Competitive Language Games as Creative Tasks with Well-Defined Goals

Brad Spendlove and Dan Ventura
Computer Science Department
Brigham Young University
Provo, UT 84602 USA
brad.spendlove@byu.edu, ventura@cs.byu.edu

Abstract

Creative computer systems grapple with challenging tasks that exist within effectively endless combinatorial spaces. Further complicating these already difficult tasks is the fact that the goal of high-quality creative output is itself nebulous. A creative domain with concrete goals would therefore be a fruitful domain for studying computational creativity. We propose that competitive language games are just such a domain—they require creativity but also feature concrete win and loss states. We present an analysis of creative agents that play one such game: Codenames, a 2016 board game of communicating hidden information via single-word clues. Our model-agnostic framework allows us to compare agents that utilize different language models. We present our findings and discuss how future computational creativity research can continue to explore competitive language games.

Introduction

AI agents pursue a goal within an environment. Creative computational (CC) systems are AI agents that seek to generate or identify high-quality creative artifacts within the environment of an effectively endless combinatorial space. A plethora of potential output artifacts exists within that space, each with varying levels of quality. CC systems, therefore, often contend with the unique challenge of seeking a goal that is not well defined.

The space of all possible artifacts for any given human creative domain is so large that defining a goal for a creative agent can be as difficult as building the agent that pursues that goal. Because human creativity is extremely complex, and its mechanisms are only partially understood, CC systems' goals must necessarily be abstractions. The degree to which those abstractions represent the goals of real-world creativity corresponds to the maximum creative potential of systems that use them.

Thus, seeking or developing better defined creative goals is a fruitful avenue for computational creativity research. Enter board games. Concomitant with the boom in modern board gaming (Jolin 2016) is the rise of new social, language-based games in which participants use their creativity to come up with clues, guesses, and deceptions.

Classic guessing games such as Guess Who and newer games like Mysterium (Nevskiy and Sidorenko 2015) require players to reason and make verbal guesses about images. Hidden role games like Werewolf and Spyfall (Ushan 2017) are freer form and involve players talking with one another to deduce others' hidden roles while keeping their own a secret. These games all involve reasoning, creativity, and language skills but critically also include clear objectives and win/lose states. Playing these games is a creative task with the well-defined goal of winning the game. We propose that they are therefore ideal candidates for computational creativity research.

In this paper, we present and analyze a creative system that plays Codenames (Chvátil 2015), winner of the prestigious Spiel des Jahres in 2016.[1] Codenames is a word-based guessing game in which two teams play on a shared grid of 25 word cards drawn randomly from a large deck. One player from each team serves as a "spymaster" who must give their teammates one-word clues corresponding to certain words on the board that are assigned to each team, secret to all except the spymasters. Clues are phrased as a single word and a number, indicating how many cards the clue is intended to relate to.

The teammates then discuss the clue and select word cards on the grid to guess one at a time until they either guess incorrectly or pass. A correct guess identifies one of the team's assigned words. An incorrect guess accidentally identifies one of the opposing team's words, a neutral word belonging to neither team, or an "assassin" word that results in instant game loss. Teams take turns giving clues and guessing until one team wins by identifying all of their assigned cards (perhaps with inadvertent assistance from their opponents) or the opposing team guesses the assassin.

Figure 1 shows an example of a Codenames board of 25 word cards. Previously guessed words are covered with colored tiles corresponding to their hidden roles: blue and red for the opposing teams, grey for neutral, and black (out of frame) for the assassin.

The spymaster's role is to come up with one-word clues that elegantly identify multiple correct words while excluding incorrect words. Importantly, the spymaster's clues are not restricted in any way other than by simple rules about not using words on the cards or acronyms, etc. This task requires knowledge of what each word means and how they

[1] https://www.spiel-des-jahres.de/spiele/codenames/

Figure 1: An example Codenames board, showing covered and uncovered word cards.
Credit: Skip McIlvaine, boardgamegeek.com, CC0 license.

relate to one another. The role of spymaster is easy to attribute creativity to—clues must often navigate tricky positive and negative relationships, and human players can recognize particularly clever or helpful clues.

The spymaster's teammates who guess based on the clues have a less open-ended task, as it is restricted to linking the clue word to one word card at a time. This role requires relatively less creativity, but it is still a non-trivial language task for humans and computer systems.

In order to complete either of these tasks, an agent must have an understanding of how words both positively and negatively associate with one another. Using that knowledge, the spymaster searches for a clue word that their guessers will most likely associate with a chosen subset of the team's word cards while avoiding associations with incorrect word cards. The guesser uses a similar language faculty to guess word cards that most closely relate to the clue.

In this paper, we present a framework for a system that completes those tasks to play Codenames in both the spymaster and guesser roles. We explore how different models perform at these tasks in competition against one another, with the goal of demonstrating how competitive language games can serve as useful test beds for creative systems.

Creativity in Codenames

The spymaster's task involves elements of puzzle and problem solving, and we may view the potential for creativity in solving the task through that lens. The skills required of a good spymaster player are related to problem framing (Guilford 1956; Dorst 2011) and re-representation (Ohlsson 1992; Veale 2006), both of which are well-known to facilitate creativity.

It is important to realize that even though the output produced by the spymaster is a single clue word (and an associated number), the artifact which the spymaster is creating is not a word. Rather, it is something like a multi-word relationship graph (including both positive and negative connections). The spymaster uses skills such as those mentioned above, as well as, of course, their knowledge of language to create this graph structure, which, in a more traditional creative setting, would constitute the output artifact. To make this a game, the artifact is instead obfuscated, with only the clue word and number giving hints about its structure. The guessing players' job is, essentially, to re-create this relationship graph from the clues and use it to identify words assigned to their team.

Serendipitously, it is this gamification of creativity that affords us a well-defined, if indirect, measure of creativity in the form of game outcomes. While in many traditional creative settings, artifact value is often very difficult to measure, the appeal of Codenames—and other competitive language games—is that the value metric is (at least) strongly correlated with the win/loss outcome. This serves as a powerful proxy for evaluating the creativity of the system itself, or at least one critical element of it.

Creative domains are characterized by their extremely large combinatorial spaces, which are a prerequisite for novelty. The word relationship graphs that are key to playing Codenames are simpler than other types of artifacts such as literature or visual art, but they are nevertheless complex enough to be considered in the same class as metaphors or short jokes and witticisms—domains which make a strong case that creativity can be manifest where artifacts take the form of a small number of words connected in a clever or surprising way.

Related Work

We use language models to provide the word association faculties that our Codenames player agents need. In particular, we use word2vec and GPT-2. We make use of word2vec's word embeddings and both GPT-2's word embeddings and text generation capabilities.

Word embeddings are a way to represent words as vectors such that vectors that are close to each other occur in similar contexts in text. The distributional hypothesis posits that words that occur in similar contexts have similar semantic meanings (Harris 1954). This hypothesis is the basis for distributional semantics (Sahlgren 2008), a theory that forms the basis for word embedding models. A model built on this theory is a natural fit for use in playing Codenames because words that appear in similar contexts are likely to be associated with one another in a way that will help players guess related words.

A word embedding model is trained on a corpus of text to encode the relative contexts of the words in the corpus into a vector space. Word2vec (Mikolov et al. 2013a) is a neural network model that learns word associations in this manner. The word2vec model we use implements a skip-gram and negative sampling unsupervised learning model, which learns to predict the context for a given word in the corpus. It is trained to build an embedding that minimizes the distance between a word and its context while maximizing the distance between a word and a hallucinated (random) context. The weights that are trained with this method are treated as the vector space into which words are embedded.

GPT-2 (Radford et al. 2019) is a powerful, large language model (LLM) that implements a transformer architecture (Vaswani et al. 2017). This unsupervised training method uses attention mechanisms to focus learning on

small but important parts of the training corpus. GPT-2 has been demonstrated to perform well at a variety of tasks such as summarization, translation, question answering, and text generation. It is notable, however, that the model was not trained on any of those tasks explicitly. Its attention-based language model learns implicitly to complete such tasks via training to predict text from a prompt.

GPT-2 can be used (with varying degrees of success) as a general-purpose model by providing it a text prompt that describes a task to be completed. The model generates output that it predicts to be a likely continuation of the prompt. This output is highly dependent on the prompt, giving rise to a new and still-developing discipline known as prompt engineering (Liu et al. 2021). A common form for LLM prompts is listing a handful of complete examples of the task to be solved and then providing an incomplete task for the model to complete.

Prompt engineering is now an integral part of natural language processing using LLMs. Recent projects like PromptSource (Bach et al. 2022) facilitate the sharing of prompts for various tasks, allowing the research community to build upon past successes and find more useful prompts.

We built our word2vec agents with the Gensim implementation (Řehůřek and Sojka 2010), using "pre-trained vectors trained on part of the Google News dataset (about 100 billion words)" (Mikolov et al. 2013b). Our GPT-2 agents used HuggingFace GPT-2 Small (124M parameters) (Wolf et al. 2020).

Methodology

To experiment with using Codenames as a test bed for creative language systems, we built an agent-agnostic game playing framework and implemented language model-agnostic AI player agents for both the spymaster and guesser roles. As this work is focused on language-based creativity, we use the same rudimentary decision-making process for all the agents we experimented with. There are undoubtedly many possible improvements to their strategies, but they are reasonable for the purposes of this research. By keeping the agents' strategies static, we are better able to isolate the effect of using different language models.

All of the code described in this section can be found in a public GitHub repository. [2]

Defining the Codenames Task

The Codenames game board is a set G of 25 word cards drawn from a deck of size 400 (for play, the cards are arranged in a 5x5 grid). G is partitioned into four subsets: T, an unknown set of target words, P, an unknown set of opponent words, N, an unknown set of neutral words and $A = \{a\}$, an unknown singleton set that contains an assassin word. $G = T \cup N \cup P \cup A$ represents an instance of the game with $|G| = 25, |T| = 9, |P| = 8, |N| = 7, |A| = 1$.[3]

[2]https://github.com/gbspend/codenames

[3]This admits $\binom{400}{25} = 3.374984143967 \times 10^{39}$ unique draws, each of which admit $\binom{25}{9}\binom{16}{8}\binom{8}{7} = 2042975 \times 12870 \times 8 = 210,344,706,000$ possible games, for a total of $7.099100475174 \times 10^{50}$ unique games.

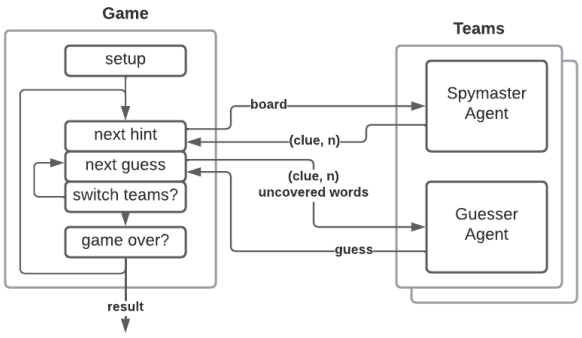

Figure 2: The high-level information flow of our Codenames framework.

While the set of cards G is public knowledge, its partitioning is not; this information is initially only available to the two teams' spymasters, who have access to a secret key.

Let U be the set of cards whose partition membership is currently unknown. Initially, $U = G$. The object of the game is for the spymaster to help their teammates discover which words are in T before the opponent discovers which words are in P and without discovering the identity of word a. Teams alternate playing in rounds. Whichever team plays first will have nine words to guess, while the other will have eight (note that for convenience and without loss of generality, we assume $|T| = 9$).

Play proceeds in the following manner. The active team's spymaster generates a clue $c = (w, k)$ consisting of a clue word $w \in W \setminus G$ and a number $0 < k \leq |T|$, where W is the set of all English words[4] and $k = |I|$ where $I \subseteq T$ is a secret set of words to which the spymaster intends the clue to correspond. I itself changes every round, depending on the spymaster's strategy, and is not recorded in the game.

Given a clue word w, the guesser may then make a maximum of $k + 1$ guesses. The guesser's task is to guess a word $v \in U$ whose partition membership is then revealed (by covering it with one of four tile types), removing it from U (by removing it from its secret partition). If $v \in T$, the team guessed correctly and may pass or guess again as long as they have not exceeded $k + 1$ guesses for the current round. The round is over if the guesser has used all of their guesses, if they pass, if $T = \emptyset$, or if $v \notin T$. If $v = a$ or $P = \emptyset$ (meaning they guessed the assassin word or their opponents' final word), the team loses the game. If $T = \emptyset$, they win the game. Otherwise, play passes to the other team. This process of playing rounds repeats until one team wins.

Codenames Framework

We built a framework for playing teams of Codenames agents against one another. Figure 2 shows a diagram of our framework's architecture. The Game module randomly determines which team will play first, sets up the board and secret key, and begins the gameplay loop. In

[4]We assume W excludes acronyms and proper nouns.

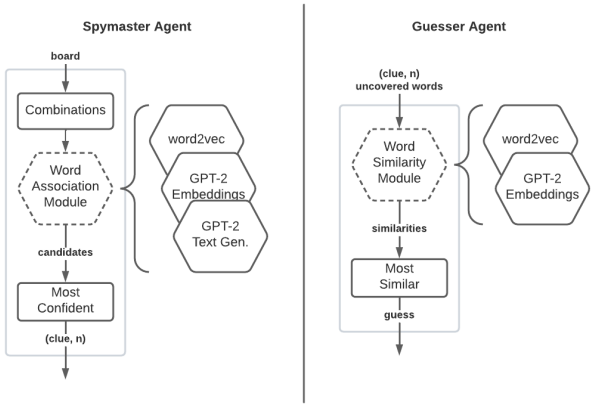

Figure 3: Information flow in our two types of Codenames agents. Note that the language task modules are pluggable.

each round, the Game module passes the current board state $b = (T, P, N, A)$ to the active team's spymaster. The spymaster agent returns a clue $c = (w, k)$. That is, the spymaster agent implements a function $\sigma : \mathcal{B} \rightarrow \mathcal{C}$, where $\mathcal{B} = 2^T \times 2^P \times 2^N \times 2^A$ is the set of possible game board states[5] and $\mathcal{C} = W \times \mathbb{N}_{|T|}$ the set of possible clues.

The Game module then passes c and U to the active team's guesser agent, which returns its guess v. That is, the guesser agent implements a function $\gamma : \mathcal{C} \rightarrow U$. Depending on the conditions described above, the Game module determines whether the active team may guess again, if the round has ended, or if the game is over. At the end of the game, the Game module outputs a result that indicates which team won and includes the board, the key, and the history of player actions for convenience in later analysis.

Spymaster AI

The spymaster's task is to choose an intended set $I \subseteq T$ and a clue word w. To play effectively, it must balance maximizing both $|I|$ and its estimated likelihood that w will induce a guess v such that $v \in I$. Playing very safely by giving clues that correspond to exactly one word card will likely lead to a low number of incorrect guesses but may progress too slowly to beat the opposing team. Conversely, choosing a clue that tries to represent too many of the team's word cards at once will likely lead to a vague clue that will confuse or mislead the spymaster's teammates.

Recall that guessing incorrectly ends that team's turn at best and at worst reveals one of the other team's target words for them or loses the game immediately by guessing the assassin. Thus, it is also helpful to consider the set $X = N \cup P \cup A$ of words with which the clue word w should *not* be associated.

Figure 3 gives a diagram of our spymaster AI agent that implements the function σ. It generates combinations of words to consider for I and passes each combination, along with X, into its word association module. That module (described below) returns a list of candidate clue words along with a confidence score for each. The agent selects the candidate with the best confidence as its clue word w. Our spymaster's procedure for choosing I relies on the heuristic that a clue representing between two and four words is a reasonable balance of reserved and aggressive play.[6]

In the case that only one of the team's word cards remains uncovered, it is trivial to select $I = T$. In all other cases, the spymaster AI will use the set $\mathcal{J} = \{J \in 2^T \mid 2 \leq |J| \leq 4\}$ to query a *word association module* (WAM), which computes a function $\alpha(\mathcal{J}, X) = (w, k) = c$. The spymaster then passes the returned clue c to the game module.

Word Association Module (WAM)

To implement α, the WAM uses the sets J, X to construct an abstract parameterized family of scoring functions $\sigma_{J,X} : W \rightarrow \mathbb{R}$ that maps a word $u \in W$[7] to a confidence score reflecting how well u is positively associated with the words in J *and* negatively associated with the words in X.

Given σ, we compute α as follows. For each $J \in \mathcal{J}$:

1. rank order W by the score $\sigma_{J,X}(u), \forall u \in W$
2. put the top m words into a candidate set C_J
3. compute $\mu_J = \frac{1}{|C|} \sum_{u \in C} \sigma_{J,X}(u)$

Next, find the set with the highest average confidence score, $J^* = \text{argmax}_{J \in \mathcal{J}} \mu_J$ and, from its associated candidate words, C_{J^*}, find the word with the highest score, $w^* = \text{argmax}_{u \in C_{\mathcal{J}^*}} \sigma_{J,X}(u)$. Finally, return the tuple $(w, |J^*|)$.

By choosing the combination J^* with the highest *average* confidence, the model favors combinations that are more closely related altogether, even though another combination may have a single candidate word with higher confidence.

The primary language faculty required to play Codenames is knowledge of the relationships between words, both positive and negative. Knowing which words positively relate to each other is a necessary baseline skill, and understanding negative relationships between words is important for an agent to perform well.

For example, two Codenames word cards are "ambulance" and "doctor". These words are closely related, but if "ambulance" was one of a team's word cards (that is, "ambulance" $\in T$) and "doctor" was the other team's [or the assassin] ("doctor" $\in X$ or "doctor" = a), it would be important to exclude clue candidates that positively associate with "doctor". In that scenario, "siren" or "fast" would likely be a better clue than "emergency" or "injury".

Thus in this example, when $J \subseteq T$ contains "ambulance", we desire $\sigma_{J,X}(\text{“}siren\text{”}, k) > \sigma_{J,X}(\text{“}injury\text{”}, k)$.

[5] The notation 2^S is shorthand for the power set of S.

[6] As stated in the Codenames rulebook: "Getting four words with one clue is a big accomplishment."

[7] While this likely is technically incorrect, in the sense that any language model is likely subject to some out-of-vocabulary words, the language models used here support large enough vocabularies that they render the point basically moot—word2vec has a vocabulary size of 500K words, and GPT-2 uses a vocabulary of 50K sub-word tokens that likely translates to a functional word-level vocabulary even larger than that of word2vec.

The WAMs we experiment with are intended to serve as good players regardless of whether their teammates are human or other black-box AI players. Thus, a relevant consideration is whether the clues generated by the WAM are understandable to a broad audience. An obscure clue word and/or uncommon relationships to the word cards would likely confuse or mislead the spymaster's teammates. For this reason, statistical language models are a good fit for this task. We leverage them for both types of WAM.

We reiterate that the spymaster AI is designed to use *any* word association module that computes α, independent of how the WAM models language or how it uses the model to generate clues—the WAM can implement the function family σ in any way that maps words to scores. We implement three different versions: two that use a scoring function based on cosine similarity between word embeddings and one that uses conditional probabilities from autoregressive text generation by a language model.

Word Embedding WAMs

Both word2vec and GPT-2 feature word embeddings which are well suited to word relationship tasks because they allow semantic word comparisons using simple geometric, vector-based operations.

We built two word association modules that use word2vec and GPT-2 embeddings, respectively, to compute σ. Words are converted to real-valued embedding vectors using a language-model-specific embedding function $\upsilon : W \to \mathbb{R}^d$. For a word u, positive word set J and negative word set X, $\sigma_{J,X}$ is then computed using cosine similarity between the word vector $\upsilon(u)$ and a mean set vector $\mu_{J,X}$:

$$\sigma_{J,X}(u) = \frac{\mu_{J,X} \cdot \upsilon(u)}{||\mu_{J,X}||\,||\upsilon(u)||}$$

where

$$\mu_{J,X} = \frac{\sum_{v \in \upsilon(J)} v - \sum_{v \in \upsilon(X)} v}{|J| + |X|}$$

where we slightly abuse notation by overloading υ to embed a set of words into a set of embedding vectors. Note that this embedding function is the only language-model-specific component of this approach—word2vec and GPT-2 learn their embedding spaces in different ways.

Text Generation WAM

Text generation is a powerful function of the GPT-2 language model. For our third implementation, we built a word association module that uses text generation to construct $\sigma_{J,X}$. To generate text, GPT-2 takes a prompt and generates tokens that are, according to its model, likely to follow. We designed prompts that state that a list of words related to an input word will follow, ending with a colon, followed by a comma-separated list of such words. The last line of the prompt ends after the colon, prompting GPT-2 to complete what comes next with an appropriate list. Here is an example of such a prompt; note that all three lines comprise a single prompt:

This is a list of words related to ambulance: paramedic, emergency, doctor.
This is a list of words related to boat: water, fish, captain.
This is a list of words related to school:

We experimented with three such prompt templates for use in the spymaster AIs. Each prompt asks GPT-2 to list words related to an input. The first prompt asks for words related to a single (positively associated) word. The second asks for words that are positively associated with two words. The third asks for words that are positively associated with one word and negatively associated with another. Because J and X can contain more than one or two words, we iterate over all possible template completions using words from J and X to construct the set C of possible generated completions. Let $\pi_i(Y)$ be the prompt created by adding the tuple of words Y to template type i and let $Z_i(Y)$ be the set of words generated by GPT-2 when prompted with $\pi_i(Y)$. Then, for the first template type (one positive association),

$$C = \bigcup_{u \in J} Z_1(u)$$

for the second template type (two positive associations),

$$C = \bigcup_{(u,v) \in J \times J} Z_2((u,v))$$

and for the third template type (one positive and one negative association),

$$C = \bigcup_{(u,v) \in J \times X} Z_3((u,v))$$

In the case that $|J| = 1$ or $X = \emptyset$, the module defaults to the single positive prompt template.

Each of these prompts is templated to allow for arbitrary input words, and the generated text is post-processed to extract only words in a comma-separated list. Any other output is discarded. The list is then filtered so that only valid, nonduplicate Codenames clues remain (e.g. single words that do not contain any of the word cards in U). The set of words that remains is included in the candidate set C.

For a word u, positive word set J and negative word set X, $\sigma_{J,X}$ is then computed using the generative model's conditional probability for u:

$\sigma_{J,X} = p(u|\pi_i(Y))$ for Y a valid tuple from J, X for type i

Table 1 shows examples of each prompt template. The first and second columns list the template inputs: a single positive association, two positive associations, and one positive and one negative association. The third column shows the complete prompt with GPT-2's generated completion text. The input words are shown in italics, and the generated text in bold; all other text is the template. Note that any newlines are explicitly contained in the template or generated text. The final column shows the result of post-processing the generated text to extract valid clue candidates.

The GPT-2 model is not fine-tuned; its output relies solely on the prompt. The open-ended nature of text generation means that it is susceptible to noise in the output. We found that using a small number of template inputs reduced that

Positive	Negative	Templated Prompt + Generated Text	Candidates
cook		This is a list of words related to ambulance: paramedic, emergency, doctor. This is a list of words related to boat: water, fish, captain. This is a list of words related to *cook*: **urn, fire, vessel.**	urn, fire, vessel
hospital, spell		This is a list of words related to flag and state: country, government, county. This is a list of words related to mammoth and pyramid: ancient, large, heavy. This is a list of words related to bridge and skyscraper: concrete, blueprint, tall. This is a list of words related to *hospital* and *spell*: **crisis, catastrophe, crisis, disaster.**	crisis, catastrophe, disaster
lock	carrot	This is a list of words that are related to ambulance but not doctor: siren, engine, fast. This is a list of words that are related to bat but not duck: cave, night, fur. This is a list of words that are related to queen but not king: regina, woman, wife. This is a list of words that are related to *lock* but not *carrot*: **urn, house, castle, castle.** **This list is the closest of the**	urn, house, castle

Table 1: Examples of the prompt templates used in our three text generation word association modules. The positive and negative inputs are inserted into the templates which GPT-2 then uses to generate the bolded text. That text is post-processed to extract a list of valid candidate clues.

noise, which is reflected in our three templates. As such, both of the positive-only templates disregard X. We experimented with how well each template performed while making these trade-offs.

We also experimented with different wordings for the prompt templates, for example beginning each line with "These words are related to..." instead of "This is a list of words related to..." We discuss why we chose the wording and number of inputs for the final prompt templates in a later section.

Guesser AI

As discussed previously, the task of the spymaster's teammates is to guess which word cards the clue is intended to represent. Therefore, the guesser agent is simpler, and the module requires only a word embedding model to calculate it. Figure 3 includes a diagram of our guesser AI. The agent uses its word similarity module to choose the word $u^* \in U$ that is most related to the clue $c = (w, k)$, again using cosine similarity:

$$u^* = \underset{u \in U}{\operatorname{argmax}} \frac{v(u) \cdot v(w)}{||v(u)||\,||v(w)||}$$

We note that this guessing process disregards the number k provided in the clue. While that is additional information that the guesser could leverage, we believe that this simplified approach is sufficient for the research task at hand, namely the exploration of different language models as creative Codenames players. From this perspective, the spymaster is the more interesting agent and was therefore the focus of our experiments. Furthermore, whatever information the clue number provides is supplementary to the associations between the clue word and the word cards. At most, it could be used to refine the language module's association scores.

GPT-2's text generation function is open-ended; it can generate any tokens that appeared in its training corpora. Therefore, the likelihood of the model generating the specific words found on the board is very low. We experimented with building prompts for the guessing task. For example (again, all four lines comprise one prompt):

> Which of the words ambulance, shoe, and Moscow is most closely related to siren? ambulance
>
> Which of the words chick, China, and bolt is most closely related to lightning? bolt
>
> Which of the words opera, casino, pilot is most closely related to fancy? opera
>
> Which of the words India, needle, shop is most closely related to sharp?

In this example, the intended result is that GPT-2 generates "needle" as the next word. We tested whether the intended word appeared at all before the first generated newline character. Our experiments showed that GPT-2 generated the intended word in less than 5% of trials. We therefore did not build a GPT-2 text generation guesser at this time.

Model Comparison

As discussed above, the word association tasks that are required to play Codenames, especially in the spymaster role, provide opportunities for creativity. Each of our player agents includes a pluggable, language model-driven module that serves as the creative heart of its playing procedure. By comparing these modules within the well-defined creative space of a competitive language game, we can concretely reason about their performance.

To make these comparisons, we built a lightweight test harness that plays games of Codenames between two teams of agents. These games are played using the same list of word cards available in the retail game. Each team consists of a spymaster agent and a guesser agent who are agnostic to the implementation of the agents they are playing with and against. Codenames can be played with a small team of guessers collaborating to guess their spymaster's clues, but teamwork between guesser agents is outside the scope of this work. The fundamental task of testing language models in this game setting can be adequately explored with a solo guesser.

To provide benchmarks for the agents' performance, we implemented simple guesser agents that guess randomly or cheat. The random guesser agent serves as a lower bound on acceptable performance for an AI agent. The cheat guesser agent simply guesses n correct words each round, then passes. This serves as a rough but easy-to-compute pace against which to compare each agent.[8] A benchmark team consists of either a random or cheat guesser agent paired with a trivial spymaster agent that returns a dummy clue that the guesser disregards.

We created teams out of every pairing of spymaster and guesser AI agents, regardless of their underlying language models. These teams were played against one another and the benchmark agents, and their win/loss ratios were recorded. The name of each team is given as "[spymaster]4[guesser]", meaning the spymaster is making clues fo(u)r the guesser. "w2v" stands for word2vec, "gpte" stands for GPT-2 embedding model, and "gptp" stands for GPT-2 prompt (text generation) model. The six teams were w2v4w2v, w2v4gpte, gpte4w2v, gpte4gpte, gptp4w2v, and gptp4gpte. Each team played 30 games against every other team and 30 games against a random team, a cheat team with $n = 1$, and a cheat team with $n = 2$.

Results

In this section, we report the results of our experiments. Our primary objective is to demonstrate that a competitive language game task allows for quantifiable comparison between agents. By powering our player agents with language association modules, we show by extension how such modules can be evaluated with concrete performance metrics.

We ran experiments using three spymaster agents and two guesser agents, in addition to the cheat and random benchmark agents. The two guesser agents were built on word2vec and GPT-2 word embeddings. The spymasters used word2vec word embeddings, GPT-2 word embeddings, and GPT-2 text generation, respectively, to perform word association tasks. The text generation spymaster could further be configured to use one of the three prompt templates shown in Table 1.

GPT-2 Prompt Comparison

We compared the performance of the three prompt templates by playing text generation agents (paired with both guessers) against cheat benchmarks with $n = 1$ and $n = 2$ as well as the random player. Table 2 shows the results of playing 10 games between those teams. Each template performed similarly against the random player, with most teams being able to beat it consistently. Similarly, all teams performed uniformly poorly against the cheat benchmarks. In the tests that follow, we used the template with one positive input as the prompt for the GPT-2 text generation module.

Comparing Agent Teams

By playing our various teams of agents against one another, we can judge their relative performance at the word association task. The benchmark cheat and random agents provide a more objective performance measure. Figure 4 shows the win/loss ratio for each combination of agents playing 30 games against every other team and the benchmark agents. A darker color indicates a higher win rate for the team on the y-axis versus the team on the x-axis. Recall that we did not implement a GPT-2 text generation guesser. We also did not play a team of agents against a team of the same agents.

Looking at the results, we can see that the word2vec spymaster and guesser team performed best overall. Conversely, the team of the GPT-2 text generation ("gptp") spymaster and GPT-2 embedding guesser was the weakest. Most teams beat the random agent at least half of the time, with gpte4gpte and gptp4gpte teams losing as often as they won against it. None of the teams could consistently win against the cheat agents, but the word2vec spymaster/guesser team was able to beat the cheat with $n = 1$ about a third of the time.

Finally, we note that these tests are automated and can be carried out on any new or modified agent to test its performance at the creative spymaster task under the same circumstances. This will allow for easy evaluation and comparison

Prompt	Guess	Cheat 1	Cheat 2	Rand
1 positive	w2v	0-10	0-10	6-4
	GPT-2	1-9	0-10	7-3
2 positive	w2v	0-10	0-10	7-3
	GPT-2	0-10	0-10	5-5
pos + neg	w2v	0-10	0-10	9-1
	GPT-2	0-10	0-10	4-6

Table 2: Win-loss ratio results of playing different text generation spymasters against benchmark agents, each using one of the three prompt templates.

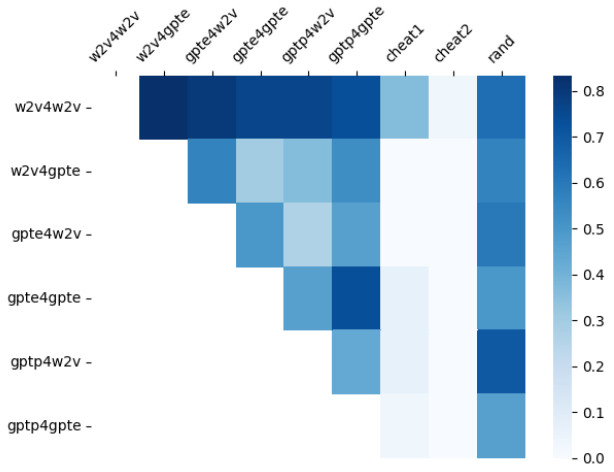

Figure 4: Heatmap of win/loss ratios after 30 games for each team of Codenames agents playing against one another and the unintelligent benchmark agents.

[8] Anecdotally, it seems a human team would find a cheat agent with $n = 3$ challenging and struggle to ever beat one with $n = 4$.

as improved models are developed in the future.

Discussion

Our experiments with Codenames AI agents serve a dual purpose: to present an initial attempt at designing agents to play the game; and (more importantly) to demonstrate that a competitive language game is a creative domain with a unique capability for evaluating agents.

Our Codenames Agents

It is somewhat surprising at first blush that word2vec outperformed GPT-2 word embeddings at playing Codenames. This may be attributable to differences in how the two models are trained. Word2vec's skip-gram and negative sampling model is trained under circumstances that are very similar to the task of finding words associated with an arbitrary set of positive and negative concepts. GPT-2, while not unsuited for the task at hand, is trained to more generally minimize cross entropy in its language model. Perhaps training or fine-tuning a transformer module using skip-grams and negative sampling would bring their power to bear on this more specific task.

This surprising result was demonstrated very clearly by the test methodology of playing a competitive game with the two models. This serves as another example of how this creative task is useful to CC research. Additionally, improved future spymaster agents can be tested against these same models to evaluate their performance.

Designing a word association module using GPT-2 text generation relied heavily on prompt engineering. Prompting the model with examples in the form of a comma-separated list resulted in the generated text taking a similar form. This allowed for consistent input to the post-processor to extract clue candidates.

More challenging was engineering a prompt template that harnessed the power of the language model to generate high-quality word associations. As described previously, we settled on three prompt templates that sought associations with one word, two words, and one positive and one negative word. By contrast, the word embedding models calculated word associations using an arbitrary number of positive and negative word embedding vectors.

We found that increasing the number of input words in the template tended to increase the noise of the generated text without improving the quality of its associations. However, the results reported in the previous section show that the GPT-2 text generation module prompted with one positive input performed about as well as the GPT-2 word embedding model.

The Future of Codenames as a Creative Task

Successfully playing Codenames requires robust knowledge of relationships between words, but the input and output for player agents are single words or lists of words. This stands in contrast with a game like Werewolf which requires more complete communication skills as players attempt to figure out hidden roles. There is a smaller conceptual distance between the language model and the agent's performance playing Codenames.

Playing a competitive game allows for automated and easy-to-compare metrics for modules with open-ended tasks, such as using GPT-2 text generation to compute word associations. Further, by first building a Codenames test harness, we were able to quickly test and compare prompts. For example, we found that prompts beginning with "This is a list of words related to..." gave better results than those beginning with "These words are related to...".

The nature of competitive language games like Codenames allows for future improved and novel agents to be tested under identical conditions to the ones presented here. We foresee an improving field of creative Codenames agents that can be tested automatically against one another.

Bodily and Ventura present an argument for increased social consciousness of CC systems, especially as they eclipse human performance (2020). This is largely motivated by the triumph of AlphaGo over a top-ranked human player at Go, which is a creative task with a "well-defined and universally-recognized way of comparing" performance.

Codenames does not have the same depth, history, or audience that Go has, but it is quite popular in its own sphere. It shares a similar potential for creativity but operates in the domain of language. Creativity in language domains is a valuable and well-studied aspect of computational creativity, and Codenames could serve as a test bed to develop creative language modules that could be exported for use in those more traditional domains.

These arguments for Codenames as a valid and useful creative domain apply to other competitive language games as well. We encourage the research community to seek out and experiment with such games as well-defined creative tasks.

Conclusion

A common difficulty in systematizing creativity is identifying an accurate and concrete goal. High-quality creative output is difficult to quantify, and abstractions or estimations are usually required. We argue that competitive language games such as Codenames are a useful creative domain because they feature well-defined win and lose states while still allowing for creative expression.

We present a test framework for playing games of Codenames between AI agents both to describe a new creative system and to demonstrate the efficacy of the domain itself. This framework is modular to allow for any player agent to be evaluated and includes benchmark agents to provide more objective performance metrics.

Each creative domain provides unique challenges and new perspectives on what creativity is, how to reason about it, and what tools facilitate computational creativity. Adding competitive language games to CC's suite of canonical creative domains will allow for more rigorous evaluation and comparison of its creative systems. There is more to creativity than winning a game, but in the face of a dearth of concrete measures of creative performance, competitive language games can serve as a valuable proxy for such evaluation.

Author Contributions

Both authors planned and designed the system, B.S. wrote the code and ran experiments, and both authors contributed to the writing.

Acknowledgements

None.

References

Bach, S. H.; Sanh, V.; Yong, Z.-X.; Webson, A.; Raffel, C.; Nayak, N. V.; Sharma, A.; Kim, T.; Bari, M. S.; Fevry, T.; Alyafeai, Z.; Dey, M.; Santilli, A.; Sun, Z.; Ben-David, S.; Xu, C.; Chhablani, G.; Wang, H.; Fries, J. A.; Al-shaibani, M. S.; Sharma, S.; Thakker, U.; Almubarak, K.; Tang, X.; Tang, X.; Jiang, M. T.-J.; and Rush, A. M. 2022. Promptsource: An integrated development environment and repository for natural language prompts. arXiv:2202.01279.

Bodily, P., and Ventura, D. 2020. What happens when a computer joins the group? In *Proceedings of the 11th International Conference on Computational Creativity,*, 41–48.

Chvátil, V. 2015. *Codenames*. Kladno, Czech Republic: Czech Games Edition. Board Game.

Dorst, K. 2011. The core of 'design thinking'and its application. *Design studies* 32(6):521–532.

Guilford, J. P. 1956. The structure of intellect. *Psychological bulletin* 53(4):267.

Harris, Z. S. 1954. Distributional structure. *Word* 10(2-3):146–162.

Jolin, D. 2016. The rise and rise of tabletop gaming. *The Guardian*. Accessed: 2020-10-05.

Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; and Neubig, G. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. arXiv:2107.13586.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *arXiv* abs/1301.3781.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, 3111–3119.

Nevskiy, O., and Sidorenko, O. 2015. *Mysterium*. Paris, France: Libellud. Board Game.

Ohlsson, S. 1992. Information-processing explanations of insight and related phenomena. *Advances in the Psychology of Thinking* 1:1–44.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. `https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

Řehůřek, R., and Sojka, P. 2010. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50. Valletta, Malta: ELRA.

Sahlgren, M. 2008. The distributional hypothesis. *Italian Journal of Disability Studies* 20:33–53.

Ushan, A. 2017. *Spyfall 2*. Moscow, Russia: Hobby World. Board Game.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 31, 6000–6010.

Veale, T. 2006. Re-representation and creative analogy: A lexico-semantic perspective. *New Generation Computing* 24(3):223–240.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Online: Association for Computational Linguistics.