**The Maze Game**

Full Design Process

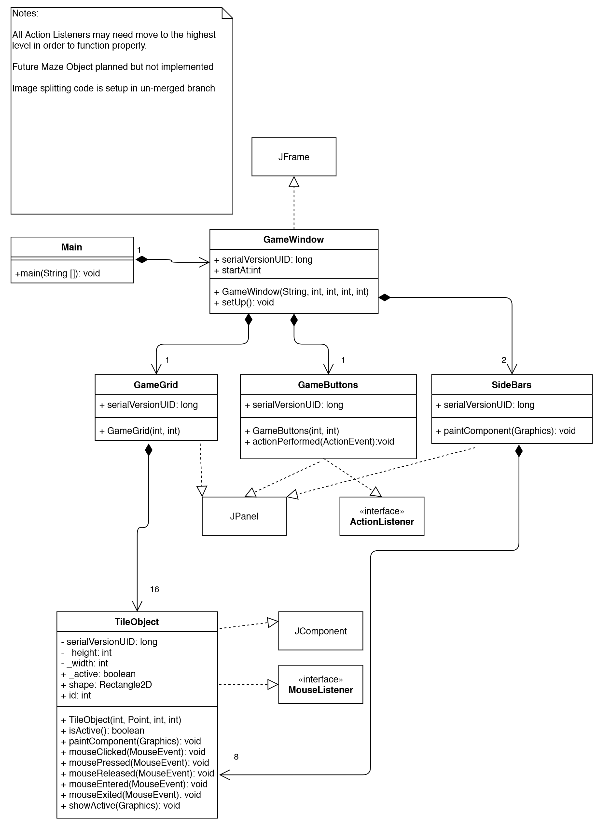
Group O

Matthew Floyd, Tristan Redding, Ben Wabschall, Yun Chi, Bryce Ostrem

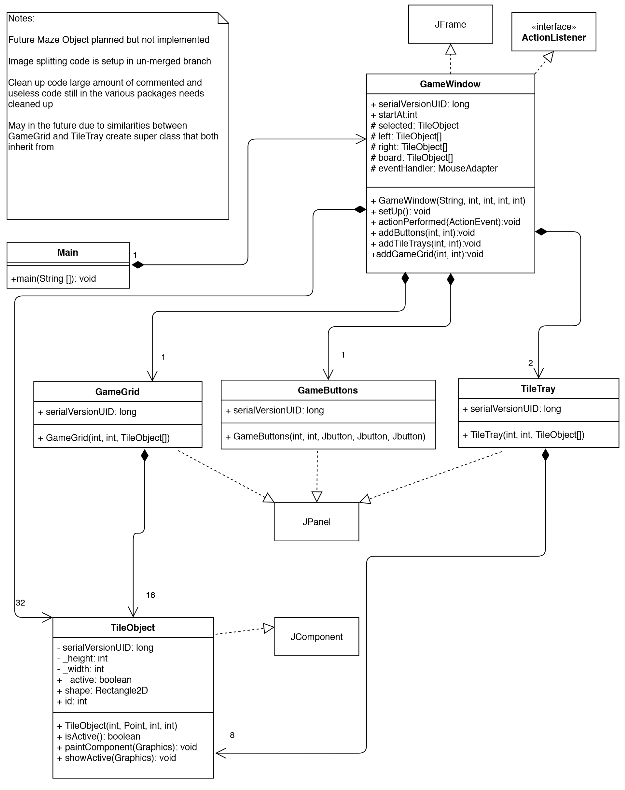
This semester-long project was a huge learning experience for all those involved, primarily a lesson in dealing with a job in the real world. It is impressive to look back and see the growth we have all had from dealing with unexpected requirements, problematic code refusing to work (especially on a deadline), and working as a team.

In the beginning of the design process, the team was very good at identifying potential features that would be used in advance. From the meeting minutes of 2/9/20, we brainstormed on how to check the success condition of a completed puzzle and predicted the rotation functionality. We also spent way too much time on making the project dynamically resize, a feature that was not needed due to requirements.

Our first major design hurdle was the lack of knowledge on how Java functioned. Everyone had come from varied programming backgrounds but none of us had extensive prior experience with Java and the eclipse workflow. Learning about this was a struggle as, at the same time, we had to make our first major design decision. What do we use to create the grid system, the building block that the rest of the project works on? Our first UML document (3/12/20) illustrates our first attempt to structure the grid system using GridLayout where the grid is the sole user of the TileObject and each Tile is responsible for understanding when a user is clicked. Interaction is handled at the Tile level and the grid is just a container to keep it organized as seen in the UML.

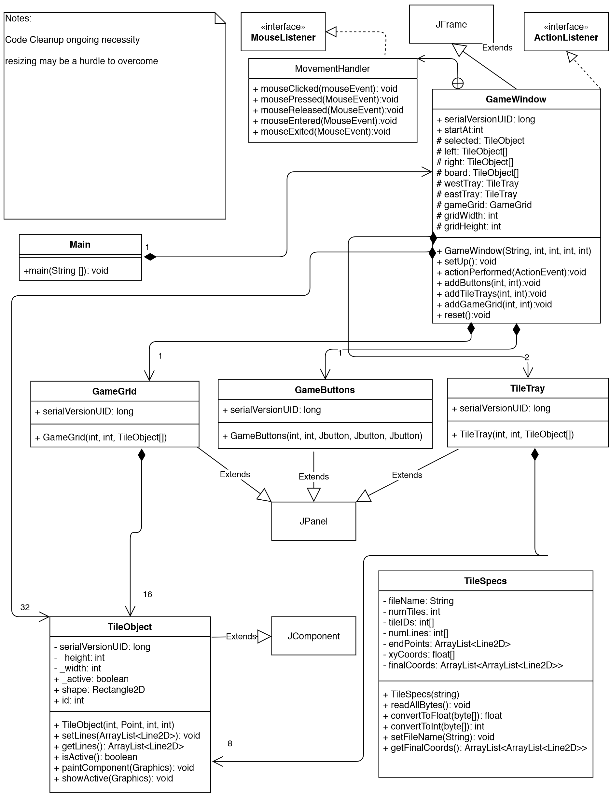


We quickly realized this was an unsuitable design flow as the Tiles needed to be able to work with not just the GameGrid but with the rest of the project as well. To this end, we moved the interaction of the user to be handled at the GameWindow level to make this work as well as allow easier functionality changes later on in the project. The UML document (3/13/20) changed to be this:



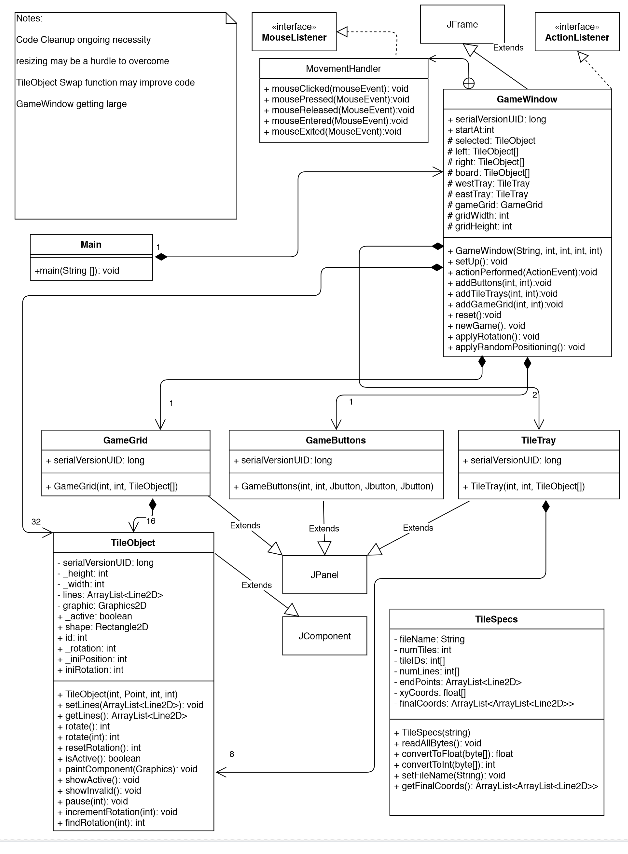
Our next hurdles were first; implementing the snapping feature of moving TileObjects into “slots” on either the left and right side TileTray or into the center GameGrid and second; loading in raw binary data from a .mze file. This is the point where we really started to fall into more clearly defined roles based on which parts of the code base needed to be modified. Yun Chi and Ben were the experts on anything file access / reading related. They were responsible for converting the raw binary data into usable floats for the rest of the project. Matt was an expert for any parts related to the TileObjects: drawing lines, movement, rotation, etc. Also functioning as scribe for planning documents. Bryce was the expert on all research and testing. He made sure the team was well aware of any problems from the style guide or other mistakes. Tristan was a more all-reaching expert. While not tied to a particular area, he helped everyone with fixing / making all parts of the code work correctly. At many times when a checkpoint was due that day, he would stitch together code and create a working product to ship out for the deadline. He also created all UML documents.

With these tasks clearly planned out to the respective team members, our plan for Tile snapping and file handling was portrayed in the UML document (4/8/20). Additionally, the reset function was added at this time, utilizing the newly created file handling code in TileSpecs to double as a saved state that reset could use to restore the game to its previous instance.



We quickly settled into a routine for new checkpoints. At the start of the new checkpoint cycle, a meeting would be held to discuss the new requirements, assign tasks to each team member, and fix any problems from the previous checkpoint. From there, a second meeting would be scheduled in advance to check in with everyone and help out those that had a larger portion of work or had trouble.

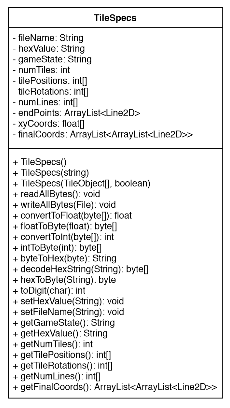
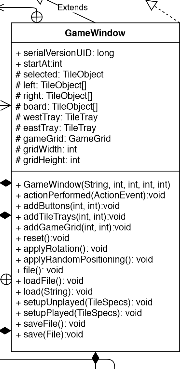
This checkpoint fixed the outlines of the center grid which were not present in the last checkpoint and eliminated any anonymous classes we had. It added rotations to TileObjects as well as error windows for save / load. Our earlier questions about resizing the game window were also answered as the hard set 100 by 100 size for tiles were placed in the instructions. The checkpoint also generated a random order of tiles, creating a “game” element to the project, it also approximated an even distribution between the 4 possible rotation states for each tile.. We utilized this new feature to implement the functionality of the New Game button. The UML document(4/17/20) reflects these changes.



Our routine for handling new checkpoints proved to be a valuable adaptation which saved the team large amounts of time that would otherwise be wasted in extra planning. For this cycle, the end of April, played and unplayed versions were added to the load sequence. Saving a game became a feature as well as file selectors were created to facilitate the use of both loading and saving.

For design decisions, we decided to nearly copy the loading process to use as the saving process. Since the two operations were just the reverse of each other, it made the most sense to use the loading code to create the saving code. With this cycle, our overall design was set as the foundation for all the new updates was already created. New features just built upon the existing framework created over the past checkpoints.

For this checkpoint, only the GameWindow and TileSpecs classes were modified. The UML document(5/1/20) changed as such.



For the final checkpoint, we had three main goals: implementing a win condition to let the user know when they have successfully completed the game, implementing a timer to let the user know how long it took to arrive at a correct solution, and creating the final documentation, the big essay / planning document that details our design journey from start to finish.

From a design standpoint, the win condition was interesting. The question of what was considered a “won” game was up for debate. It is certainly valid to assume that all tiles being present in the center grid in the correct positions and with the correct rotation, 0 degrees, constituted a won game. It could also be reasonable that the same solution flipped to another rotation could also be perfectly valid. In this case, we would have 4 separate won game states. In the interest of simplicity, we decided to make only the 0 degree rotated version the valid solution.

The implementation for the timer was much more straightforward design wise but was the most difficult code wise of the three goals. Specifically, its interaction with the save and load utilities created some trouble as it needed to save the current time or load a new time to the current timer. Also, in a new game, it needed to start only on the first move by the user.

The full design document was the simplest code wise for obvious reasons. Detailing the entire process made us realize exactly how much we had progressed as a team and how much we learned to adapt to different circumstances. Perhaps it is too meta to analyse the design document inside the design document?

In any case, this semester-long project was a huge learning experience for all those involved, primarily a lesson in dealing with a job in the real world. Learning to work better as a team is an invaluable skill, one that we will continue to work on in all of our careers.

The final UML document(5/15/20) is below.

