**LONI Brain Puzzle**

**Documentation**

**CS 130, Fall 2012**

**Bilal Bhakhrani**

**Emily Li**

**Tianyuan Qin**

**Ding Zhao**

**Mengyi Zhu**

Introduction

*Motivation*

There exist many software applications available for people in the neurological fields with three dimensional diagrams of the human brain. There may also be games where a user (player) is able to construct a 3D brain in virtual space. However, it is likely that most of these are commercial products in the market. We are offering a free and open-source application for people wanting such a software, including to the UCLA LONI department.

*Goal*

We want to provide software containing a fun, interactive, and educational puzzle game to help people learn neuroanatomy. Our project, the Brain Puzzle, is an interactive tool that will allow users to study and learn about the different parts of the human brain in an online three-dimensional space. It is a component of the LONI PIPELINE project based on a web application called the Brain Viewer. Our goal is to finalize the working prototype of the game by adding functionalities that allow the user to select a brain piece in order to move it and/or change the piece’s properties for the sake of assembling the complete brain.

*Chosen Solution*

We have continued the development of the LONI project based on the work from previous developers. The languages used are the same as the languages used in the original project: HTML, CSS and JavaScript. We have added functionalities to the existing code while ensuring consistency, uniform formatting, and appearance.

Application Design

*Assumptions and Constraints*

This application was designed with the following assumptions and constraints.

Assumptions:

• All original code from the client is “perfect” as it was before our additions.

• All files associated with each rendered brain piece are in the correct format.

• The files needed for the custom level are provided by the user.

• User-inputted files (custom level) are valid file links that point to the correct file in the correct format as needed.

Constraints:

• User may only use this application through a Mozilla Firefox or Google Chrome web browser with WebGL enabled.

• As this project had a time constraint of one UCLA school term, we may not have had the time to add special features as desired other than the specified requirements.

*Design Components*

To satisfy the requirements, we decomposed the project into the following components. Furthermore, in this section of the report, we emphasize the general design choice of each component. For the details of the implementation, please refer to our design specification document.

Dragging

We decided to use a checkbox to enable dragging. Once the dragging is enabled, the user can move a single piece by clicking on it and dragging the mouse. The user can only move a piece in a 2D plane due to the limitation of the mouse cursor. To move a piece freely in 3D space, the user needs to combine dragging with rotation.

Rotation

Rotation is the default mouse movement, and it is turned off when dragging is enabled. This functionality is already implemented in the Brain Viewer project and is the default setting, so we added the ability to turn it on and off. To reduce complexity, all rotation is relative to the center of the scene. This means that individual brain pieces are not rotatable.

Selecting levels

We allow users to select different levels (easy, hard, custom) before the game starts, with a “show answer” option for each level. In the custom level, the user has to provide URLs for each brain piece in order to play. The program relies on the user to provide valid links to the brain mesh files, and does not perform any security check on the links. Thus, if the user provides invalid or harmful links, the game will not be able to start and may even crash.

Combining pieces

We consider two brain pieces to be “connected” if the distance between them is within the accepted range. This range is the distance between two brain pieces in a completed brain with an additional margin of error. Thus, two pieces can be “connected” even if they are not adjacent to each other. Furthermore, if two pieces are connected, their colors are changed to notify the user. Since the user is unlikely to move a connected piece, we do not restore the piece’s original color if it is detached from a connected group. Every time a piece is moved, the program checks its location against all the other pieces to see if they are combined. However, to reduce the amount of calculation, we only check for a connection when the user ends dragging by releasing the mouse button. Therefore, the user will not be able to tell if a piece is connected while he or she is moving it.

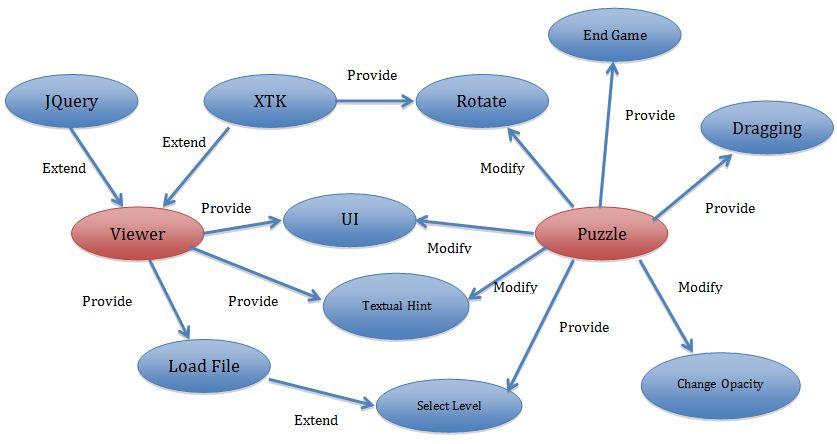
Ending game

The game can be officially ended by either clicking the “end-game” button or combining all the pieces. The program automatically checks if all the pieces are combined whenever the user moves a piece. After the game ends, the calculated score is shown and the user is redirected to the main portal page for the Brain Viewer.

Brain piece property options

Each brain piece has three properties that the user may change. The user can change the color of a selected brain piece, change its opacity or transparency, and/or toggle its visibility. These modifications can only be made while “drag” is enabled. The user needs to click on the desired piece first before making any property changes.

Architecture



**Figure 1: Project UML**

*Overview*

This project is an expansion of the LONI Viewer Project, so it inherits most of the original architecture. The application consists of two major components: a 3D rendering component and a user interface component. The 3D rendering component is supported by the XTK library, which provides an interface to manipulate objects in 3D space. The user interface component includes the display and game interactions such as selecting a level. The two components communicate with each other through multiple events triggered by the user.

Figure 1 shows this project’s UML diagram. It shows the relationships between the existing Brain Viewer project and the Brain Puzzle extension. It displays what functions are added under our puzzle components and how they interact with the viewer components. As a note, the term “modify” is referring to our groups’ modification of the code. The puzzle does not actively modify any code. Rather, the puzzle uses our modifications.

*Tools Used*

XTK library: Handles 3D rendering, such as transformations of each mesh and camera movement.

JavaScript: Used in the main part of the application.

JQuery: Maps HTML components to their event handlers.

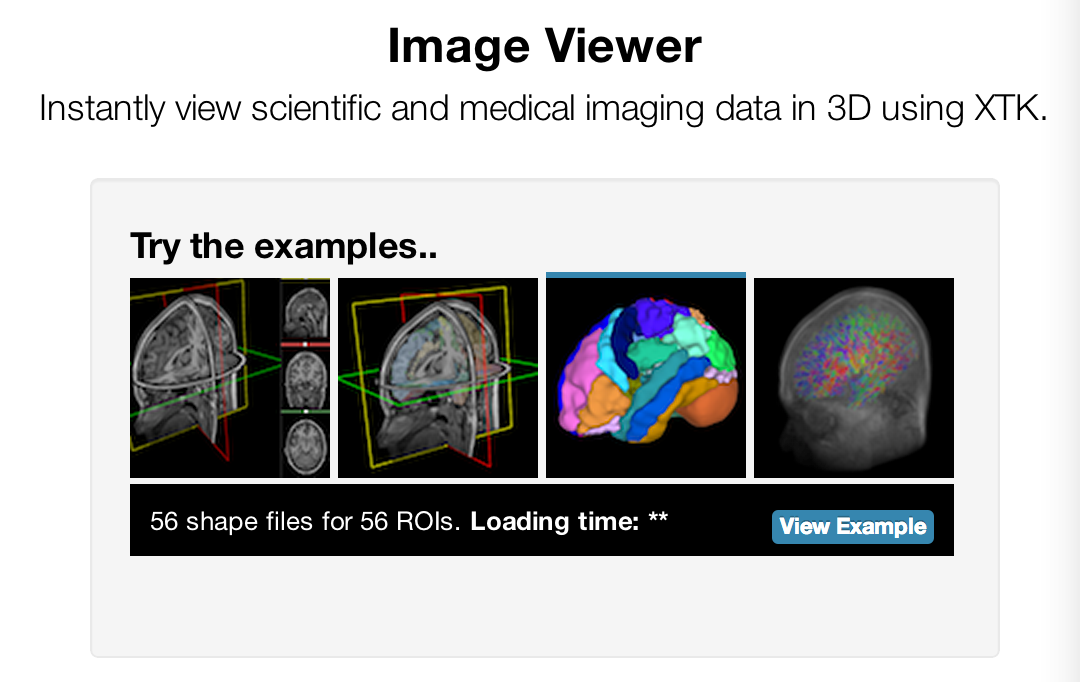
HTML and CSS: User interface design.

Application Use

*Main Portal*

The main portal is a page designed by the original LONI team.

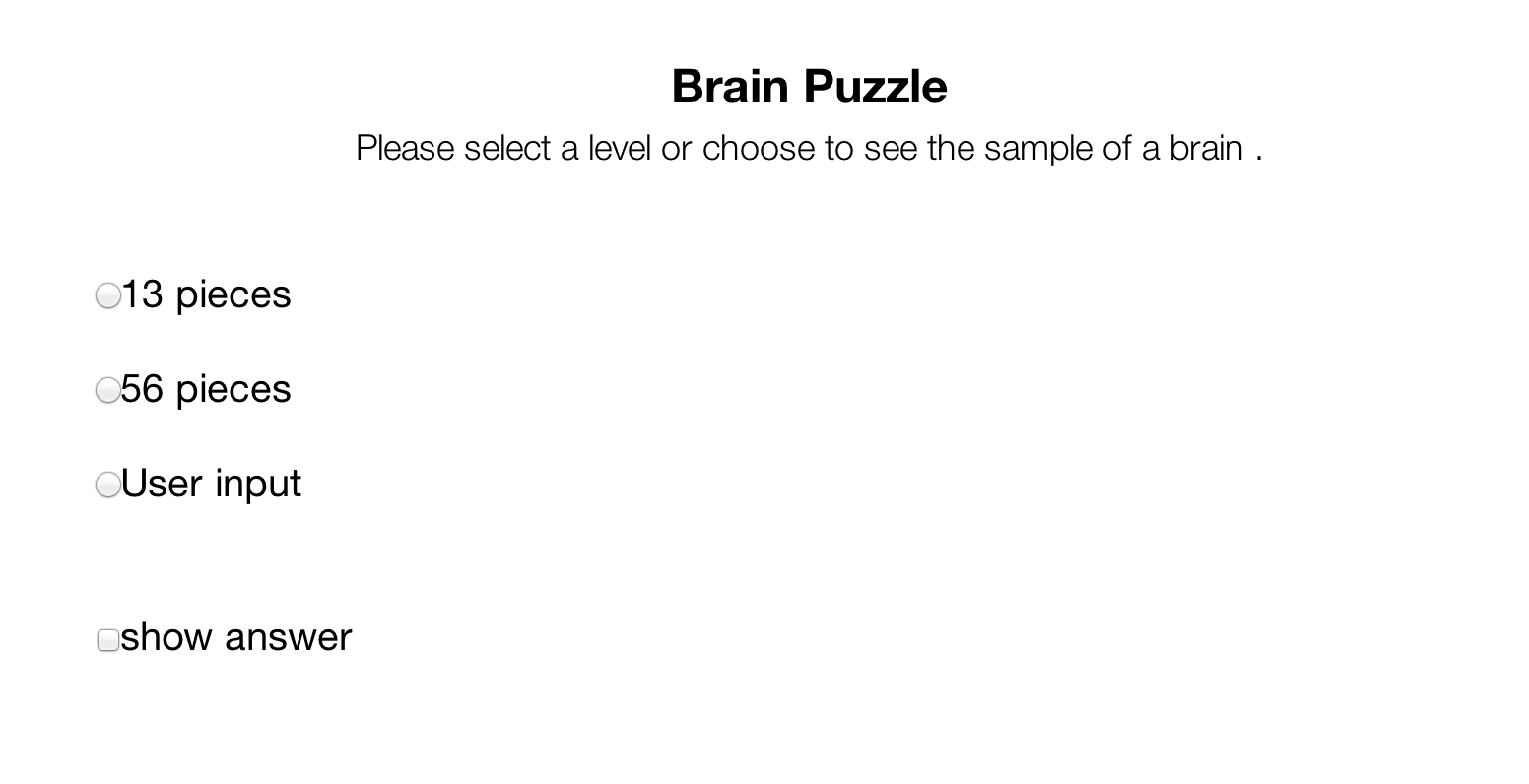
There are four distinct options that the user can choose from. Each option is chosen when the user clicks the corresponding picture. The first loads a single brain volume, the second loads a brain volume and a color map. The fourth loads a volume, a colored label map and a TrackVis file. The third picture-button starts up the Brain Viewer application to run our project, the Brain Puzzle game. (Figure 2)



**Figure 2: Brain Puzzle being selected in the main portal**

*Level Selection Menu*

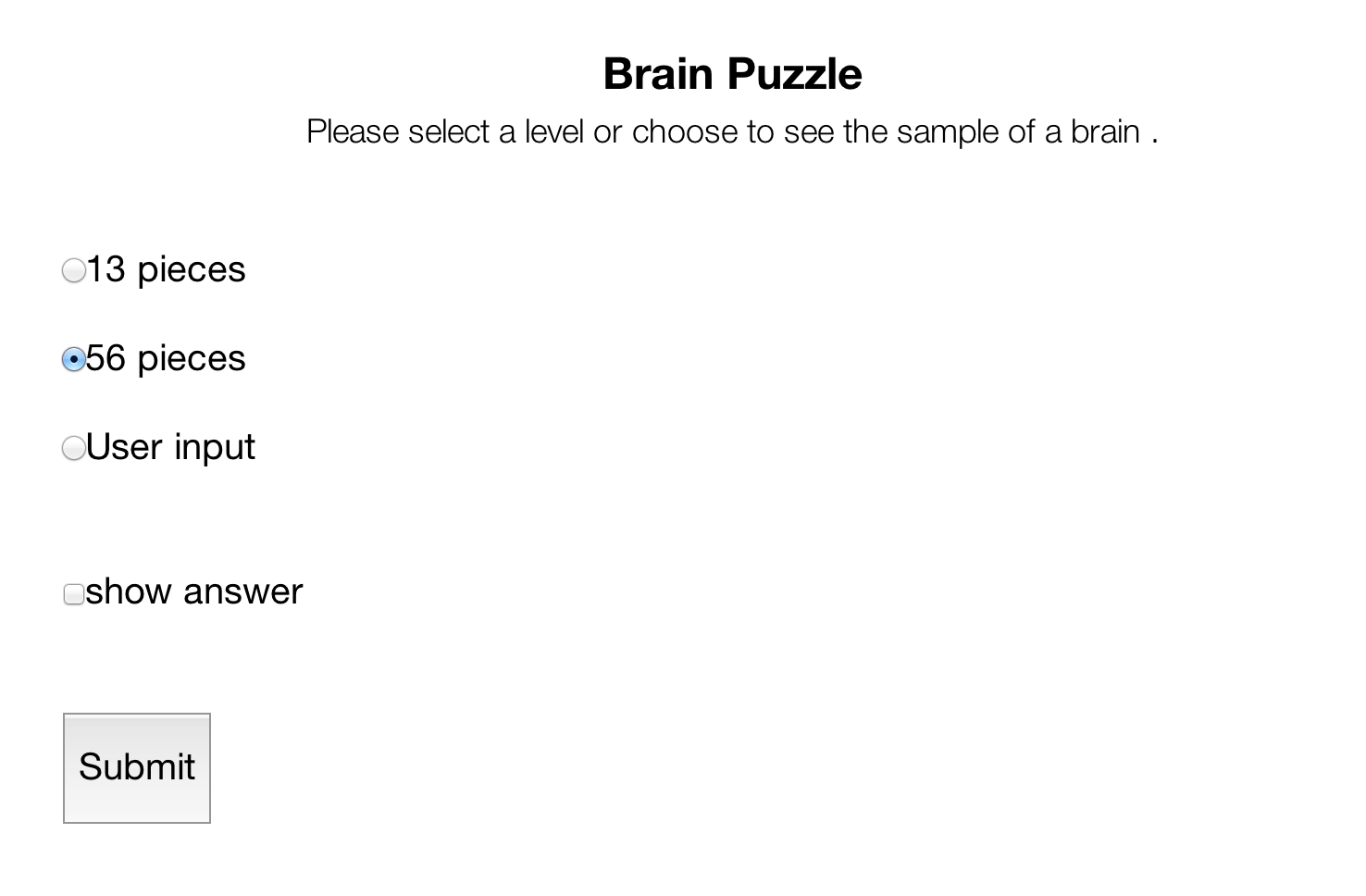
Before the Brain Viewer is launched, we first ask the user to select a level. There are three levels available: easy, hard, and custom. (Figure 3)



**Figure 3: Brain Puzzle level selection menu**

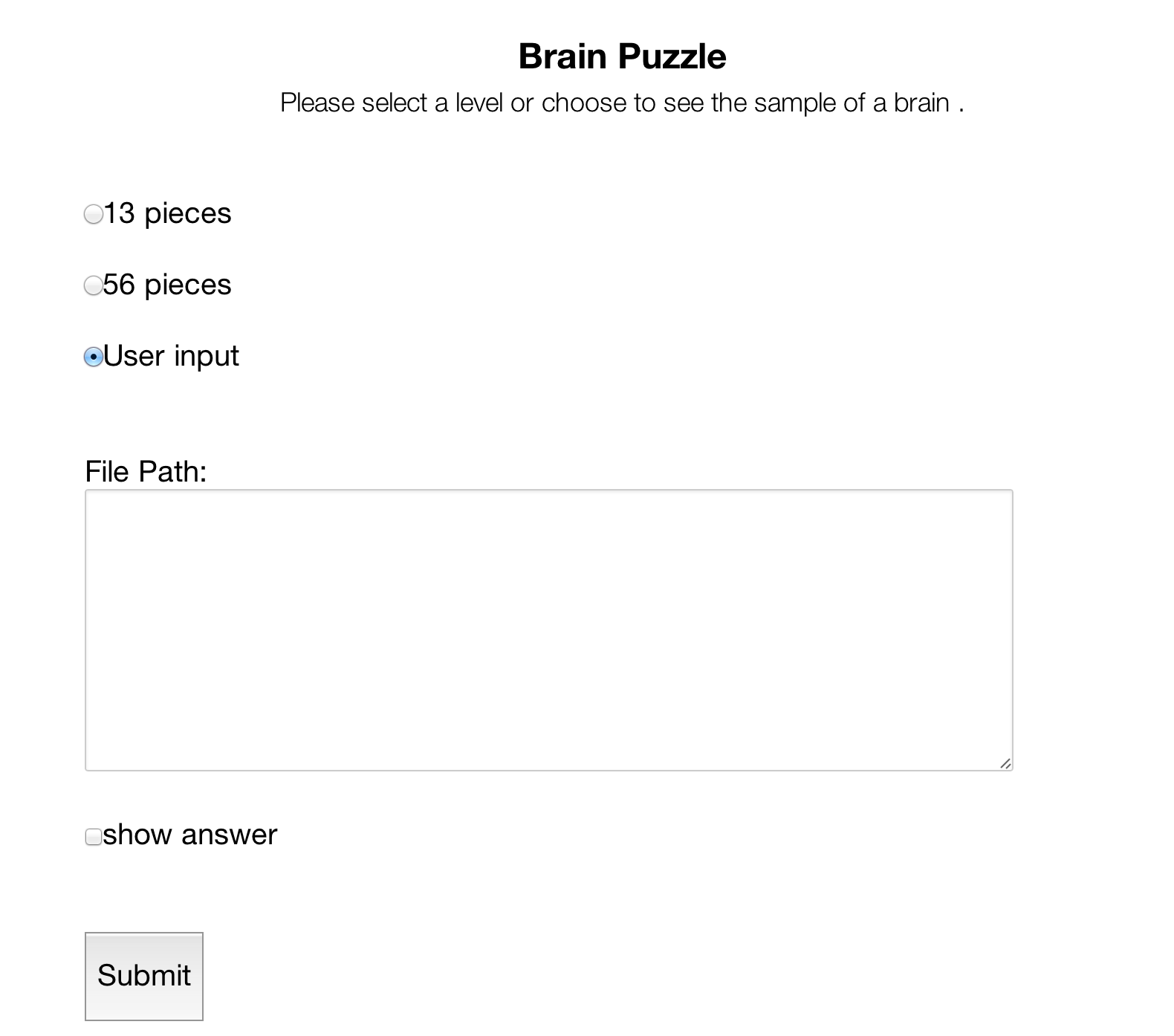
Easy contains a game with the brain split into 13 pieces, while hard has 56 pieces. The custom level is where the user creates their own game by supplying the desired brain pieces. For example, if the user would like a game with 30 pieces, then the user must input the file links for each of the 30 pieces.

There is also an option box “show answer.” The user may check this box if they would like to see the completed puzzle for the chosen level. Once a level is selected, the submit button becomes available. (Figure 4)



**Figure 4: Level selection menu when hard is selected**

If the custom level is chosen, then a text box appears, allowing the user to input the links to their brain piece files. The links must be inputted one file per line. (Figure 5)



**Figure 5: Level selection menu when custom is selected**

*Brain Viewer*

This is the actual game page. It consists of a black 3D virtual space with multicolored brain pieces scattered about the space. There is a back button at the top left (see Figure 6) that leads to the main portal. The user may click on this button to end the game without calculating any scores. There is also a side menu on the left, under the back button.

Options Side-Menu

The side menu has the available options:

- A show/hide icon to toggle the visibility of a brain piece.

- A color icon to change the color of a brain piece.

- A pin icon that turns auto-hide on and off for the the side menu.

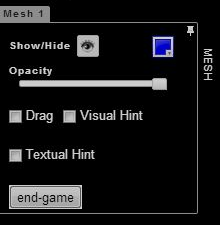
- An opacity slider to change the opacity of a brain piece.

- A drag option box to change mouse movements to dragging.

- A visible hint option box to show this hint.

- A textual hint option box to show this hint.

- An “end-game” button to prematurely end the game.

**Figure 6: Back button Figure 7: Options side-menu (the Mesh)**

The show/hide icon can be clicked to toggle the visibility of a selected brain piece. (Figure 7) The user may select any of the brain pieces. The selection is done by clicking on a brain piece while the drag option is checked.

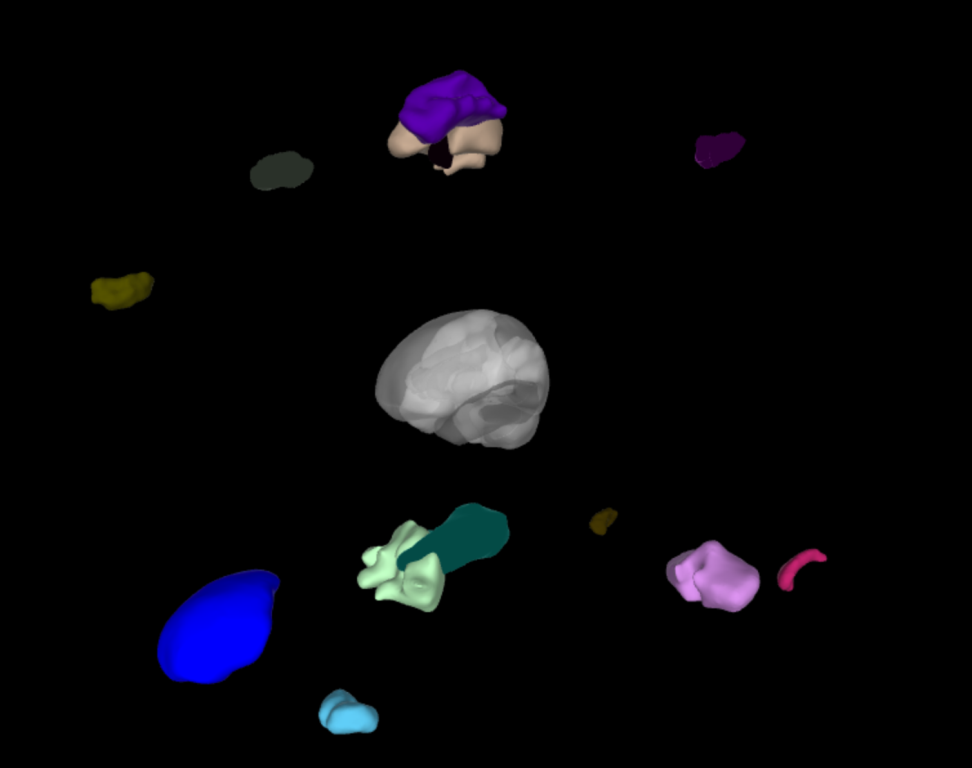
The color icon can be clicked to show a color range. (Figure 7) The user can then choose a color on this range. The selected brain piece will then change to this color. However, like mentioned above, piece selection must be done while the drag option box is checked.

The pin icon in the corner can be clicked to turn on auto-hide for the side menu. When the pin is clicked on, it rotates to the left side and effectively pins the menu to the space beyond the left web browser wall. The side menu is then hidden unless the user places the mouse over this area.

The opacity slider allows the user to change the opacity of a selected brain piece.(Figure 7) Sliding the slider to the right will increase the opacity, while sliding it to the left will decrease the opacity. Again, the piece must be selected when the drag option is selected.

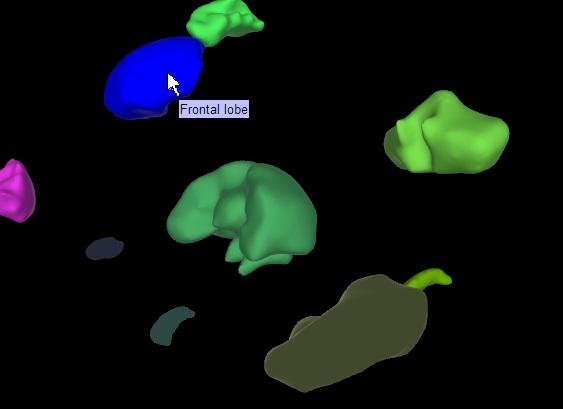
The drag option box is one of the most important features of the game. (Figure 7) Selecting this option will change the behavior of the mouse movements so that the user may hold the mouse down on a brain piece and move it around the screen. However, as the screen and mouse coordinates are based on the 2D screen, the user may only move the piece along the current 2D plane. Please see “Game Play” below for more on how to move pieces.

The option box for the visual hint is to turn the visual hint for the game “on” when selected. (Figure 7) The visual hint is a completed model of the assembled brain shown in the middle of the space. The model is gray and transparent as to differentiate from the game’s brain pieces. See Figure 8.



**Figure 8: Visual hint turned “on”**

The option box for the textual hint is to turn the textual hints for the game “on” when selected. The textual hints are labels for each of the brain pieces. When the user turns these hints on and places the mouse cursor over a brain piece, that piece’s name is displayed. See Figure 9.



**Figure 9: Textual hints turned “on”**

The “end-game” button allows the user to end the game prematurely. (Figure 7) For example, if the user is only half-way done with the game, but would like to stop playing, the user may click on this button to end the game. Please see “Game Play” below for more on scoring and the end of the game.

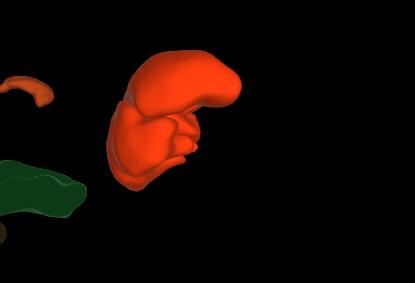
Game Play

*Movement*

The game is slightly inconvenient to the user because the user is moving 3D objects in 3D space in a 2D window screen. As such, moving pieces must be done in vectors along the plane the user’s screen shows. This means that moving the pieces to assemble the puzzle is roughly done by alternating between dragging a piece and then rotating the view to check its location.

Rotation is the default mouse movement. It is done by clicking on the screen and dragging the mouse. The view of the screen camera is then shifted according to the direction the mouse movement indicated.

Combining pieces is done by dragging a brain piece close to one of its neighbors and rotating the scene to check if the pieces are close together. If they are not close enough, the user can drag the piece closer then rotate again to check. This is the alternating movement of game play. Once two pieces are combined, the two pieces’ colors change to match each other. (Figure 10)



**Figure 10: A few brain pieces combined together**

*End of the Game*

There are two ways to end the game. One is to completely assemble the brain and the other is to click on the “end-game” button. Once either method is done, the score of the game is calculated and displayed on a pop-up window. (Figure 11)



**Figure 11: Popup at the end of the game**

Once the user clicks “OK” on the pop-up window, the web page refreshes and the user is taken back to the menu portal.

*Scoring*

The scoring is done by taking the average of two sources of points: time and brain completion. Each source is worth a total of 100 points.

The time portion is calculated by the duration of the game. No penalty is given if the user ends the game within 30 minutes. If the duration of the game is between 30 minutes and two hours, the program deducts points gradually. If the duration is over two hours, the time score is zero.

The brain completion portion is calculated by the equation (n/N) \* 100, where n is the number of brain pieces not connected to any other piece and N is the total number of brain pieces in the game. In addition to this, one point is subtracted for every five seconds hints are on. This can be either one type of hint or both hints on.

Challenges

*Technical Challenges*

Environment setup: Apache server and LONI project

We were required to set up a local Apache server in order to run the project on our own machines. Also, the LONI project only runs on specific browsers. However, there was some trouble here based on the operating system on the machine.

*Mac OS X*

The Apache server can be turned on with the GUI option in System Preferences > Sharing to turn on Web Sharing. However, the new OS X 10.8 dropped this option. We later figured out that Apache is pre-installed in the OS and only needs to be enabled via the command line. We also had to modify some configuration files and create a Site folder in order to have Apache running appropriately.

*Ubuntu/Linux*

Although the Apache server can be installed and set up correctly in the Linux environment, we found that Ubuntu’s version of Mozilla Firefox does not have some component needed for the LONI project. We believe that the problem is that Ubuntu’s version does not have WebGL.

Movement: Transformation matrix and camera view

One of the most challenging parts in this project was not in the actual code, but the math and physics involved in computer graphics. Since we are looking at a 3D space through a 2D computer screen, we have to map a 2D position on the screen to a 3D position in the scene. This involves understanding how camera view matrix and transformation matrix work in 3D rendering, which requires strong math skills.

Understanding the code

Since we were building on an existing application, we had to first understand a good portion of the thousands of lines of existing code before we could begin implementing any of the features. Thankfully, the XTK library had documentation, making the task much easier, but nonetheless, challenging.

Event Handling

The event handling in this project plays a large role in every additional feature we were adding. For example, rotation of the scene and dragging of a brain piece rely on mouse events. The options were based on user input selection on a form consisting of radio buttons, checkboxes, and buttons. Thus we had to define a space for all event handling code.

*Team Challenges*

In addition to the technical challenges, our team had some trouble as a group. Because we all had different schedules, it was sometimes difficult to set meeting times. Due to this issue, it was quite tough to keep track of who was working on which task. Thus, making sure work was allocated evenly was somewhat problematic.

Conclusion

*Final Thoughts*

This project was a great learning experience which required learning a lot of useful tools and skills. We were able to learn web programming, a useful skill to have in the workforce. Additionally, for most of us, this was our first time creating a game with graphic rendering. It made us realize some of the complexity that goes into game development. The most interesting experience was probably the documentation, something most of us had not done before. We never realized how much effort was actually needed for that portion of software engineering. Overall, it was a great experience that gave us a good understanding of how to work as a team and how to complete a software project in its entirety.

*Future Goals*

Our project contains the basics of this brain puzzle game. Here are a few features that would be either beneficial or nice for consideration in the future:

* Checking the validity of links given for the custom level.  
  In our current application, we only check the input given by the user contains generic links (“http”). If the link given is not a valid brain file but passes the generic link check, our application currently has undefined behavior.
* Improve user manipulation of brain piece position.  
  The current method of moving pieces is quite cumbersome for the user, since the space is 3D but the user can only move in certain viewer planes. Perhaps some improvement in the future can be made to lessen this strain.
* Adding sound effects when:
  + Loading game
  + Pieces are combined
  + Game is finished
* Pause button.  
  The game flow in the current application is constant, meaning that the game allows no interruption between the start of the game and the end of it. Therefore, if the user, for some reason, needs to leave the game during game play, the time duration of his or her absence is included in the final score. A pause button would help this problem.
* Saving the game.  
  This is similar to the pause button, except on a higher level. For example, the user must close the web browser but would like to continue his or her current game. This is on a higher level than a pause button because it would involve saving game statistics and data, possibly on the server. It would also include knowing how to track whose game was saved.
* Score tracker  
  This extra feature, like in saving the game, would require ways of differentiating users and saving data to some database. The score tracker may be keeping track of one user’s top scores or tracks the total high scores for all users.

*Acknowledgements*

We would like to thank the LONI department, especially Zhizhong Liu and Alen Zamanyan, for allowing us to work on this project, and giving us the necessary resources and support we needed. Furthermore, the documentation of the XTK library played a key role in our ability to complete this project and understand the code to the level we did.

*References*

Source code: <https://github.com/andy0727/LONL2>

XTK Toolkit API: [http://api.goXTK.com](http://api.goxtk.com)

LONI Team: <http://www.loni.ucla.edu/twiki/bin/view/LONI/Pipeline_2012_StudentProjects>

Slice Drop: <http://slicedrop.com>