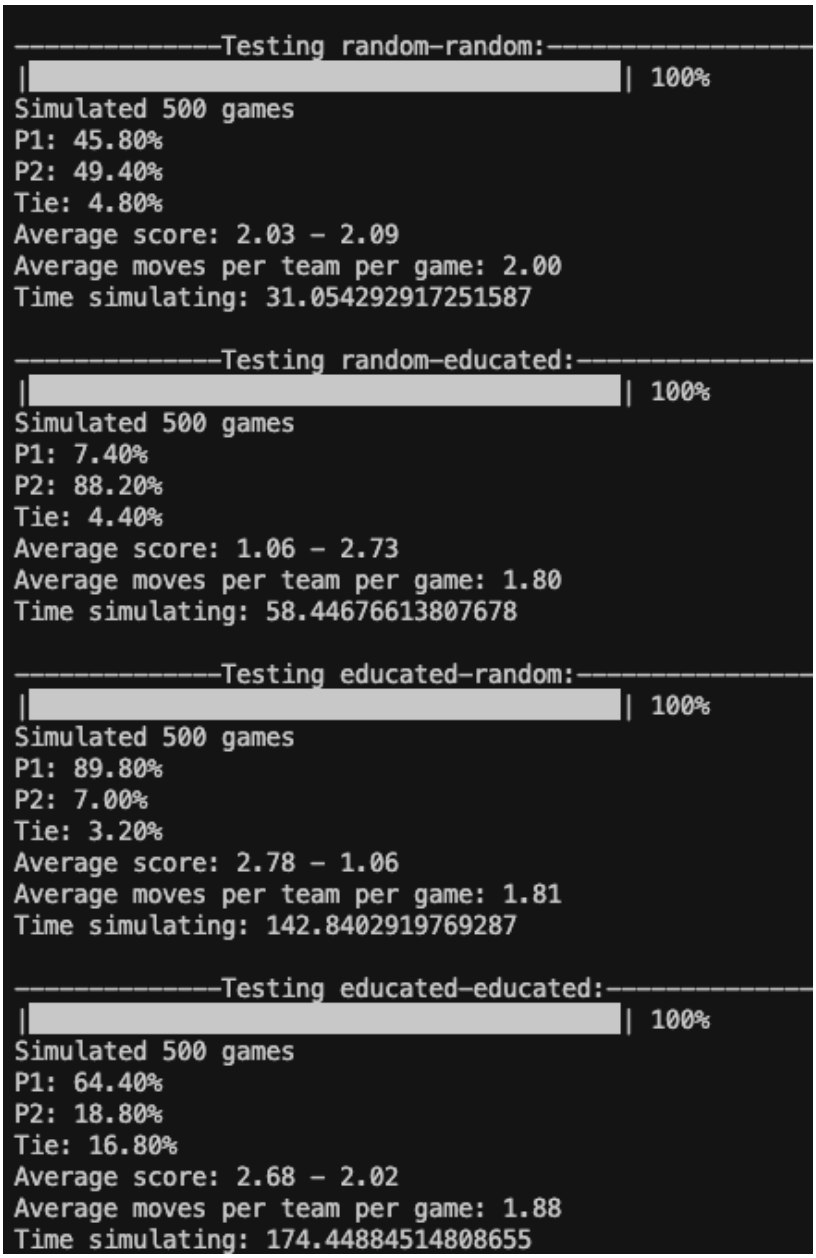

Figure 1: 500 simulations, 3x3 board, 5 plies, no death word

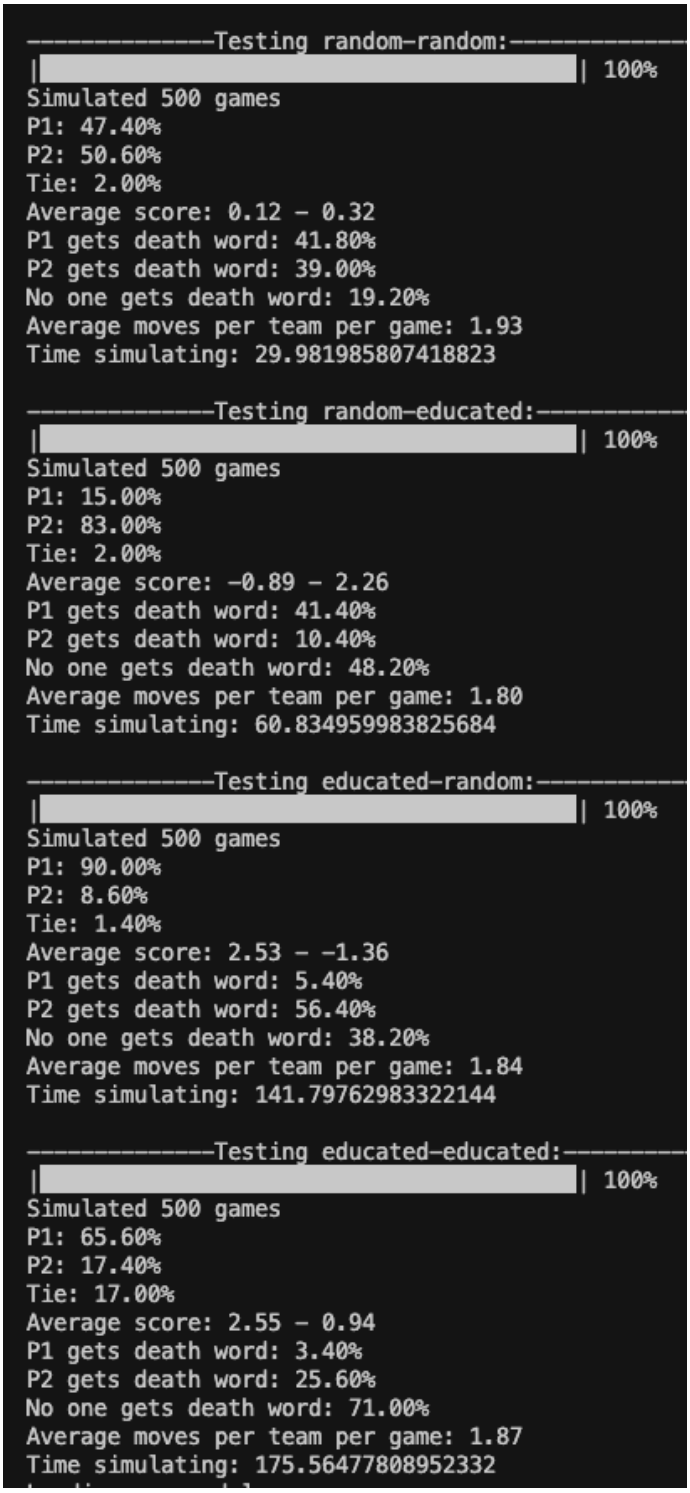

Figure 2: 500 simulations, 3x3 board, 5 plies, yes death word

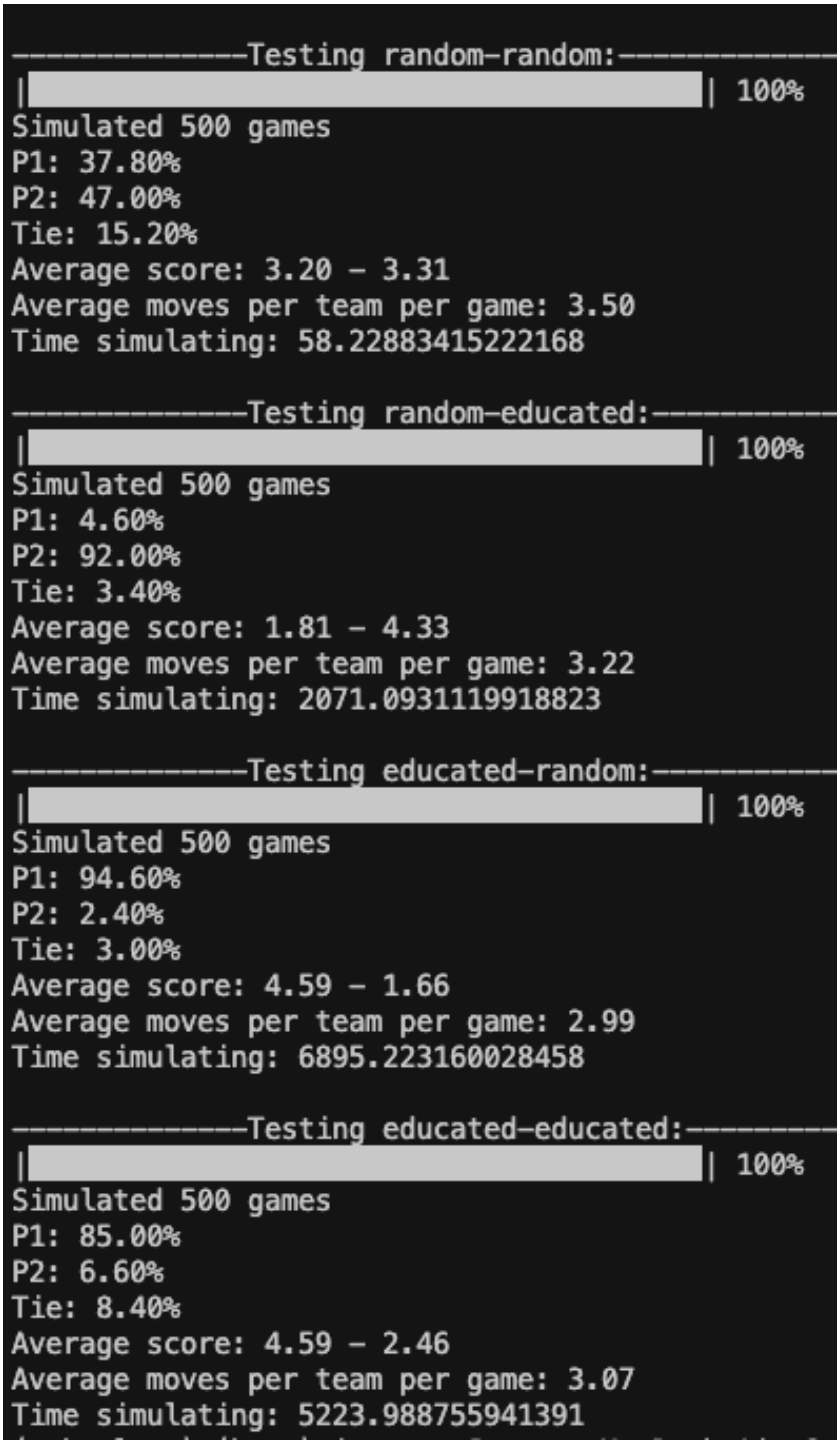

Figure 3: 500 simulations, 4x4 board, 5 plies, no death word

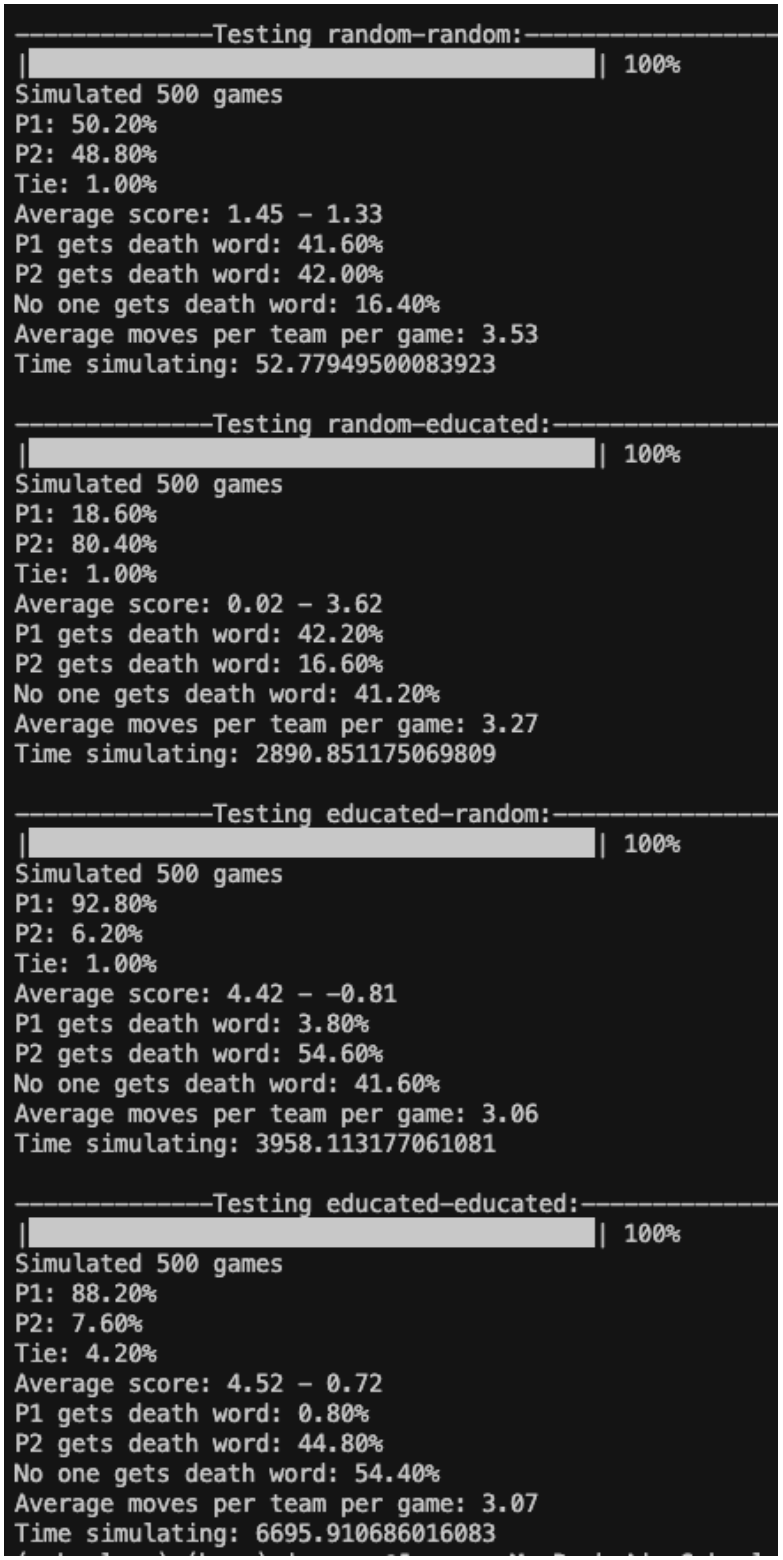

Figure 4: 500 simulations, 4x4 board, 5 plies, yes death word

Death Word?	Board Size	P1 Win %	P2 Win %	Tie %
NO	3x3	45.8	**49.4**	4.8
	4x4	37.8	**47**	15.2
YES	3x3	47.4	**50.5**	2
	4x4	**50.2**	48.8	1

Table 1: Summary of results for **random-random**

Death Word?	Board Size	P1 Win %	P2 Win %	Tie %
NO	3x3	7.4	**88.2**	4.4
	4x4	4.6	**92**	3.4
YES	3x3	15	**83**	2
	4x4	18.6	**80.4**	1

Table 2: Summary of results for **random-educated**

Death Word?	Board Size	P1 Win %	P2 Win %	Tie %
NO	3x3	**89.8**	7	3.2
	4x4	**94.6**	2.4	3
YES	3x3	**90**	8.6	1.4
	4x4	**92.8**	6.2	1

Table 3: Summary of results for **educated-random**

Death Word?	Board Size	P1 Win %	P2 Win %	Tie %
NO	3x3	**64.4**	18.8	16.8
	4x4	**85**	6.6	8.4
YES	3x3	**65.6**	17.4	17
	4x4	**88.2**	7.6	4.2

Table 4: Summary of results for **educated-educated**

IV. Summary of Results

The results largely show what was expected: Table 1 shows a relatively 50/50 split between players, Table 2 shows a significantly higher win percentage for player 2 (the educated player), and Table 3 shows a significantly higher win percentage for player 1 (the educated player). Some other interesting points about the results are as follows:

a. The presence of a death word dramatically reduces the likelihood of a tie, with the exception of adding a death word to a 3x3 matchup between two educated players (Table 4). This makes sense - the presence of a death word causes more scoring variability, especially for random players.
b. A larger board leads to similar results between random and educated players, but dramatically increases player 1's win percentage in a matchup between two educated players.
c. The average number of moves per game decreases when educated players are introduced. This makes sense, as educated players are more likely to guess all their words faster.
d. The educated player's odds of guessing a work are significantly lower than a random player's odds, but player 2 still has a higher chance than expected, even when educated

(17%). Also, player 2 gets the death word much more than player 1 when both are educated (44.8% - 0.8%).

These points all point to the most interesting question from these results: Why does it seem player 1 has an advantage over player 2 when both are educated? This advantage is displayed both by win percentages and (especially) death word guess percentages.

One might think in an educated-educated game that there would be a fairly even winning chance for each player to win if they are guaranteed the same number of moves. Or, if not that, that they would tie fairly frequently as both players are generating great clues. We see a decently high tie % on the smaller 3x3 board in Table 4 (17%), but this decreases with a larger board and an increase in player 1's win percentage to boot. Player 2 even gets a fairly high death word rate against a random player, as shown in Table 2, decreasing its win percentage.

The answer to this question, I believe, is that I underestimated the value of having a full board to generate guesses. Player 2's disadvantage does not come from fewer moves, but rather from the fact that a reduced board state to generate clues from *dramatically* decreases its win percentage. Player 1 can look ahead at moves on its first turn and create a clue that will remove certain words from the board and create a board state where 1) the death word might be semantically closer to the remaining P2 words and 2) the remaining P1 words might be semantically closer to the remaining P2 words. This idea is reflected by P2's high chance of finding the death word, even when it is educated and even when it is playing a random player.

Also, this idea is reflected in the higher win percentage for player 1 in a larger board, by nearly 20%, as shown in Table 4. In a 3x3 board with an average number of moves of 1.87, the alpha beta is not looking ahead 5 plies because the game does not last that long. Therefore, player 1's looking ahead advantage is minimized (although still present). I would expect player 1 to win even more frequently in an educated-educated game on an even bigger board.

V. Conclusion and Further Research

Unfortunately, I did not have enough time to explore an element of this project that was a large reason for me undertaking it in the first place. I wanted to explore different word similarity functions other than simply gensim cosine similarity, but I ran out of time. In the pygame element of this, it is sometimes difficult to discern the connections the AI Spymaster is making between words using their vectorized forms, and it would be interesting to see if more "human" similarity methods (either from extensive human sampling or other similarity metrics) would impact the results.

I also would like to explore the player 1 advantage a little more, confirming it by either switching who plays first every round or measuring the quality of each player's clues in each game. Another possibility for further research would be changing the approach to the guesser model, perhaps making that more "human." I also believe that there are many ways to speed up the code because it is very slow at the moment.

All in all, however, the high win percentages of the educated players in my simulations (>90% for no death word, >80% for death word) renders the project a general success, albeit with room for improvement.