Our computational model was designed with a human-like strategy in mind. The situations we modeled were the many steps in the gameplay loop for hangman. The standard gameplay flow for a human is as follows: First, you are told how many letters there are in the word. Then, the player thinks of a word that matches the current understanding of what it could be (currently only the length). Thinking of this word, the player chooses a letter from it to either backup their thoughts or prove them wrong. If they are correct they return to the third step, and continue trying to guess at the word they had previously believed. If however they are incorrect, they return to the second step, but this time with more information than they had the last time they were there. Instead of having a computer choose the word and the human be a player, our computational model flips the script. The computer becomes the player, and the human becomes the proctor. Our model is quite resistant to misinput. Not only does it tell the proctor what answers it will accept, it is also resilient to inputs that don't make any sense and can even bounce back from being lied to! We are incredibly satisfied with not only how human-like our model is, but also how we were able to notice that it has a tendency to choose letters that it "likes" (for lack of a better word) more often than others. Though we did not hard-code this functionality, it seems to have emerged as a logical behavior for our model. We wanted to avoid telling it to guess the most statistically likely letter, but it seems to have learned on its own that the letter "e" is a safe guess, because of how many words contain it in it's knowledge base.

One significant limitation to our program is it's relative inefficiency at higher dictionary sizes. As the size of the dictionary increases, the runtime of the program increases at a rate higher than linearly. In order for the model to function in a reasonable time, we can not exceed a dictionary size of about 1,000 words on our school 's computers. If we choose a larger size knowledge base, we very quickly start running into runtimes of minutes for each input. As it stands, the response is nearly instant for the 1,000 word dictionary. There is a notable tradeoff between the two, but on our personal systems we were able to push it upwards of 10,000 words. We did notice that if there is only one word left in your dictionary that the model knows fits the criteria, even if you tell it that is not the word, it will keep guessing the word. Otherwise, most inputs are no cause for concern.

Taking a cognitive model to mean "Some computational model that behaves in a way that a cognitive system would have behaved in the same situation", we have hit the proverbial nail on the proverbial head with our model. We have been able to model a human to human interaction with a computer imitating one of the players in a way that feels smooth, and real. There is active belief revision within our model, after every input given by the human proctor. Even when our player - the computer - is correct about it's guess, though it will continue to try to guess the same word that it had been guessing, it is actively trimming its dictionary down to only the words that fit. As mentioned in our response to task 4, the set of words contained in our model is directly translatable to a

problem space that contains those words, and the rules of the game hangman. This computational approach to modeling the problem is a step away from a simplistic cognitive model, but the level of detail has not been sacrificed. This is because the behaviors outlined in this cognitive model are as close to identical to those seen in adept hangman enthusiasts as can be achieved with the scope of this course.

Though we are proud to identify with the software that we have developed, it is clear that there are areas which we would be able to develop this product further. Foremost, we would like to look into expanding our knowledge base to encompass a larger portion of the English language. This raises the additional question of possibly incorporating other words from other languages. Before either of those goals can be met, we must find a way to lessen the processor requirements taken by our model. We must look into a way for the same functionality to be compressed into a simpler script. After some discussion, we have decided that it may be easier to implement this larger database version of our system in a language such as Python, thanks to our familiarity with it. This is not to say that the same can not be accomplished in prolog, just that we feel it would be a less efficient use of our time to try to do so. Ideally, we would be able to add words to our knowledge base with no noticeable impact on the runtime.