Design Document

This program has an awesome zoom function, a bunch of types of animations, and layering.

The game is drawn as background and foreground. The foreground is kept as the “state” of the game, and takes all the input. This means I can have the main menu, pause menu, and any level of the game be a state and respond to input differently. Beyond that, all of the levels have a “zoom” function that lets you only interact with one part of the level at a time. This makes the program more efficient by only drawing and updating one part of the level at a time, and allows each aspect of the level to have different inputs and tutorial text. For instance, the build button calls a different checkSolved function for each stage of the puzzle, even though it’s the same exact button. Changing between stages is as easy as appending and popping an element from the depth.

The updateInfo function let me update the higher puzzles based on what happens on the lower ones when you switch between stages. This lets the update happen once, and so not slow down the game. A lot of the functions in the game have been optimized to avoid lag, since lag would be very obvious when the static rolls slow down.

The background is drawn and updated independently of the rest of the game, so I can switch it out or keep it in between levels without issue. The static rolls are the first animations that you see. To keep it from lagging, I only calculate all of the alpha values once, and it shuffles them and rolls them from bottom to top. That way, they’re always different, always changing, and doesn’t take too much computing power.

There are specific animations for each level of the puzzles, with pulses being the most obvious ones. I found out that it’s possible to have an infinite loop, making it impossible to move on, so I made the pulses keep track of their own path, which gives it a cool targeted look.

There’s also a cutscene class, which is kept in the levels for seamless ordering of levels and cutscenes. The cutscene class takes a background image and animates things over it. In my game, it only has writing text over the image, but it would work with any animation class targeted for this kind of integration.

The puzzles are all solved recursively, which allows for multiple solutions to the same puzzle. This way, I can design lower levels to have multiple solutions while the higher ones only work if you started off with the right base. They also build, so that each stage of the puzzle is based on the lower level. You can only solve the top level puzzle if all the bottom levels are solved correctly.

The puzzles are contained in a wrapper that makes building a level incredibly simple. Even my most complex levels are just 30 lines of code. From the point that I draw a puzzle on paper it takes about 10 minutes to put it into code. I make all of the levels and cutscenes in a levels file and import them into the game through a dictionary, so I can keep some order. Each level has a next level, a level name for saving, and a music file name if there’s a music change.

The stateChange function basically takes care of all the transitions. It resets the puzzle you were on, fades in and out, and does the music change.

The main menu will look different depending on whether or not there exists a save file. If you go back to the main menu from in game, it saves your progress and you can load later. Going back to the main menu from in game shows a loading screen, as if Albert is being loaded.

At the start of the project, I spent around 14-15 hours just on structuring the code for later. I didn’t add in any graphics until the bottom level puzzles were already finished and coded, along with the zoom function and main menu. To have the bottom level puzzle working I had to have the basic structure for the concepts and associations done too. By the first TP I had the structure for almost the entire game, then I tweaked the more specific graphics, settings, structure, and rules over the next 70 hours to make it into a full game. This meant animations, top-down puzzling, switching between levels, and differentiating between the higher and lower level puzzles (it wouldn’t be interesting if they all followed the same rules).

My user studies tended to all say that the game was less intuitive than I thought it would be, so I put in a lot more help text, and made the differences between the nodes more obvious. The levels I have in the game are generally unchallenging because each one was built to test a new aspect of my code rather than be an actual stumper. These levels, once I continue writing the game (which I plan to) will act as tutorial levels that teach the player how to use each new feature.