Our program’s main function is to amaze by successfully reading the mind of the user. Though it may sound incredible, the program’s functionality is rather simple. It first prompts a user to come up with a number from 1-63. Until the guessing pool is left with one potential number, the program creates random cards of 32 numbers, then asks the user if their number appears in the card. Finally, it outputs the number to the user. While the user responds to the program, they will be serenaded with music, creating an all around enjoyable experience for our user.

Our team faced many challenges when developing this program.It was proven quite difficult to convert our algorithms into MIPS and that is where we struggled with the most. Even after converting into MIPS, we also struggled with combining our separate codes together into one simple program. There were times where we wrote codes that overlapped or overrode another members instructions which we then had to fix on the spot whenever we met up together. Not to mention the continuous crossing of using the same registers to store data made it difficult to keep track of bugs when combining our code.

I now have a better understanding of MIPS as a language overall. It is different from high level languages in that you have to designate space and memory for everything you store for use in the program. Because of that, writing logic becomes more difficult and requires more thought than a normal program in a higher level language. Since I worked a lot on the display and output of the program, I feel as if I have a better understanding when it comes making sure the user interface looks clean and the program works efficiently for the user. Creating input and output was much more difficult than in C++, especially when I was working with outputting the array of numbers every loop. I was much more challenging than C++ when you need to make sure you have the write index for all the elements of the array.

Many parts of the program required us to create algorithms to solve our individual problems. For the card generation, we had to use system calls to generate random numbers to fill the cards. We used a linear search to only allow unique numbers into the array, avoiding any repeats; once complete, we sorted the card array using a selection sort. With output, we had to create a loop that’d print ascii characters and digits of the card array while keeping in mind the alignment of the numbers by digit. The final algorithm was finding the number using the cards and user input. We created an array of numbers 1-63 (deck) and cross-referenced the numbers on the given card. For each number of the card, we converted the number to an address in the deck and converted it to zero. Once the amount of numbers in deck equated to one, we printed that number out as the found number.

Our team made sure to divide up the work in order to make sure we got all the sections done at the same time. I worked on the display and output of the user interface while Joseph worked on the generation of the cards and Henry worked on the eliminating the cards from the pool so it was easier to guess. Because of our pre planning of our methods, we were able to combine them together into one program with only a couple painful overlaps that caused more issues than we thought. But we stored our data the same so it was easy to just add our code as methods and call them in the main function. We all worked hard on our individual tasks and because of that, we were able to get the assignment done within our time limit.