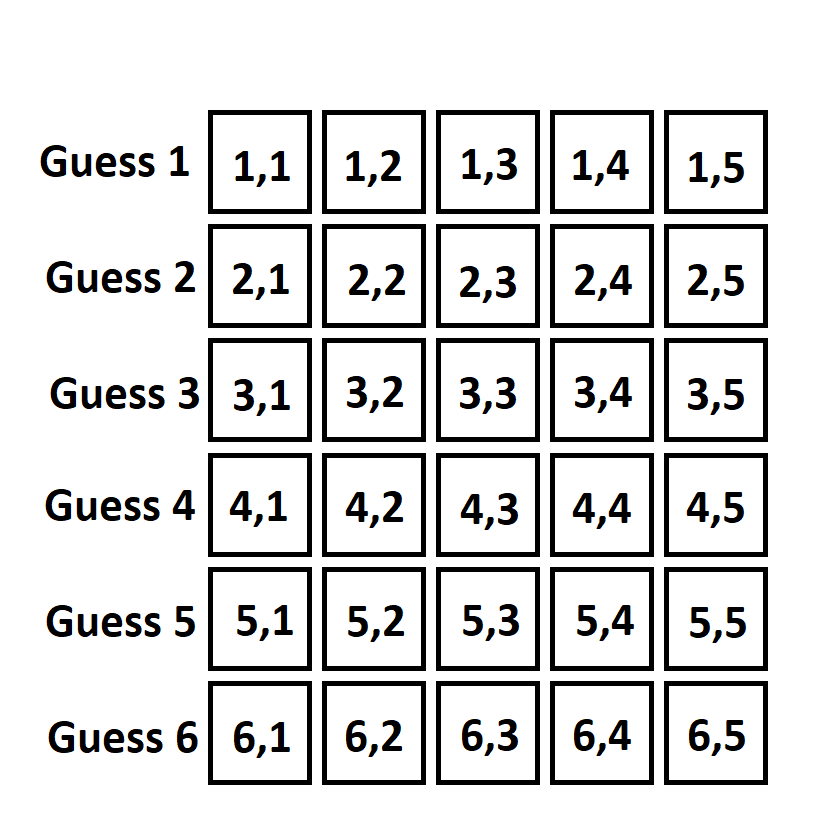
# Project Summary



Our goal is to find an optimal solution to wordle using logical propositions. We will have a randomly sized dictionary of words to choose a word and have our logical proposition find words to try out and using the clues the logical proposition will search for better words to guess.

# Propositions

Wordle is a 5 letter word guessing game with 6 tries which equates to 30 slots to put letters in. Each of the letters can be put into 4 different states . (x, y) denotes the location on the board.



ROW = [1, 2, 3, 4, 5, 6]

COL = [1, 2, 3, 4, 5]

STATUS = ["CORRECT", "INCORRECT", "PARTIAL", "EMPTY"]

LETTER = ["A”...”Z”, “BLANK”]

* Slots(r, c) Each row has a column
* SlotStatus(r, c, s) each slot has a status
* SlotLetter (r, c , l) each slot has a letter



# Constraints

Basic constraints

* Each row maps to every column
  + There are 6 rows for the guesses and 5 columns for each letter, this maps the entire wordle board creating slots.
* Each slot would have exactly one status
  + Each slot would always have a status where we could infer information to give us clues on the mystery word.
* Each slot would have exactly one letter
  + We need to use letters so when we get information we can connect the statuses to letters to give rule out words or pick out words from the list.

Advanced Constraints

* If the letter is blank then the status must be empty and vice versa also the entire row is blank and empty
  + We needed a way to represent blanks
  + An entire row is filled out at once so it makes sense
  + This also prevents blanks being an option for a letter

Solving Constraints

* Our first guesses will be words with 5 distinct letters within the guess\_list (See WLR.java)
* If we run out of letters in letter bank allow duplicate letters
  + Our list of words may not contain all 26 letters so there is a letter bank containing all the letters in the Wordle.guess\_list (See WLR.java). If we run out of words to guess with 5 unique letters we can start using partially correct letters and correct letters in our guesses.
* If there are 5 partially correct and totally correct letters start trying to guess the word
  + If this case occurs there should in theory be enough information to quickly guess the word.
* If the letter is not in the correct word, then all words in the word list with the incorrect letter should be removed.
  + Not only the partially correct and correct statuses give us information, but the incorrect status does too. This will dramatically decrease the amount of work for it (Words are removed with invalid() from WLR.java).
* If the correct letter is in the right position, then remove all words without the correct letter from word list.
  + Similar to incorrect guesses for correct guesses if there isn’t a letter in that position we remove the word.
  + This accounts for words with multiple of that letter.

# Model Exploration

**Process:**

*Part 1: Things we did up until now:*

*Things we discovered while coding the project and such*

*Difficulties we faced and steps taken during the project*

We had ended up not achieving all of what we had set out to do. However, we had completed setting up the propositions of each slot in the wordle game, their letter, status and even lists of constants (An example of this being STATUS[] which holds values of incorrect/correct/partial/empty).

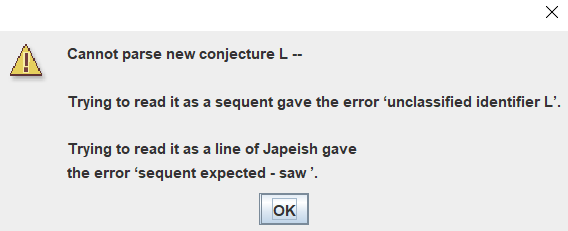
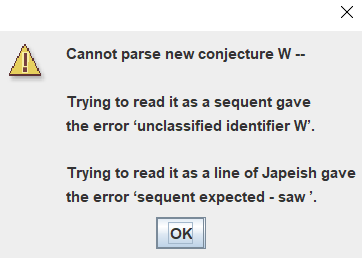
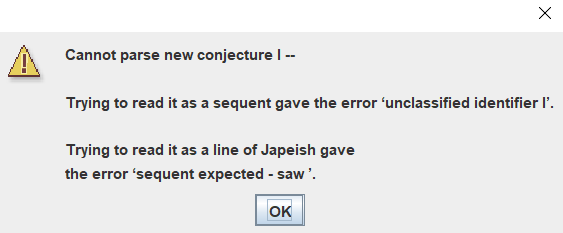
We had also implemented a series of constraints that take information about the status of a slot in previous rows and columns and translate it into information for the next guesses in the wordle board. It was difficult for our group to think about what types of constraints would be fitting to help the SAT solver do its job. We had decided on taking slot propositions and using implications to determine the states of the word (ie. If a slot is incorrect, we use our supporting file to take all of the words with said letter in the slot and remove them from possible answers).

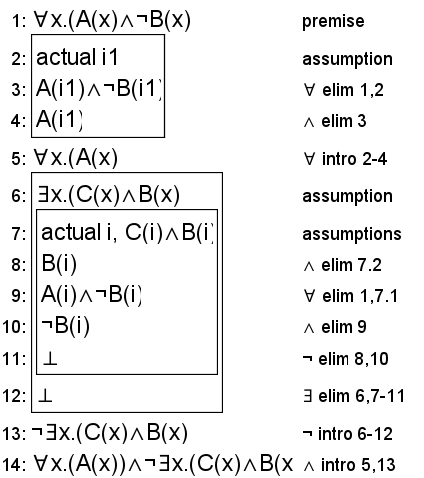
At this point we had moved on to working on our supporting file WLR.py, we intended to use it to set word propositions to true or false based on a given list of words. However, we had not fully implemented the connection between the two files.

*Part 2: Things we didn’t implement:*

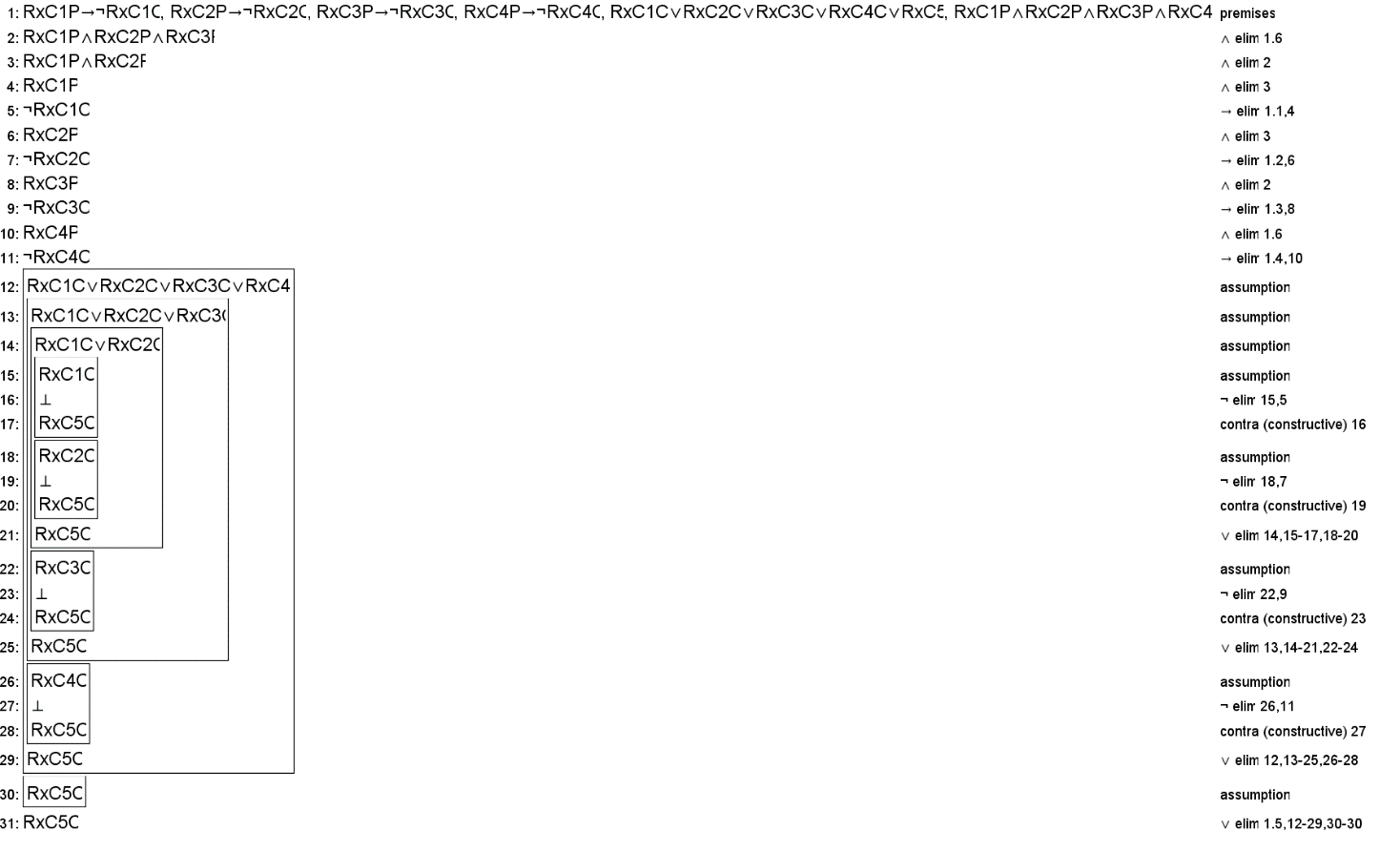
We had intended to extract a list of possible words from the SAT solver and compare it to our own guess list which we had created from the WLR.py file. We also planned on fully implementing our “invalid guess remover” function inside our WLR.py file and continuing to make guesses much like an actual Wordle game (continue six times or until the word is guessed). Unfortunately, we had not even addressed the idea of “turns” and continuously making guesses, so we had settled for a system of attempting to take in an example set of data and trying to interpret a guess instead.

# Jape Proof Ideas

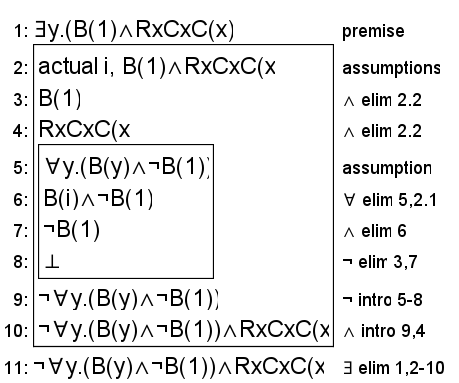
* Proof 1 ∀x.(A(x)∧¬B(x))⊢∀x.(A(x))∧¬∃x.(C(x)∧B(x)): If we remove words with incorrect letters Then there will be no words with incorrect letters
  + A = Words
  + B = Letters
  + C = Incorrect letters
  + (W, L, I are illegal arguments for jape apparently)



* Proof 2, If a partially correct letter is in slot 1,2,3 an slot 4 in our guesses then it must be correct slot 5, RxC1P→¬RxC1C,RxC2P→¬RxC2C,RxC3P→¬RxC3C,RxC4P→¬RxC4C,RxC1C∨RxC2C∨RxC3C∨RxC4C∨RxC5C,RxC1P∧RxC2P∧RxC3P∧RxC4P⊢RxC5C
* RxC1P→¬RxC1C,RxC2P→¬RxC2C,RxC3P→¬RxC3C,RxC4P→¬RxC4C : If there is a partially correct slot at Row x col x then that slot cannot be correct.
* RxC1C∨RxC2C∨RxC3C∨RxC4C∨RxC5C : For the letter it could be correct in slot 1, 2, 3, 4, or 5
* RxC1P∧RxC2P∧RxC3P∧RxC4P: The letter has appeared partially correct in slot 1, 2, 3, and 4
* ⊢RxC5: Having 4 partially correct letters in different slots means the last slot is a correct letter.



* Proof 3: If there exist a correct letter and it is in the correct slot, then all other letters cannot go into that slot. ∃y.(B(1)∧RxCxC(x))⊢¬∀y.(B(y)∧¬B(1))∧RxCxC(x)



# First-Order Extension

*Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated.* ***There is no need to implement this extension!***

**Updated Propositions:**

**E(s):** Means that the slot (s) within the guess is empty

**I(s):** Means that the slot (s) within the guess is incorrect (wrong letter)

**P(s):** Means that the slot (s) within the guess is partially correct (wrong slot, right letter)

**C(s):** Means that the slot (s) within the guess is correct (correct letter in correct slot)

**U(s):** Means that the slot (s) is a unique letter compared to the other slots inside the guess

**W(g):** Means that the guess (g) is the correct word from the word bank

**L(g):** Means that the guess (g) is a word inside the candidate list of possible words

**G(g):** Means that the guess (g) is a “good guess”

**B(g):** Means that the guess (g) is a “bad guess”

**R(g):** Means that the guess (g) is a guess that has already been made

**Updated Constraints:**

– There exists a correct word from the word bank (AKA a possible answer)

– There exists a guess where all slots are correct slots, implying that the guess is the correct one from the word bank

– If a guess is repeated, there are no correct slots or there are slots that are incorrect/empty/partially correct.

– For all slots, each slot is unique in comparison to the other slots inside the guess and the guess is not considered a “bad guess”. This implies it is a “good guess”.