README – Sliding Blocks

Joe Salisbury

David Pick

1. Division of Workload

For this project we met with each other many different times to work together on the code and on the overall layout of our project. We live on the same floor in BSB so it was really convenient and worked out well for us to meet so often. When we met together to work we would talk out ideas of how to do what part of the program we were working on. We would often get a part working then spend a fair amount of time debugging the specific method to perform the way that we wanted it to perform. If we were in the middle of solving something when we needed to end our meeting, we finished up what we were working on and then explained what we had accomplished once we got back together. If ten bonus points were to be awarded I would say eight points to David and two points to me. David played a much larger role in planning out what exactly needed to be done and how we should go about doing it. While we both contributed to the code and the planning, David really put a lot of time into it.

1. Layout of Program

We stored the current tray configuration inside of an array. This array was of type Block which we created in our program. The Block class stored the upper right row and column of the block as well as the height and width. We also stored the goal configuration in an array. Based on the specifications that were entered in the command line argument, we generated what that goal configuration was. We stored the current configuration in a linked list as well that took the current array configuration and a linked list of the moves. We used a linked list for the moves so that backtracking would be made easier because we just needed to delete and create new nodes. The linked list took in all the previous configurations as well that way it would help with backtracking. Also we made a hashing function for to apply to each board configuration. Specifically, our hash code function seemed to work fairly well. It takes the hash code for each block and then stores it in a hash set. Each configuration has a hashset. All the configurations’ hashes are stored in a linked list. This was useful when we were determining if the current board configuration was equal to the goal configuration. If the hash code for each was the same that would mean that we reached the desired configuration. Also this made it easier to check to see if a previous configuration had already been reached by simply using the hash code. Our findMoves method looped through each block in the tray and then looped through the height and width of the current block to see if it could successfully move up, down, right, or left.

1. Debugging Output Options

Our program implements several debugging operations that are available to be printed out if the user wishes. Many of these debugging options were ones that we had used along the way in making our program solve puzzles correctly. Our program checks to see what the length of the command line arguments are and if it only has two inputs, it will not print out any debugging mechanisms. If the length is three, it will look at the first of the three inputs and determine which type of debugging we want to use. We have three different types of debugging options available to the user. First, -oMoves will print out the entire list of possible moves available from the current configuration. We have a linked list of moves, so we simply print out the linked list of the possible moves. The –oBoardConfigs will print out the current board configuration that our program is at. The –oLastMove will output what the last black and its position and then the position that it has moved to. These were useful along the way in determining where our program ran into problems and what methods we needed to go back and fix. The print statements were the only way that we could get a representation of the board so it was extremely important that we used a way in which would be beneficial to us to see what was happening, where it was happening, and when it was happening. The three options that we specify to the user were three that we used often when working out bugs in our program.

1. Known Bugs

We are likely to have a heap space error at some point.