## Mechanics

|  |  |  |
| --- | --- | --- |
| **Word property/relationship** | **Mechanic** | **Example** |
| Language of origin | “ | . |
| Part of speech | ~ | ! |
| Homonym | ! | ; |
| Emphasis (where the stress falls in the word) | ‘ | , |
| Diction (fanciness of speech) | Attack Height   1. Slang == low attack 2. High diction = high attack | 1. “Fam, you are bad” is a low attack 2. “Colleague, you are inferior” is a high attack |
| Length | Damage/Range | 1. Player One says “You have good taste in movies” 2. Player Two says “You have incredible taste in movies” 3. Player Two’s sentence does more damage |
| **Emotional connotation** (determined by algorithm trained by judges??)  iSchool McCool: 5Happ, 3Fear, 10Anger, 1 Sad  Me: | Damage Type: one of:   1. Sadness 2. Happiness 3. Fear 4. Anger | 1. Player One says “You are like a rainbow” 2. Player Two says “You are like a train wreck” 3. Player One’s sentence does Happy damage & Player Two’s sentence does Sad damage |
| Antonym/Opposite | Counter   1. Sadness & Happiness are opposites 2. Fear & Anger are opposites | 1. Player One says “your mom sucks” 2. At the same time or shortly after, Player Two says “my mom rocks, did you know that also ur mom sucks” 3. Player Two has countered Player One’s attack |
| Synonym | Block | 1. Player One says “I like your shoes” 2. Player Two says “My shoes are cool” 3. Player Two has blocked Player One’s attack |

## Milestones

1. Step one (probably the hardest): as a proof-of-concept, write a Python script with a command-line interface that generates batches of words as described in this document. You should be able to construct something that looks like an English sentence, similar to randomly tapping auto suggested words when composing a text.
   1. Generate the first batch of words and wait for the user to select a word from among these in the terminal
   2. Suggest further words based on the first choice
   3. Keep suggesting words until word limit is reached, then print out the sentence and terminate

**Need to learn how to do language modeling / use a language modeling library**. Quick summary of n-gram modeling: <https://stackoverflow.com/questions/18728290/word-prediction-algorithm>

1. Step two: add the ability to compute the emotional value of a constructed sentence.
   1. Hook into some API or library that provides semantic analysis. Maybe we will write our own in the future.
   2. When the user is done constructing their sentence, give it a score out of 4 in each of: Happiness, Sadness, Fear, Anger.
2. Step three: add more rules and dimensions for suggested words
   1. Make distinctions between words that are slang/informal and words that are formal. Give a “formalness” score for a sentence.
      1. Are there APIs or libraries that identify/rank slang on some sort of scale? That would be extremely useful.
   2. Distinguish between words of different lengths. Longer words = longer range or more damage
3. Step four: add a second player, such that the Game Flow described in this document can be carried out on the command line.

## Game flow

1. Round start
2. Each player character gets a semicircle-shaped menu of 3 words near their their head.
   1. Each of these sets of three words is called a batch.
   2. Players’ batches are not independent. That is, players should have the option of playing a word that has a relationship to their opponent’s word
      1. Should have a couple opportunities in a sentence to play an antonym, synonym, slang word, formal word, etc.
   3. When a player selects a word
      1. The word is added to the player’s current sentence
      2. A new batch of words replaces the current batch of words.
         1. This batch consists of words that can logically continue the sentence.
         2. For example, if the first word was “I”, the next batch might be consist of verbs like “went,” “fought,” and “ran.”
         3. This is similar to iOS’s and Androids suggested words when composing a text. The sentences don’t necessarily have to be grammatical, make sense, terminate correctly, etc. All that matters is that it resembles an English sentence.
   4. The player continues selecting words until the timer runs out (3 seconds?) or the maximum number of words is reached (5-7 words).
   5. Players start a new sentence at the same time.
3. Continue exchanging words until end state is reached
4. At the end of the round, show some highlight such as most damaging attack or happiest sentence, etc.
5. Continue playing rounds until there is a winner for the match
   1. Best of 3 -- player who wins 2 rounds wins the match

## Unanswered Questions

1. How do players telegraph their weaknesses/emotions?
   1. Face.
2. Do players only see opponent’s total hit points, or a breakdown of hit points in each emotion?
   1. Probably just the total
3. How do players rank up an emotion?
   1. XP points?

## Controls

* Attack
  + Words chosen from a menu
  + Menu can be cycled through with a directional control e.g. mouse or joystick
* Positioning
  + Moving left/right on the screen
  + Jumping
  + Ducking

## Victory condition

* Fighting games usually have a health/life bar for each player. The match goes on until a player’s life bar reaches zero or the timer runs out. If the timer runs out, the player with the most health wins. Otherwise, the last player standing wins.
* Instead of health bars, our fighters have “Composure,” the sum of their Happiness, Sadness, Fear and Anger points. If the timer runs out, the player with the most Composure wins. Otherwise, the last player standing wins.

## Game modes

* We could have various game modes, each with an associated dictionary, each targeted toward a specific audience. Examples:
  + Shakespearean: each sentence has to be in iambic pentameter
  + Rap Battle: a dictionary composed of the most common words in the lyrics of 1000 popular rap songs
  + ESL (English as a Second Language): a simplified English language dictionary
  + G-Rated: English language dictionary, minus any vulgar words & swears