Project Description Greg^3

Project Description

This semester, Delta decided to implement a puzzle cube using P5.js. Greg^3 is a puzzle cube where users race against the clock to see how fast they can solve a series of six different puzzles. The user can start with whichever puzzle they like, and the clock starts immediately upon entering the game. Once the user has successfully completed all six puzzles, a win screen animation appears.

The group initially implemented a lot of the project structure procedurally, and we quickly found that this was not going to work for such a large scale project. After realizing the need for change in the project, the group refactored our code to introduce a modular class based implementation that uses inheritance. Each puzzle was implemented as a derived class to our base puzzle class. Once the refactoring was complete, the group then moved to getting our 2D puzzles rendered on the face of the cube. We incorporated the mPicker JavaScript library and WebGL in order to help with the rendering of our puzzles on the actual cube faces.

Technology Used

The technology we used for developing this project starts with P5.js. This was the initial library we started with to begin implementing the puzzle cube. Specifically, we used WebGL, JavaScript, and the mPicker library to create the cube. The visual design aspect of the project was developed using HTML and CSS. Initially we planned on incorporating Selenium and WebDriver for testing, and despite early strides in the implementation, the 3D aspect of the game caused too many issues for us to continue it into the final product. Another intended feature was the use of Firebase for the database, but in order to maintain hosting on GitHub pages, this feature was scrapped as well.

Our team used Github to maintain proper version control, allowing for more collaborative development and tracking of changes. Specifically we used the Github issues and Github projects features to efficiently track and manage bugs and project necessities. Throughout the project our team used discord for the primary communication channel for real-time discussions, updates, and quick coordination. Different channels were set up for specific topics, ensuring a streamlined communication flow.

Process Methodology

In order to successfully drive our project from start to finish, our team agreed on how to handle our Scrum sprints early on. Our process saw us meeting week to week on Wednesdays to discuss our progress from the previous week (the sprint review), what we could have done better (retrospective) and planning our tasks for the next week (planning). From this process, we were able to maintain steady progress throughout the course of the semester. An additional benefit that we saw from this methodology was having a consistently working project.

We managed large changes in the project by developing solely on branches of our repository, never on main. As such, we were able to test and review any changes that were made to the product before merging it into main. This was how we ensured that unfinished changes were not creating downtime for our deployed product. This was largely facilitated by the use of Git and Github.

Another important aspect to our development process was the structure of our project. We started with a single folder full of different files, and most people only knew what a handful of them did. We quickly realized that the setup we were using created a large barrier to entry for new developments. We spent a couple of sprints reorganizing the project into several folders that made sense, as well as using a more object-oriented approach to our puzzles. The result of this effort was a more readable repository and code with less repetition. Moreover, the new structure of the project allowed for a more clear splitting up of tasks among team members.

<u>Product Testing</u>

For product testing, we used Webdriver, separate branches in git for testing, and extensive testing when verifying tasks in the Kanban to move them to the completed section. We used Webdriver to test the functionality of the different cube faces and buttons. An example was when we added a text box where we would enter the color of the face we would like to go to, and Webdriver would jump to that cube face. We also would create separate branches to add new functionality and then test the new functionality ourselves extensively before merging with the main branch and testing ourselves again. Most of our testing was manpower. We also would use the Kanban's pending verification section to see what needed to be tested and would test it extensively before putting the task in the done section. We also had people outside our project group play the game and give us feedback on any issues with the game or any bugs in order to improve.

<u>Interesting Parts of Product Creation</u>

In order to accomplish our goal of creating a 3D interactable game we had to work within a number of limitations of P5.js. In particular there is no simple method of 3D object selection which made it complicated to interact with the Cube and the puzzle games. Two methods that were considered during research were a color coding based selection method and a vector based method. For its simplicity the color coding method was chosen. This method, using our updated version of the mPicker library, uses a P5.graphics object, which is essentially an offscreen canvas. onto which 3D canvas elements are copied with a uniquely identifying color. When mouse events occur within the canvas the color of the pixel at the same location in the offscreen renderer is checked and compared against the identifying colors of each 3D canvas element to determine the object which was selected. This was the first step in realizing our 3D implementation.

The second way we accomplished this was by reorganizing the puzzle elements of our game into a class structure utilizing inheritance and P5.js' feature which allows a graphics object to act as a texture for a 3D shape in a webGL context. A base Puzzle class was designed upon which each puzzle was constructed to interact with the 3D game. This allowed us to break up the Cube into 6 cubeFace objects. The cubeFace class contains itself a Puzzle object as well as a 3D P5.graphics object as well as members used for correctly building the cube and its movement. The cubeFace object calls the individual Puzzle objects setupGame and drawGame functions during the global calls to setup and draw. The puzzle object draws all game elements to the same graphics object in the cubeFace object. Additionally all mouse interactions with the puzzle games were required to be scaled due to the 3D perspective altering the apparent size of objects based on their distance to the screen. This allowed each Puzzle object to behave as if it were the main canvas even though it was simply a drawing rendered as a texture.

How it has Been Received

The reception to Greg^3 was generally positive from those who playtested the game. We had received feedback stating that the puzzles were entertaining and intuitive. The majority of people who had played the game stated that it was easy to play. A portion of the playtesters had engaged in friendly competition with each other to see who could achieve the fastest time.

There was a fair share of valid criticism that we could use to improve the look and feel of the game. A comment we had consistently received was that some puzzles needed to be explained better. A few of the individuals who had tested the game were not familiar with simon says and switch puzzles, resulting in having a frustrating experience. A third of the people who shared their opinions on the game claimed that they had trouble understanding the instructions. In the future, we plan to deal with these comments by creating more comprehensive instructions on our puzzles.

What We Learned

There were really a plethora of learning opportunities throughout the coursework.

- Alot of us hadn't collaboratively used version control systems prior to this, and we honed our skills during this process.
- Alot of us hadn't worked in such a large group before this, and this served as the perfect opportunity to learn how to handle the different nuances that size brings in collaborative programming.
- We needed to figure out good ways to merge all our differing coding preferences
- Alot of us hadn't done any game programming before, this served as a perfect intro project.
- Alot of us hadn't used p5.js before, and we were pleased to dabble around the nuances of it.
- We needed to use WebGL to generate 3d visuals
- We saw a perfect use case of how object oriented programming helped make our codebase more readable, maintainable and extensible

Future Plans

There are quite a few things we would like to implement in the future. By the end of our sprints we were trying to get everything working, but did not have enough time to fix some of the bugs we were having. For example, in the Simon Puzzle, ideally there should be a small wait time between each round so that the user knows when one round ends and another starts. Also, for the blood sugar puzzle, there is a bug where certain numbers cannot be put back or the sum is not restarted when different numbers are put in. Some of these interactions between puzzles would be a good way that more people could successfully play the games.

Another bit of feedback we were encountering from our user feedback was that if one was not familiar with each puzzle they did not know how to play it. It would be ideal to add an instruction screen, like we have on the cube page, to pop up on every puzzle.

We would also like to implement more puzzles in the future. For example, a random puzzle would pop up on any side of the cube. These puzzles can come from a giant database of different puzzles that we created. We could also implement different levels for each puzzle to make it a little bit more difficult to solve. There are a lot of different ways we can implement this project as a whole, but as a group we have decided to keep the cube at 6 puzzles per game.