Deven Brewer 19201772

Dr. Liliana Pasquale

COMP10050

28/04/2020

Assignment 2 Design Document

When designing this project, I decided to take an almost object orientated approach by storing all data in structs (objects) and manipulating them with functions divided into modules. I took this approach to allow large scale changes (like changing from text-based IO to graphical IO or playing Knots and Crosses instead of Domination) by making a few small changes and swapping a couple of files. I broke up the parts of my code into five sections: the object definitions, the game logic, the game initialization, the input and output, and the driver code.

The object definition section consists of a single header file in which objects are defined, and no source file as this section has no methods. By defining our objects in a dedicated header file we are able to import it into the rest of our modules with little overhead. The game pieces are stored as a linked list because this allows us to naturally move and store as ‘stacks’. All game data is wrapped up in a single ‘gameState’ object, which allows us to allocate and free all game objects with just two functions (one for initialization, and one for freeing).

All object initialization and freeing is handled in the Game_Init section, which consists of a header file and a source file. I wrote 2 functions for every object defined in the object definition section, one which allocates and initializes memory for an object and one which frees the memory allocated in the first after we have finished using that object. Because our objects are tiered (our gameState object holds the player objects and square objects, the square objects hold gamePiece objects, etc) we make use of the functions for initializing and freeing the lower-level objects within the higher-level objects, allowing us to naturally build up to using a single pair of functions within the driver code.

All input and output is handled with the ASCII_IO section, consisting of one header and one source file. I chose to group the input and output together instead of placing them in separate files because they are not overly complicated, and it helps cut down on the disordered mess than can arise from having too many files. Originally, I had intended to make use of SDL to create a GUI, but after running into some problems installing the necessary libraries, I decided to resort to using built-in IO functions with the STDIN and STDOUT file streams. I did, however, make sure to design the modules in a way that would allow for a future update replacing the ASCII_IO files with Graphics_IO files with few changes to the driver code.

The game logic is handled in a dedicated module, again consisting of a single header and source file called Game_Logic. The main goal here was to handle the peculiarities arising from the OOP approach within a single module, cleaning up the driver code. In order to do this I had to make use of a couple of strange algorithms, like the one which moves stacks, which has to make use of a pointer to a pointer in order to allow us to move a stack onto both another stack and onto an empty square.

The driver code is located entirely within the main function in main.c. The goal here was to create code that is extremely easy to read and is almost agnostic about the game being played. The main bit of code is almost literally “while you can move, print the board, get move, do move, swap turn”, a series of steps which takes places in almost every two-player game ever made, which tells us that we’ve successfully contained the peculiarities inherent to Domination without our modules.