Requirements Specification

We made a two player battle version of the classic game Snake. It is very similar to the original game but rather than ending when the single snake dies, the game ends when a single snake is left alive. There is still only a single fruit on the game board at a time so both players have to fight over them to grow.

**Requirements**

We had to find a way to get live input from the keyboard as well as deal with two players pressing buttons at the same time. We also needed to work out the best way to deal with snake on snake collisions and the order than the different body parts of the snake were added and deleted.

We did our project in the console so we did not have to deal with setting up a windows project. In many regards this made our project easier to deal with but, because we were using the console, we had to deal with a lot of spacing issues when we output to the screen.

**Assumptions**

We assumed that once we had finished implementing a single snake the second one would be as easy as copy and pasting and then change a few values. This turned out to be very wrong as all of our collision timing and detection was off.

We also made our board made up of squares of two characters each. We assumed that this would not affect our project but we ended up having to deal with quite a few off by one errors pertaining to the output and how we set up the board.

**Design**

The program utilizes one structure called a location which stores an x and y coordinate as a byte. This was useful because our board was never more than 20 wide and tall, so two bytes could store all of the information.

None of the functions in our program pass information. They all simply edit the values in memory via the registers, but data is never stored in registers between functions. This simplified the process of passing data and making the game function properly. The most important part of the program is a function called addHead which checks the current direction of the snake, then adds the head in memory, deletes the tail, adds the head to the board, then checks for collisions. To do this, almost all the rest of the functions are called within addHead with the exceptions of main, initBoard, and editKey. Main is obviously the launching point and should not be called from addHead and editKey is only called from main. editKey checks the current key press and edits the aSnake’s direction if the keypress is on the keyboard and bSnakes direction if the keypress is an arrow key.

initBoard is only called once in the course of the program at the very beginning of main because it sets the initial position of the snake and outputs the walls of the board as well as seeds the random generator.

There are two functions to remove a tail, removeTailA and removeTailB. They remove the tail from the corresponding snake both in memory by moving the front value out of range and shifting the rest of the values in the vector up by one as well as outputting a space to the screen so as to clear the board at the location. The function addFruit places a fruit at a random location on the board where there is currently no snake. This is done by looping through both snakes and making sure that the locations don’t match.

**Pseudo Code**

1. Initialize the board
2. Get keypress and save if it is wsad or an arrow key
3. Move both snakes heads in memory
4. Delete the tails so long as the snake’s head is not at a fruit
   1. If is at a fruit, don’t delete the tail and move the fruit to a new location
5. Show the heads on the screen at their new locations
6. Check for collisions
   1. Exit if there is a collision and output the corresponding win message
7. Goto 2