CodeNamer: A Codenames Spymaster AI

CS 701 Final Project, Middlebury College, Fall 2021

Ethan Saxenian
Middlebury College
esaxenian@middlebury.edu

Corey Scheinfeld
Middlebury College
cscheinfeld@middlebury.edu

ABSTRACT

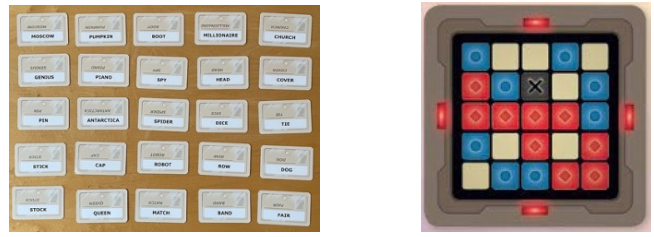
In the game Codenames, one player on each team, designated the role of "spymaster," must provide one-word clues that relate to some number of their team affiliated words in a 5x5 grid. Simultaneously, the spymaster must attempt to avoid relating these clues to neutral words, opposing words, and the "assassin" card so that their partners are able to correctly identify team words. The challenge of finding effective clues motivated us to seek an algorithmic solution that could be utilized by spymasters in order to consistently find the best hints for a given team and board state. Our mobile app acts as this tool. We use image processing and text recognition to allow players to take photographs of the game components, from which words and colors are extracted to produce a virtual representation of the board state. Then, at the spymasters request, an algorithm generates clues for each team using a database of pre-trained word vectors. Our model is able to produce impressive clues that capture nuanced semantic similarities, allowing CodeNamer to act as an effective tool for players of all experience levels.

1 INTRODUCTION

Codenames is a team-based word association game. The game consists of a 5 by 5 grid of words (Figure 1a) and a corresponding key card (Figure 1b), which maps each word to a color. This key is only visible to one player on each team, the spymaster. The spymaster's task is to generate one-word clues that relate to words of their team's color on the board. Their teammates, or "field operatives," must use these clues to guess which words the spymaster is referring to, with the goal of uncovering all the words of their team's color on the grid. Each one-word clue is followed by a number, which indicates how many words the spymaster is referring to, and allots the number of guesses a team may make, plus one extra, per turn. If a team ever selects a neutral or opposing word, their turn ends, and if the assassin word is ever selected, that team immediately loses the game. The team that uncovers all of their allocated words first wins the game.

Often, it can be very difficult for a spymaster to produce effective clues. The challenge lies in relating multiple words of one color while simultaneously disassociating any clues from words of the other team's color, and the assassin word. Racing to to offer high word count clues in order to outpace the competition adds to the difficulty.

In this report we describe our app, CodeNamer, a spymaster tool that can be utilized by players to process images of the 5 by 5 word grid and color key in order to produce valid and meaningful clues. We detail the methods we used, provide an in-depth analysis of the app interface, and finally discuss its performance and limitations.



(a) An example 5x5 Codenames board.

(b) An example key card, which maps each word in the 5x5 grid to a color.

Figure 1: Codenames Game Components

2 PROBLEM STATEMENT

The nature of our goal broke the project into two distinctive parts, utilizing image processing techniques in order to allow users to upload the word grid/color card in their beginning states and developing an algorithm to generate valid clues from these photographs.

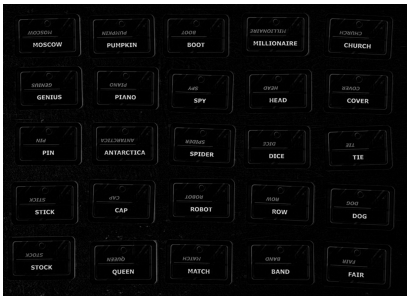
Image Processing

To create a usable and intuitive interface that contributed meaningfully to gameplay, we wanted to allow players to easily upload photographs of their own games to create a virtual board. This eliminates the inconvenience of having to manually type every word and color. To do this, we needed to develop two separate image processing algorithms, one to process an image of the game board, and another to process an image of the color key. These algorithms needed to be both fast, accurate and robust, to account for variations in lighting and create quick results.

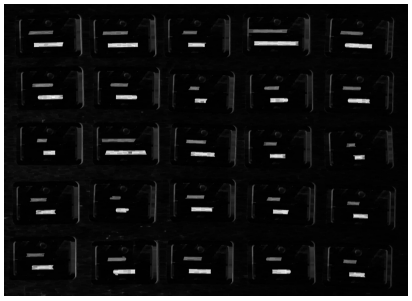
Word2Vec

In order to develop an efficient algorithm to generate valid, effective clues, we decided to use word vectors trained with word2vec [3]. Word2Vec employs two different processes, the Skip-Gram and Continuous Bag Of Words models. Both these approaches train word vectors by considering words and their contexts in bodies of text. With fully-trained word vectors, we would be able to measure semantic similarity between words by calculating the cosine similarity of their associated vectors. Cosine similarity simply computes the cosine of the angle between two vectors in n-dimensional space and is given by the formula

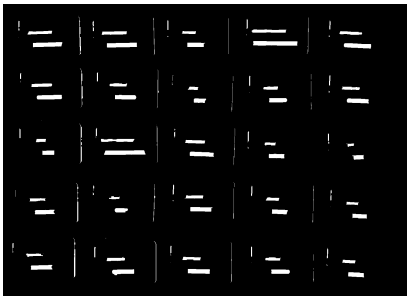
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}.$$



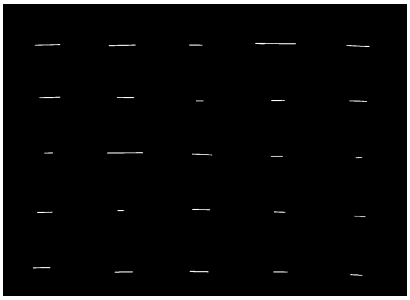
(a) Applying a blackhat morphological operator to reveal dark text against a light background.



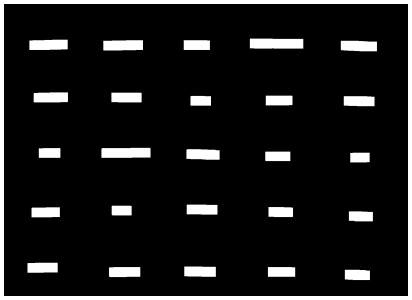
(b) Calculating the Scharr gradient and applying a closing operation to reveal vertical changes in image intensity.



(c) Using Otsu's method to separate pixels into two classes, foreground and background.



(d) Eroding at foreground objects with a 6x6 kernel of pixels to remove flipped text contours and background noise.



(e) The image is dilated with a rectangular kernel of pixels to restore text regions.

(f) Original image, with the resulting text region contours used for cropping and recognition sorted in order.

Figure 2: Text Recognition

3 RELATED WORK

There have been previous efforts to use natural language processing algorithms to generate Codenames clues from a boardstate. Kim et al.[2] employed 3 different frameworks, WordNet, word2vec, and GloVe, and measured the effectiveness of each method in a round-robin tournament of paired spymaster and guesser bots. They found that the vectorial approaches (word2vec and GloVe) performed significantly better than WordNet, which never achieved a win rate above 50%.

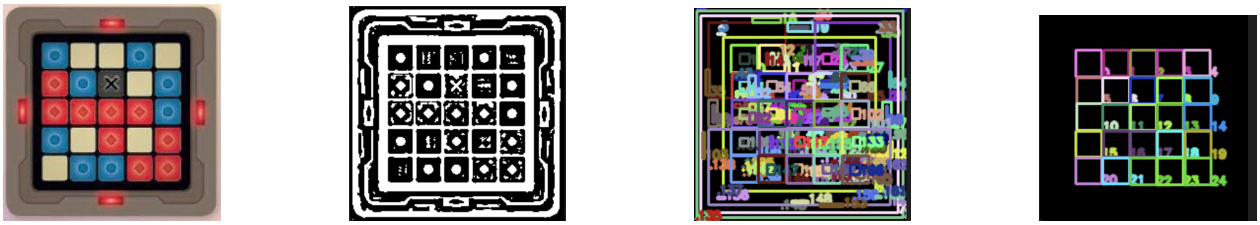
Jaramillo et al.[1] expanded on this work by using 3 new algorithms, TF-IDF, Naive Bayes, and GPT-2 Transformer. In some cases they included negative weights for all the "bad" words on the board. Of the 3 new algorithms, the Transformer produced the best results, while Naive Bayes had the worst results. Additionally, the weighted versions of the algorithms performed better than the unweighted versions. When compared to the word2vec and GloVe bot, the transformer achieved a lower average number of turns to win, and lower percentage of good words flipped. However, the vector bot still achieved a higher win percentage (100%) than the weighted transformer, which won 90% of its games.

Both Kim et al. and Jaramillo et al. sought the best performing model by comparing various natural language processing methods. The goal of our project is not to employ the "most accurate" model, but rather to optimize the clarity and applicability of word vector comparison for human use during a game. This includes the incorporation of image processing meant to allow the technology to be applicable in real-time games, and an interface supportive of human judgement.

4 METHODS

Image Processing

Text Recognition. The unique layout of the Codenames board led to various challenges in the text recognition process. In order to successfully identify the 25 game words in the correct order, we needed to create a consistent text detection algorithm that could both classify text regions in varying lighting conditions and avoid the flipped text that lies inverse to the main text box on a single card. Firstly, with Figure 1a as input, we apply a black hat morphological operation to the blurred, grayscale original image (Figure 2a). Important to this method is the "closing" of an image, or the erosion of pixels from prevalent object boundaries in the image, followed by their dilation, in which pixels are added. The black hat operation is performed by taking the difference between the closing of an image, which removes small objects, and the original input itself in order to reveal light foreground objects that lie against a dark background. Since the card words will always lie in a white text box, the black hat operation applies well. We then compute the Scharr gradient along the x-axis of the black hat image, using the first derivative to reveal vertical changes in image intensity that correspond well to text. Applying a closing operation with a rectangular kernel then closes the gaps between characters (Figure 2b). Next, we use Otsu's method to calculate a single intensity



(a) Original Image.

(b) Adaptive threshold separates pixel clusters into foreground and background classes to create sharp edges.

(c) Contour search reveals continuous edges with geometrical patterns.

(d) k-nearest neighbors algorithm narrows down contours to match grid section pixel-ratios.

Figure 3: Color Recognition

threshold that will separate pixels into foreground and background classes (Figure 2c). At this point, we're beginning to see the outline of our text blocks, but the flipped regions remain prevalent, and the card borders themselves create noise. To deal with this, we use a square kernel, or structure of pixels, to erode the boundaries of prevalent foreground objects until only 25 contours remain in the image (Figure 2d). To restore the size of our text contours, we then dilate the image with a rectangular kernel (Figure 2e). This leaves us with 25 sharp contours that can easily be identified, sorted, and cropped (Figure 2f). Using the Tesseract optical character recognition engine, we then sort and process each text region to identify a word within the set of known Codenames cards to retrieve an ordered board.

Color Recognition. The color card presents a different set of image processing challenges (Figure 3a). In order to successfully extract the color pattern as it corresponds to the game board, we need to identify individual contours within the grid. To do so, we apply an adaptive threshold on a blurred, grayscale image, which calculates an optimal threshold value for small clusters of pixels (Figure 3b). In images with a wide variety of pixel intensities, such as our color grid, this method allows for greater accuracy in identifying foreground objects and creating sharp edges. We then use our threshold image to find contours, or structural outlines with continuous edges, within the grid (Figure 3c). To narrow down these contours, we utilize a k-nearest neighbors algorithm to determine the 25 contours with a pixel area ratio closest to the known ratio of the grid sections (Figure 3d). In other words, we determine the ratio between the area of each contour and the area of the entire grid, comparing this value to the known, measured ratio of a given color card. After doing so, we can assign each contour area a color based off the minimum distance between its average pixel value and the known pixel value of the Codenames colors, and sort the contours by their coordinates to return the correct mapping.

Clue Generation

Data. Our model uses 300-dimensional word vectors that were pretrained on a Google News dataset of 3 billion words. The original word vector database contains word vectors for 3 million words and phrases, however we decided to only use vectors for the 80,000 most common words in the dataset. This ensured that our model was lightweight while also providing enough variety in clue output.

Algorithm. Our clue-generation algorithm is quite simple conceptually. At its core, it calculates cosine similarities between the words on the Codenames board and the rest of the words in our database of word vectors. More specifically, given a group of words from the board, it computes a word vector that is the weighted average of the vectors of the group. Words of the team's color are given positive weights, while words of the other colors are given negative weights, with the assassin word contributing the most negatively. The algorithm follows as such:

(1) A smaller word vector model is created consisting of the 1000 most similar words to the group of positive words only. This helps to remove nonsense words and improves algorithm speed. This smaller model is used in place of the original model throughout the rest of the algorithm.
(2) All combinations of 2, 3, and 4 words from the positive group are obtained and associated weighted word vectors are constructed.
(3) For each grouping of positive words, the cosine similarity is calculated between its weighted word vector and each potential clue in the smaller model.
(4) If a potential clue is not a valid Codenames clue, that is, it is more than one word or a form of any other word on the board, that clue is discarded.
(5) Based on the cosine similarity, the best valid clues for each group size are selected and returned to the user.

Tools

Frontend. Our frontend was developed using React Native, a component-based JavaScript framework for mobile development, and Xcode,

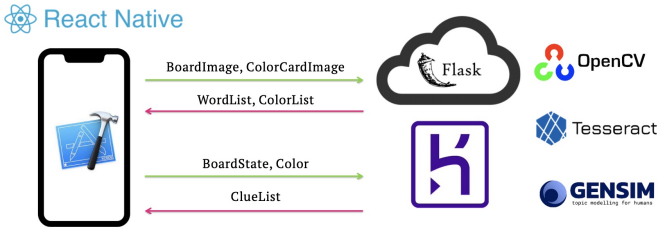
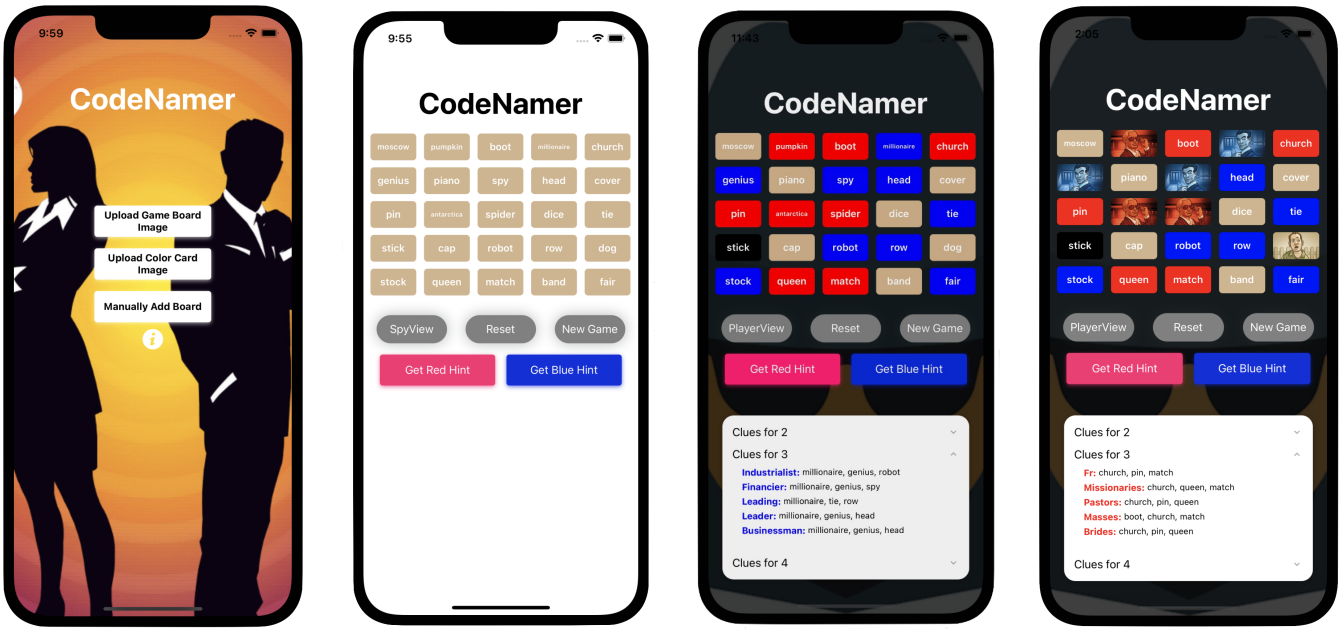


Figure 4: An overview of our tech stack.



(a) CodeNamer home screen. **(b) Player view, with color key hidden.** **(c) SpyView, with blue hints toggled.** **(d) Mid-game board state, with red hints toggled.**

Figure 5: CodeNamer Mobile Interface

Apple's development environment for iOS. The user interface allows players to upload pictures of their Codenames board, view and edit the board, withhold colors in "PlayerView," and generate clues. Issues with iOS15 required us to create and patch a development pod in order to allow image cropping with the react-native-image-crop-picker npm package. Source code for the frontend can be found at: https://github.com/ethansaxenian/CodeNamer.

Backend. Our custom API was developed using Flask, a web development framework for Python. It includes routes for our image processing algorithms (using the OpenCV computer vision library and Tesseract OCR library) and word association algorithm (using the Gensim natural language processing library). Photos are accepted and decoded from bit64 encoding within the request object. Current board state and clue parameters, including the hint color, are contained in the request query string. The API is currently deployed on Heroku. Source code for the backend can be found at: https://github.com/ethansaxenian/CodeNamer-api.

5 RESULTS

The CodeNamer Mobile App allows players to easily access our word vector clue generation for any game of Codenames. From the homescreen (Figure 5a), users may select to upload an image of their game board and color card, or enter the board state manually. The info pop up may also be selected, which outlines the best practices for photographing the game. If either text or color upload is selected, users are given the option to capture the board live using their phone camera, or to select an image from their photo library. Once a photo has been selected or taken, it can be cropped before submission. A loading bar displays the progress of an image as it is being processed, during which time further input is not available. If manual input is preferred, the user can enter the game words by color. In both instances, before generating the virtual board, each word is verified to be a valid Codenames card. If image upload does not yield enough words or colors (i.e 25), an error message is displayed to indicate as such. Otherwise, the upload button turns green to verify success.

When both the word board and color card have been successfully processed, or manual input is completed, the app displays our virtual board in "PlayerView," which withholds the color mapping only visible to spymasters (Figure 5b). Upon switching to "SpyView," the color mapping becomes visible (Figure 5c). If either the word or color of a given card has been processed incorrectly, the user may edit either value by holding a long press on that card. Clue generation can be requested for either color using the Red/Blue hint buttons. When this occurs, an accordion is displayed with drop down options to view the 5 highest ranked clues for 2, 3 and 4 cards. After a single request has been made for a given color, the resulting clue generation for that board state is saved to reduce wait time. In figure 5c, we see for instance the clue "Leader" has been generated to relate the blue words "millionaire," "genius," and "head." To update the board state, the player may simply select a card that has been removed from game play, upon which the corresponding Codenames cover image is displayed in its place. Clue generation will then reset, and further requests will reflect the new game state (Figure 5d). In Figure 5d, with various cards covered, a red clue generation yields "Missionaries" for the words "church," "queen," and "match." Noticeably, it has also suggested "Fr" for the words

"church," "pin," and "match." Although at first meaningless, "Fr" is a written abbreviation for Father when it is used in titles before the name of a Catholic priest. Such nuanced hints are often generated in later (and importantly, smaller) board states.

If the player wants to clear the board, they may do so by pressing the reset button. Otherwise, once the game is finished, selecting the "new game" button will redirect the user to the home interface, where they may begin and upload a new game. Overall, the CodeNamer UI allows for users to do much more than easily upload a game and generate effective clues for any variation of its state. Features such as spy/player view, board updating, and card editing conveniently allows users to keep track of, update, or even change a game of Codenames for any player. Users can upload saved images, or simply manually input a list of Codenames words, to play a game on the go with a friend, omitting the need for a physical board. The app acts as not only a tool for the spymaster, but any Codenames fan wanting to optimize their game play.

Our app is currently available for ad hoc distribution.

6 DISCUSSION

The CodeNamer app can successfully read in a board and its color card in order to generate considerably effective clues. Our implementation of these technologies as an applicable mobile UI makes CodeNamer not only an impressive tool for finding hints, but a comprehensive technology for playing Codenames in every sense of the game.

The text and color recognition algorithms are consistent under consistent lighting. The main impediment to both is harsh disparities in lighting that make it difficult to identify contours within the image. This is especially true for shadowy images, which often skew the thresholding process for text cards and make color identification using distance score inaccurate. Adding the manual board feature was a way of making the app and its interior technology more available (and discrete) for every setting.

Heroku proved to be our largest performance challenge. Both our image recognition algorithms and clue generation are significantly slowed when limited to a single dyno. Early on, this resulted in response timeouts for both API endpoints. In order to improve performance for text recognition, our algorithm uses multi-threading to run multiple instances of Tesseract in parallel and process text regions simultaneously, leading to significant increases in speed. We are quite pleased with the ability of these algorithms to consistently and quickly perform in well-lit settings, even with occasional glare or disorganized cards. Once text regions are identified, our text recognition and contour sorting are extremely accurate. Modifications to the clue generation algorithm, such as limiting the size of our word vector database and using memorization to handle invalid clues helped optimize our clue generation time considerably as well.

The quality of clues generated by our algorithm fluctuates. At times it misses obvious connections between words, while at other times it appears to capture nuanced semantic similarities. For instance, one of our test generations yielded “gq” as a hint for the words “cover” and “model.” Although seemingly meaningless, “GQ” is in fact a fashion magazine, clarifying the hint immensely despite its initial obscurity. Overall, the generation provides for insightful connections among words, which makes it a useful tool for a spymaster who struggles to think of clues on their own.

In the future, we hope to continue making improvements to both our image processing and glue generation. Ideally, our text recognition algorithm can be tweaked to handle harsher lighting conditions with better shadow/glare processing. We also hope to expand our clue generation to apply weighting to the various card affiliations on the Codenames board. This may improve the algorithms ability to prioritize disassociating from opposing words over neutral words, or the assassin word above all else. Strategic consideration of the inner workings of Codenames gameplay may also improve clue effectiveness. This could include techniques meant to leave words on the board that will interfere with the opponent’s ability to give effective hints, or “zero” clues that signify a word none of the team’s cards will relate to. Likewise, it is our goal to incorporate new features into the CodeNamer mobile app, such as a field operative bot capable of "guessing" the most similar words on the board for a given clue.

7 ACKNOWLEDGMENTS

Many thanks to Professor Scharstein and the rest of the Fall 2021 CS 701 Senior Seminar for their support of this project!

REFERENCES

[1] C. Jaramillo, M. Charity, R. Canaan, and J. Togelius. Word autobots: Using transformers for word association in the game codenames. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 16(1):231–237, Oct. 2020.

[2] A. Kim, M. Ruzmaykin, A. Truong, and A. Summerville. Cooperation and codenames: Understanding natural language processing via codenames. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 15(1):160–166, Oct. 2019.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv.org*, Sep 2013.