

# **Exploring Earth and the Solar System:**

Educational Outreach Through NASA's

Space Place, SciJinks, and Climate Kids Websites

JPL Summer 2012

Joseph Christopher Meneses

East Los Angeles College

Mentor: Austin Fitzpatrick – 172F

NASA's Space Place team publishes engaging content and creates an effective environment to inspire a young audience to dare mighty things. NASA uses the Space Place, Climate Kids, and SciJinks websites to cultivate interest among elementary-school-aged children in both science and technology. During my summer internship at Jet Propulsion Laboratory I used Adobe Flash and ActionScript 3 to develop content for the Space Place, Climate Kids, and SciJinks sites. In addition, I was involved in the development process for ongoing and new projects during my internship. My involvement allowed me to follow a project from concept to design, implementation, and release. I personally worked on three projects this summer, two of which are currently in deployment. The first is a scrambled letter-tile guessing game titled Solar System Scramble. The second, Butterfrog Mix-Up, is a rotating-tile puzzle game. The third project is a unfinished prototype for a maze game.

When I started my internship, my team already had a catalog of products available on the Space Place, Climate Kids, and SciJinks websites. The team was also in the process of deploying an important iOS product, thus it was important that I was able to easily fit in as a contributing member of the team, and I was able to do so by working on a few projects that were backlogged. The projects that I would complete had two important objectives: First, expand the wealth of products available on the Space Place, Climate Kids and SciJinks websites, creating more tools to educate their users. The second objective was to lay groundwork for future applications that would use similar mechanics, graphics, or code. Coming into this internship I had experience working in Java, which proved to be a great benefit as ActionScript 3, the primary work-language this summer, shares a number of similarities with Java. The most important similarity I found between the two languages was class implementation – an important facet which allowed me to understand the relation between Flash assets and the code associated with them. Adobe Flash is interesting in that it simultaneously appeals to both animators and programmers, with the code bridging the gap.

The first project that I worked on is entitled Solar System Scramble. SSS was a collaborative effort between my fellow intern Fernando Jaime and me, and is a scrambled letter tile guessing game. The game contains 18 hidden words which the user must guess by unscrambling the letter tiles given. The user is also presented with a clue to help guess the answer, with up to three clues available for the user, at a cost of a reduction of the score for the current question given upon correct guess. The color of the letter tiles are white by default, but would change to green on a correct answer, and red otherwise. Whether the user correctly guessed the hidden word, or otherwise pushed the "Give Up" button, an interesting fact about the subject of the question would be presented in a popup window. On this particular project, I was in charge of the game mechanics including movement of the letter tiles by mouse and keyboard input, as well as answer validation and the fact popup.

I encountered many obstacles in completing Solar System Scramble. The primary obstacles were learning the nuances of ActionScript 3, and understanding the relationship between graphical assets and how the code handled them. Unfortunately, the easiest and fastest way to get a handle on the language was to delve right in and make mistakes. Without much of any sort of pre-program planning, design, or even pseudocode, the source code for SSS turned into an incredible mess extremely quickly. As one roadblock after another presented itself due to sloppy code, my project eventually reached critical mass and it became clear that great swaths of code would need to be destroyed, condensed, or rewritten. The two weeks it took to reach this

point was not time wasted however, as my time spent “Cowboy Coding” without clear direction accomplished two major objectives: Primarily, by getting right to code I was able to get very close with the code and understand the finer points I would not have been able to understand without experience. Second and most importantly, although I would have to do a major rewrite of the code I understood the goals of the project and further, I understood the *wrong* way to go about implementing certain functionality.

The important revisions of the code had to deal with the mechanics of the letter tiles and the answer spaces, recognition between them, movement, and keyboard and mouse functionality. Drag and drop capability was the first feature to be implemented. The problem was that on a mouse up event the letter tile in hand was not properly snapping into place in the answer space. The tile would drop, but would snap into an invisible space at the edge of the screen. The problem was that the tile snap-to function worked correctly, but its functionality was relative to a different coordinate plane than the one I intended. This had to do with the children of different objects having coordinate planes relative to the parents, and as the answer tiles are children of the answer space, and the letter tiles are similarly children of the letter tray, the actual and intended coordinates were two entirely different things. The solution to this problem was to identify the coordinates of the tiles and answer spaces relative to the global coordinate plane. At this point the overlap of the two could be ascertained and the snap-to functionality worked as intended. The solution to this bug laid important groundwork to better understand the relationship between objects leading into the second project.

The second project, Butterfrog Mix-Up, is a rotating tile puzzle game. The game features three puzzles of increasing difficulty including a four piece triangular puzzle, a nine piece square puzzle, and a fourteen piece hexagonal puzzle. Each puzzle piece can be rotated by a click and drag gesture with the mouse, or clicked for an automatic rotation to the next possible position. At the start of each puzzle, the player is briefly shown the correct configuration before the puzzle is scrambled and the timer starts. A side bar shows the time spent and total rotations made thus far, holds the music and sound mute buttons, and contains a hint and quit buttons. When pressed, the hint button will highlight all the pieces of the puzzle with a 5 rotation penalty for use: If a piece is in the correct position the piece will be highlighted green and highlighted red otherwise.

Butterfrog Mix-up presented far fewer problems than the first project, and in fact the greatest obstacles had almost nothing to do with project design. It was important to take what was learned from the problems of the first project and ensure that the same obstacles in design were not encountered with the second. Before any code was written, I spent two days taking pen to paper drawing diagrams, writing pseudocode, and reading documentation of needed objects. I was able to effectively solve many problems before they were even encountered. The primary benefit of planning ahead of programming was the abstraction of puzzles and puzzle pieces which provided a clean framework for each type of puzzle to be created, and also allowed for an easy-to-read and maintain codebase. By clearly defining shared characteristics between puzzles, and shared behavior of pieces, a well written abstract class contained all the code necessary to deal with everything from the rotation of a puzzle piece, to correct position validation. In fact, the largest obstacle encountered was not through fault of design, but rather the way that Flash rotates objects. Rotation of an object is a degree rotation of +/- 180 degrees from the origin, so an interpreter from this to a 0 – 360 degree rotation map was written for ease of use. The second problem with rotation was the transition from 360 degrees to 0 degrees of rotation. Practically,

360 and 0 are the same, however in creating the rotation animation Flash insisted on forcing a full 360-degree rotation of the object from 360 downward to 0. It actually turned out that this particular case was already coded into the rotation class, and all one simply had to do was add the degree rotation and Flash would do an automatic conversion from 360 to 0. As long as the current rotation of the object was updated after the conversion was done, the object would proceed to rotate correctly for each position thereafter.

Though Solar System Scramble and Butterfrog Mix-Up are in the deployment stage, the third project I worked on this summer is still an early prototype. My last project is to create a maze game which can be expanded to any of the Space Place, Climate Kids or SciJinks websites by simply changing artwork. The Maze Game's challenges included designing a way for the maze to be created, stored, and generated. The first breakthrough was in creating a wall signature pattern for each cell to create walls from. The wall signature is a four-digit string where each digit is either a 0 or 1 corresponding to a wall in a cardinal direction: West, South, East, North. In the sample signature 1001, a wall exists at the west and north directions, where the south and east directions are both clear. This signature is useful for multiple reasons: First, the strings are easily stored in a small text file, are spaced out by tabs for columns, and each row is a new line. Second, Flash would be able to procedurally generate a small wall graphical object based on the wall signature of each cell, snapping the cells together like a quilt upon creation. Third, the Python script that generates the signature file is small and easy to maintain. Finally, the simple nature of the wall signature would allow for easy bitwise collision detection for the player token. In short, when the player presses a directional key, the event would trigger a check to the corresponding bit in the wall signature which would send a Boolean value whether that direction was clear or not. This solution to collision detection was far cleaner than the graphic-based object collision detection I first had in mind. Since the groundwork for maze generation based on a file containing the signatures for each cell has been developed, the next step for this project would be to develop an algorithm that would reliably create a maze with a single path through.

As my internship comes to a close I reflect that I have learned much by working on my projects. The most obvious lesson this summer is that I walk away having learned another programming language. Yet leaning ActionScript 3 was not nearly as significant as learning how to better approach project design to avoid and overcome obstacles. I also learned a great deal about properly implementing object-oriented and polymorphic design patterns to produce clean and effective code. Solar System Scramble and Butterfrog Mix-Up will soon be deployed externally and will provide two more tools for NASA to do educational outreach in a fun and inspiring manner. As I leave I also take pride that I documented my code well, particularly for the prototype maze game so that whoever follows my work can easily pick up where I left off. The skills I learned as part of this internship will allow me to be an effective team member in future projects.

*This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Minority Student Program and the National Aeronautics and Space Administration.*