**Group D: Final Planning Document**

B. Carver, D. Garcia, M. Pacheco, T. Steele

University of Wyoming

COSC 3011: Software Design

May 13, 2022

**Beginning the Project** (Program 1-2)

Going into the project we first met to assess everyone’s strengths and weaknesses, to then assign roles and begin brainstorming the project. It was decided in our first planning meetings that Beckham would take care of the documentation, Matt and Tyler would do most of the coding, and Derik would complete the UML along with any extra coding. This would later be revised.

**Program 1:** Was about setting up the environment so we could later implement the needed functionality. To start out we planned loosely around functionality we needed now, and then narrowed down the specific java library classes that would accomplish those tasks. Once we knew what java library classes we would be using, we specified our plan around them.

We settled on the following java tools/classes: JFrame, JPanel, GridBag, and ActionListener. These tools cover most of the needs for our program; drawing, rotation, positioning, etc. By implementing our own classes that extend and utilize these tools we would only need to create the game logic that interacts with the classes to complete the program as we understood it. To start the programming assignment our plan remained relatively simple, finish the JPanel based class that displays JFrames using GridBag, and can listens to actions. We also planned to create a JFrame based class called Tile that would allow greater functionality. However, this Tile class would not be implemented until program 3. This was the first rendition of our plans and was for the time loose and indescript but based in workable logic we would expand in program 3.

**Program 2:** We started to see the limits of our planning as features were being added as fixes and updates without a long-term goal. Our GameWindow class was fulfilling its needs as an environment, if not only because it contained the entire program within its code. Nonetheless it was sufficient for program 2’s needs and we looked towards program 3.

**Completed Design Plan** (Applicable Throughout)

Before completing program 3 we developed concrete long-term plans for the project. Using Derik’s UML diagrams and recommendations from the assignment sheets we identified the use cases for java tools and how we could use them within our own class structure. For specific feature implementations we sectioned these into bullet style goals for the program. For the entire project we settled on a design structure that the team was to follow and remember. The large scope plan for our design structure is below:

Using the following java tools/classes: JFrame, JPanel, GridBag, ActionListener, and MouseListener, we can create an effective programming structure built off effective class structures, for the ease of the software’s design.

Using the JFrame/JPanel relationship built into these classes, we planned to extend them to our needs. Specially a GameWindow class that extended JFrame, and a Tile class that extends JPanel. Our GameWindow class would be the running space for our program, and therefore it needs to interact with the user. So GameWindow should implement ActionListener and MouseListener. To prevent GameWindow from getting overloaded, the logic and functions that the Listeners will call should be connected to classes separate from GameWindow. Since a JFrame can contain JPanel, we will subclass JPanel into a Tile class that we can place and parametrize as needed. If tiles need new functionality, such as rotation/placement data, or the ability to load new data, we can add these with minimal change to their calls within GameWindow. To structure our Tile placement within our GameWindow we will use GridBag, which allows us to set colors, placement, and line width among other features.

Since GameWindow will be the outer framework of our program we will be able to work on different portions of the code simultaneously. This is because once GameWindow can hold tiles, and can send Listener calls, additional functionality can be implemented in separate classes that GameWindow simply needs call. With this structure we hope to reduce the time spent finding bugs, as when a new feature is implemented it should not break previous systems and will only present bugs within itself.

**Implementing the Design** (Program 3-6)

**Program 3:** Using this encompassing design structure, we implemented the specific features and fixes laid out in our original planning document for program 3. These were implementing a reset button, access to a new input directory, the reading of binary files, and various visualization issues. These were accomplished with our new design plans which included separate classes for Tile, TileDP, and Reader. This cleaned up GameWindow and allowed for easier implementation going forward.

**Program 4:** Tyler had a family emergency and would be leaving school for the remainder of the semester. This meant we needed to reassign roles and the following was agreed to have Derik to continue the UML and the I/O code, Matt to pickup the bulk of the code, and Beckham to continue with documentation and help more with code.

It was at this point the program was beginning to look like an actual game. We needed to implement win checking, dynamic setup, rotation, and a placement check sound. Following the design, we had planned, rotation was easy to setup and utilized the functionality built into JPanel (now Tile) and was added as a function call within GameWindow. GameWindow would listen for a right click, and conditionally report the data found from the listener to the rotation call. Dynamic setup was a minor modification to the original setup code, this also preserved the new game button’s functionality. Win check was decided to be checked every time tiles were moved or rotated as those could create a win condition and the code to check if it is a win was relatively light. For placement check we only needed to augment the current placement code.

**Program 5:** This time we ended up not giving the program the time that it needed for completion. However, for what we did implement it followed our design goals. We began all -but did not complete-new classes for FileSelector, general Input/Output, and updated TileDP (now TileDataPoint) to prepare for the full functionality coming for program 6. Being able to use FileSelector as a separate class called from GameWindow made its transition into the codebase relatively easy. Having our I/O class as a median accessible for both FileSelector and GameWindow followed our design goals and was highly effective for identifying bugs and working on the codebase simultaneously.

**Program 6:** we first prioritized fixing program 5 issues first. This included the proper prompt for filename, the tile flash, and the timer. We also completed/improved the classes created in program 5. Working on program 6 and program 5 requirements simultaneously was made easy by our design. Beckham was assigned to complete the final planning document and timesheet. The code specific to program 6 was relatively easy to implement, the reader class had to undergo minor modifications and the buttons within GameWindow needed minor logic changes.

**Design Conclusion:** Regarding our design set out in program 3, it has come to fruition and been successful. Our GameWindow class could be considered overloaded based off its size. However, the way that our program is set up doing additional file separation would have little improvement on abstraction and would be mostly aesthetic. For the few things currently in GameWindow that could potentially be put in an outside interface, they would only end up being very short files that on their own have little logical or abstraction value. Creating an auxiliary interface file for the sole purpose of containing methods from GameWindow would again potentially look prettier but would serve very little else and make the code arguably less maintainable.