Jeff Bender | UNI: jrb2211 | 5/8/2015 | COMS E6123 Final Project Final Report

Tooling Scratch
Developing a Collaborative Game-Based Learning System to
Infuse Computational Thinking within Grade 6-8 Curricula

1. INTRODUCTION

The final project utilizes the design tactics identified in the midterm paper to begin tooling Scratch 2.0 with SAGE, a collaborative game-based learning (GBL) system which infuses computational thinking (CT) within grade 6-8 curricula. Given the short timeframe of the final project, the deliverable, SAGE 0.1, is far from a complete system, but it does represent a preliminary step forward. The functionality implemented includes initial SAGE Design and Play modes, and palette and block restriction systems that afford teachers, as game designers, the capability to scaffold CT learning outcomes. By actualizing one of the gameful design tactics highlighted in the midterm paper, this feature set supports the situated meaning learning principle and begins to enable the creation of goal-based scenarios and story-centered curricula.

2. FEATURES

Three major features were proposed and implemented: SAGE Design and Play modes, a palette restriction system, and a block restriction system.

1. *SAGE Design and Play modes*. To enable both game-design and game-play, SAGE requires at least two modes. Although in follow-up versions, the Design mode likely will require a testing sub-mode, as well as a security mechanism, so that teachers, who might prefer that students do not alter designs, can enforce focus on game-play, this project does not address securing SAGE modes or testing. The play mode primarily serves as a placeholder for future work and does not yet provide substantive SAGE functionality. Any palettes and blocks restricted, however, are unavailable for use when in Play mode.
2. *Palette restriction system*. The palette restriction system affords game-designers the capability to scaffold learning by constraining the palettes available for use by students during particular games (and in future work, levels). When CT game mechanics embed endogenously in forthcoming versions of SAGE within goal-based scenarios, this restriction system will help create narrative structure, and thereby support story-centered curricula.
3. *Block restriction system*. The block restriction system aligns in aim with the palette restriction system, but facilitates more granular scaffolding. This project addresses restricting blocks from use in particular Scratch projects. Eventually, game-designers using future iterations of this feature will be able to specify game events that "unlock" particular blocks, and thereby guide students through finely-tuned CT learning progressions.

3. USER COMMUNITY & VALUE

The primary user community for SAGE is grade 6-8 teachers and students in formal learning environments, i.e., schools. SAGE aims to help STEM and language arts teachers infuse CT within existing curricula. The secondary user community is youth in informal learning environments, such as after-school technology centers. Typically, teachers will use Design mode, and students, Play mode. Especially in informal learning environments, however, older children might use Design mode to create games for their younger peers. As described in the midterm paper, SAGE offers the value of a collaborative game-based learning system which enables teachers and students to immerse and engage with CT concepts, practices, and perspectives.

4. RESPONSE TO FEEDBACK

Feedback to the proposal, progress report, and midterm paper offered several helpful ideas. The recommendations included: think about practicality for teachers who have zero computer science training, as well as for young students; discover if an organization has developed a CT curriculum for 6-8, and determine if SAGE can include a corresponding implementation as a suggested sample; incorporate a play/test/debug sub-mode for Design mode, using facilities from Play mode, but allowing test cases, debugging, and the equivalent of "unit testing" and "testing coverage"; produce complete sample projects that teachers can show in class with accompanying step-by-step development guides, possibly modeled on teacher guides for regular textbooks used in 6-8; determine if Scratch runs on tablets; explain the envisioned use of role-play, and figure out what the magic system will do and how. As noted alongside this feedback, many comments apply to the long-term research agenda for SAGE and not this course project. I address a subset here and suggest plans for addressing the remaining items in future work.

Teachers with limited CT content knowledge, and typically no CT pedagogical knowledge, require substantial support from SAGE. In future versions, the palette and block restriction systems can include a learning-progression-grounded series of easily-selectable presets that fit particular CT concepts and link to relevant embedded instructional material that helps teachers introduce the associated CT practices and perspectives to young children. Additionally, SAGE can include a set of example templates and games that align with the CT curriculum suggested on page 16 of the CSTA Standards Task Force's *K–12 Computer Science Standards*, and map each CT concept, practice, and perspective to Common Core standards by using CSTA's guide, *CSTA K-12 Computer Science Standards: Mapped to Common Core State Standards*. As discussed in the midterm paper, the National Research Council has advised that school administrators and teachers incorporate CT into established school subjects, rather than attempt to carve new time-slices from overloaded school days. The example games, therefore, should likely blend content from existing subjects with the curriculum developed by the CSTA Standards Task Force. A set of step-by-step development guides, modeled on teacher guides for 6-8 teachers, as well as demonstration videos, can accompany the example curriculum implementation. Ideally, in the long-term, a collaborative community will contribute curricular games across subjects for easy adoption and adaptation.

I don't have a clear vision yet for adding a testing sub-mode to Design mode and look forward to further discussions on the topic, especially considering the instructor's related expertise. I imagine teachers could react abrasively to the notion that not only might they need to design or adapt games, but also test them, and thus we might benefit from reframing the testing sub-mode as a form of "pre-play" or "beta" game. Since Scratch eliminates syntax errors, a sizable proportion of testing might fit the paradigm of "game balancing." The development of both automated and collaborative community tools to assist with game balancing might lead to interesting veins of CS research worthy of pursuit.

Unfortunately, Scratch does not run on most tablets, because most tablet operating systems do not support Flash. ScratchJr, a recently-released iPad app, is a version of Scratch designed for K-2 students, but its source code is not open-source. I'm in agreement that SAGE should support tablets at some point. For the near-to-interim term, though, as concluded in the midterm paper, I believe Scratch presents the best platform in which to begin implementing the design tactics identified this semester. The associated realized designs likely will require years of iteration, testing, and revision, and by the time we work through refinements, the technological landscape might offer alternative host platforms for SAGE. From an aspirational perspective, SAGE eventually could become a semi-portable GBL system, or at least depend more on its internal facilities than those of any host platform.

Lastly, agreed, it's time to produce more detail about what SAGE will actually do and how. The role-play envisioned does not relate to the teacher and student roles, but rather to roles that teachers can provision within their games in Design mode, and students can take-on in Play mode. This summer, I plan to transform the role-provisioning design tactic, as well as nine others identified this semester, into proposals for designs implementable within Scratch. I'm hopeful these proposals will begin to transition the perception of SAGE from an undefined magic system into an actualized learning tool grounded in theory and useful in practice.

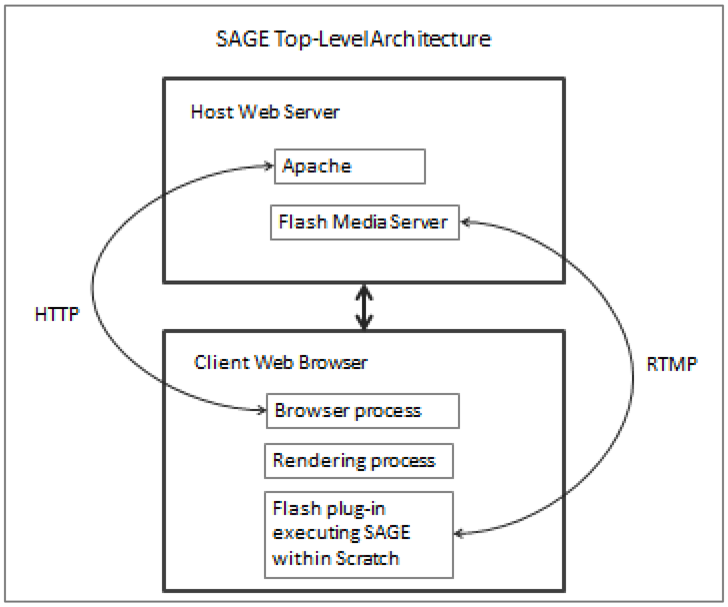


Figure 1

5. ARCHITECTURAL ORGANIZATION

The top-level architecture, depicted in Figure 1, conforms to standard organization for a website containing a Flash object. Standard web communication occurs between the client browser and host web server via HTTP. I'm inexpert in Flash communication flows, but my understanding is that the Flash client communicates with the Flash Media Server using Action Message Format (AMF), a binary format for serializing ActionScript objects and XML. It establishes connections to the server for chunked media streaming using the Real Time Messaging Protocol (RTMP).

6. IMPLEMENTATION

6.1. SAGE Design & Play Modes

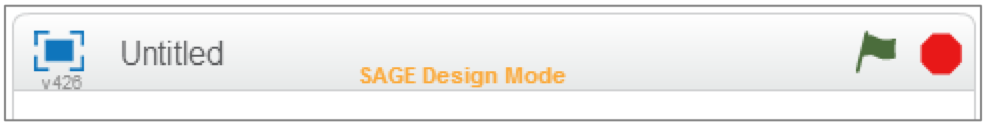


Figure 2

As earlier noted, SAGE 0.1 contains two modes, Design and Play, which present educators and learners, respectively, with distinct capabilities. Toggled on and off from the Edit menu, Design mode enables the configuration of the restriction systems. Play mode, similarly accessed from the Edit menu, does not allow restriction configuration, but does enforce the restrictions specified in Design mode. The selected mode is displayed in text at the top of the stage, as shown in Figure 2. These menu items, depicted in Figure 3, are added in Scratch.as, and their selection alters the state of Boolean properties in Interpreter.as.

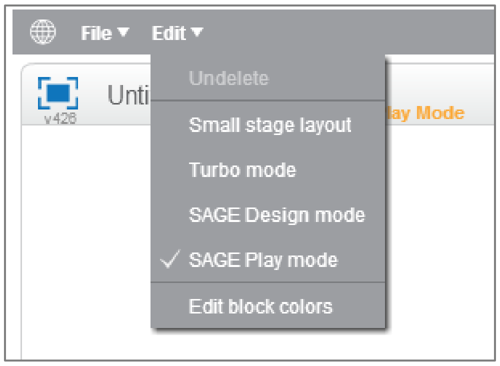


Figure 3

In upcoming refactoring efforts, these Booleans likely will be replaced by a single enumerated mode property in order to introduce additional flexibility. Maintaining the mode in the interpreter class makes sense because forthcoming versions of SAGE will require differentiated interpreter operation depending upon mode. The suggested Testing sub-mode of Design mode, for example, certainly will impact the operation of the interpreter. Furthermore, the mode is a property of the application, and not the active project, and thus does not require maintenance in a class that persists to disk with other project data upon saving. Currently, the Interpreter class is merely a convenient location, since a reference to the interpreter exists in class Scratch, and a reference to the active Scratch instance exists in many classes modified to support the restriction systems.

6.2. Palette Restriction System

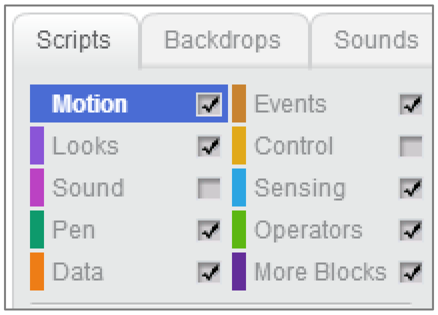


Figure 4

The palette restriction system, configured by the palette selector checkboxes displayed when in Design mode, as shown in Figure 4, enables the educator to preclude the use of particular palettes from the active project when in Play mode. When a palette selector checkbox is unselected in Design mode, the blocks in the associated block palette adjust in color and behavior, as show in Figure 5. The shift in color provides the user with confirmatory feedback that the restriction is in effect; additionally, the context menu for each palette-restricted block no longer allows configuration of the block restriction system, which is discussed in detail in the following subsection. Blocks in a restricted palette do remain available for use when in Design mode, since the educator might require such blocks to develop her game. When in Play mode, however, the user cannot access restricted palettes, which are colored grey as shown in Figure 6, nor duplicate associated restricted blocks in the scripts pane (i.e., the context-menu is disabled). To remind the user of the difference between the capabilities of a restricted block in Design versus Play mode, restricted blocks are filled with a different restricted color in Play mode, also shown in Figure 6. When the application is not in Design or Play mode, the palette selector and all blocks revert to their original Scratch look and behavior, as depicted in Figure 7.

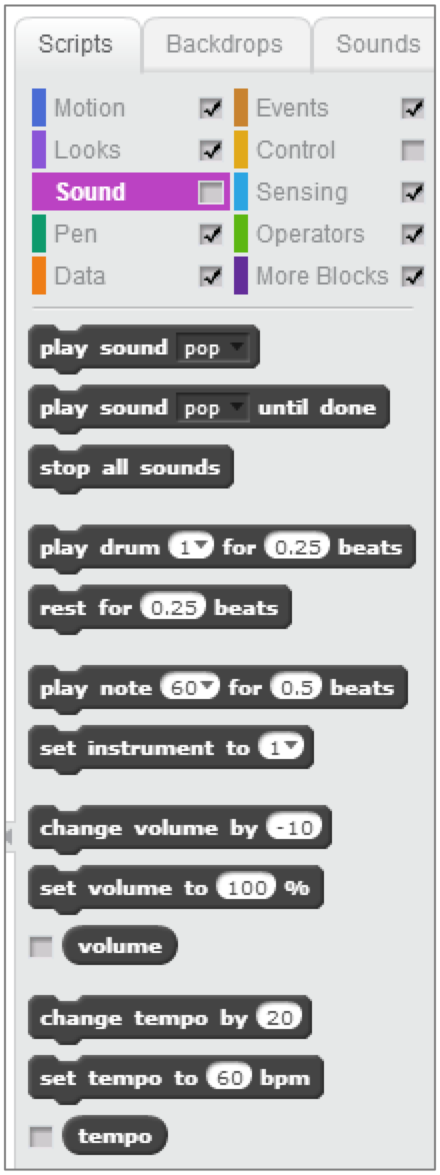


Figure 5

Currently, palette restrictions operate as a super-system, meaning that a restricted palette prevents the Play mode use of any block in the palette, regardless of possible block restrictions additionally configured. A palette, therefore, is either unrestricted with possible block restrictions, or restricted. This design choice helps users distinguish between the two restriction systems and simplifies the implementation, but might benefit from additional consideration in future versions. The initially-proposed design included a "mixed" state for the palette restriction system that manifested in the interface as a filled palette selector checkbox, as depicted in Figure 8, to indicate

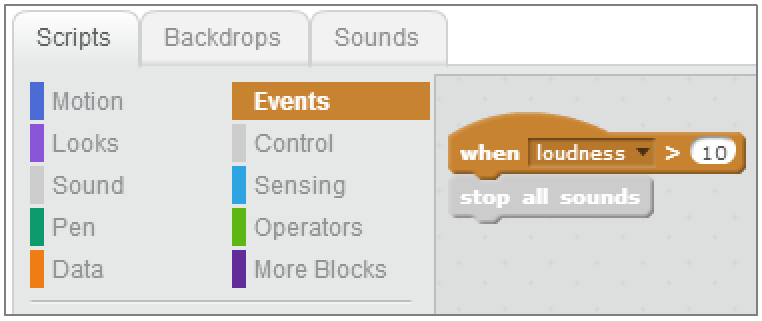


Figure 6

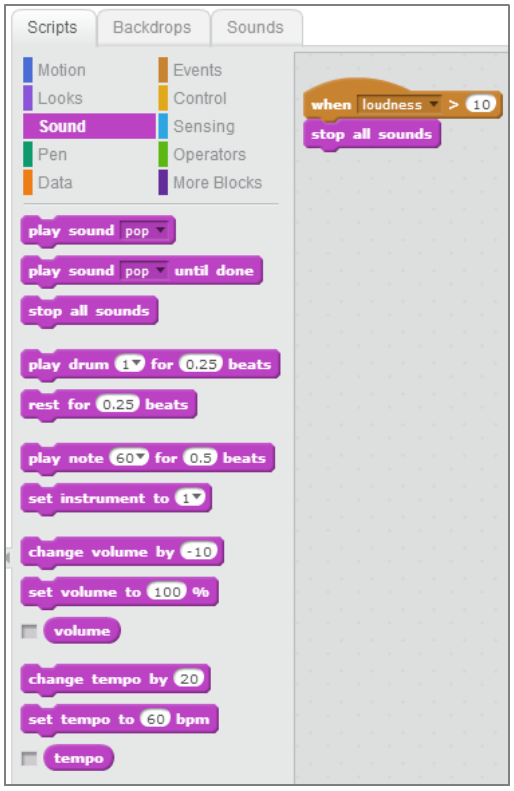


Figure 7

that the palette is not restricted, but some of its blocks are. While such a design might enhance usability, since the palette selector checkboxes could effectively serve as "select all" and "select none" from a block-restriction perspective, it too confusingly blends the restriction systems at this stage of development. Depending on forthcoming design decisions related to applying the palette and block restriction systems to game levels and events, the two systems might offer the most flexibility by remaining fully separate. In that case, "select all" and "select none" could become additional features of the block restriction system, devoid of any coupling to the palette restriction system.

The primary classes modified to implement the palette restriction system are PaletteSelector and PaletteSelectorItem. PaletteSelector now contains an array of Booleans that align with the pre-existing array of palette descriptions. When a palette restriction checkbox is unchecked, the associated PaletteSelectorItem raises an event which sets the associated PaletteSelector Boolean to false, and the blocks in the associated palette are updated to display the restricted color and behavior, via a function in ScratchObj. Palette restrictions are persisted to disk via ScratchStage, where PaletteSelector's array of Boolean's are transformed into a JSON representation and stored with other top-level project settings, as shown in Figure 9. The PaletteSelector Booleans are similarly updated upon project load from JSON, and are reset to true when the user starts a new project. Subsequently, the palette selectors, palettes, and scripts pane update to correctly reflect the newly loaded state of the palette restriction system.

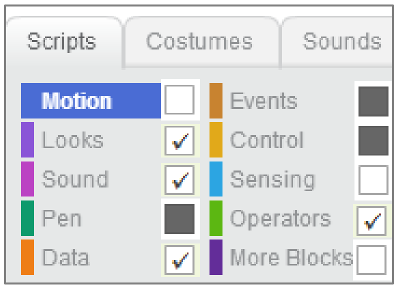
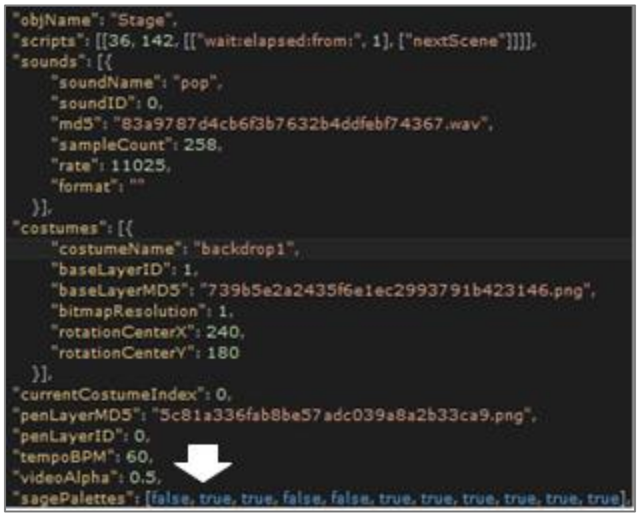


Figure 8



```
"objName": "Stage",
"scripts": [[36, 142, [["wait:elapsed:from:", 1], ["nextScene"]]]],
"sounds": [{
    "soundName": "pop",
    "soundID": 0,
    "md5": "83a9787d4cb6f3b7632b4ddfebf74367.wav",
    "sampleCount": 258,
    "rate": 11025,
    "format": ""
  }],
"costumes": [{
    "costumeName": "backdrop1",
    "baseLayerID": 1,
    "baseLayerMD5": "739b5e2a2435f6e1ec2993791b423146.png",
    "bitmapResolution": 1,
    "rotationCenterX": 240,
    "rotationCenterY": 180
  }],
"currentCostumeIndex": 0,
"penLayerMD5": "5c81a336fab8be57adc039a8a2b33ca9.png",
"penLayerID": 0,
"tempoBPM": 60,
"videoAlpha": 0.5,
"sagePalettes": [false, true, true, false, false, true, true, true, true, true, true],
```

Figure 9

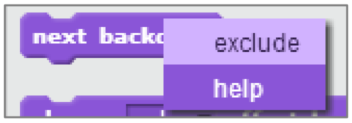


Figure 10

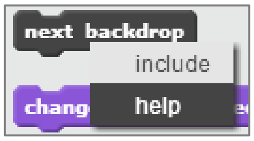


Figure 11

6.3. Block Restriction System

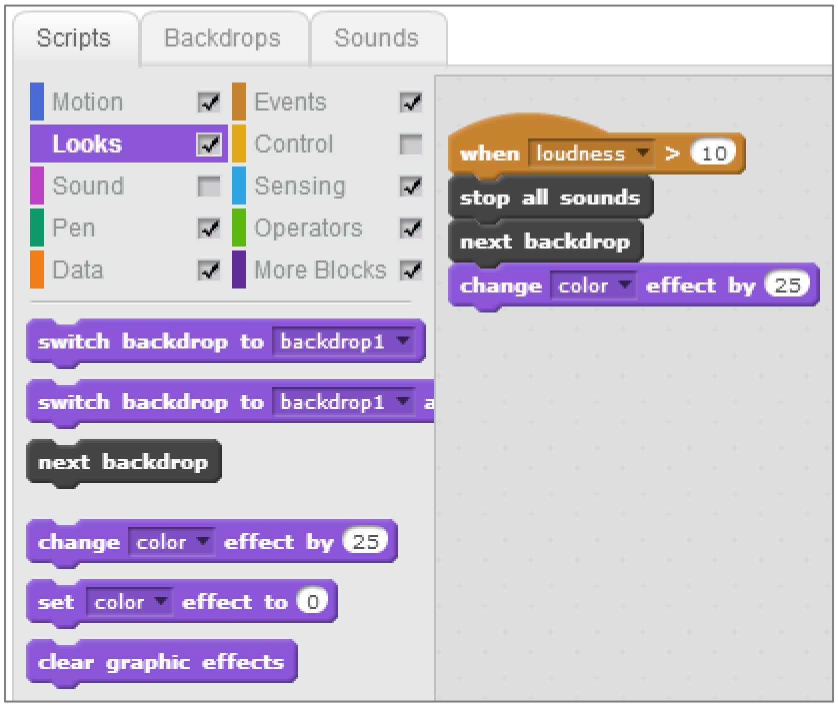


Figure 12

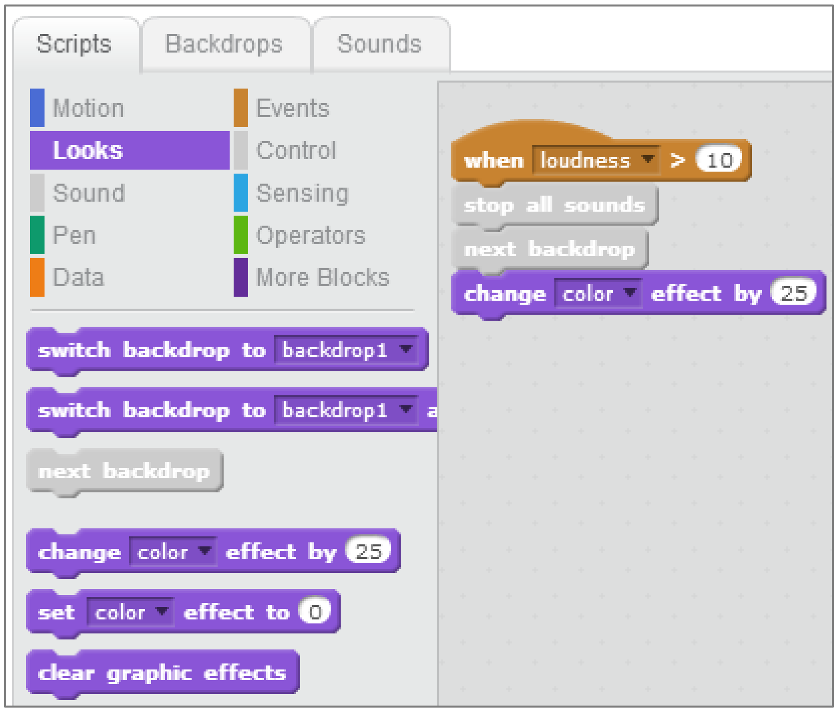


Figure 13

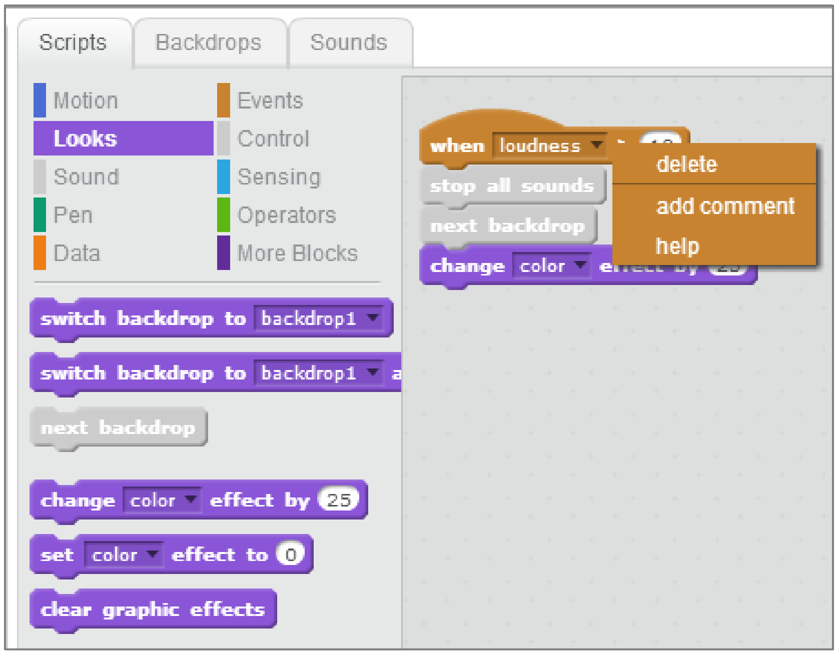


Figure 14

The block restriction system, configured via the context-menu available for each block when in Design mode, as shown in Figures 10 and 11, enables the educator to include or exclude particular blocks in unrestricted palettes from the active project when in Play mode. Only blocks in unrestricted palettes offer the include/exclude option, but the selection impacts

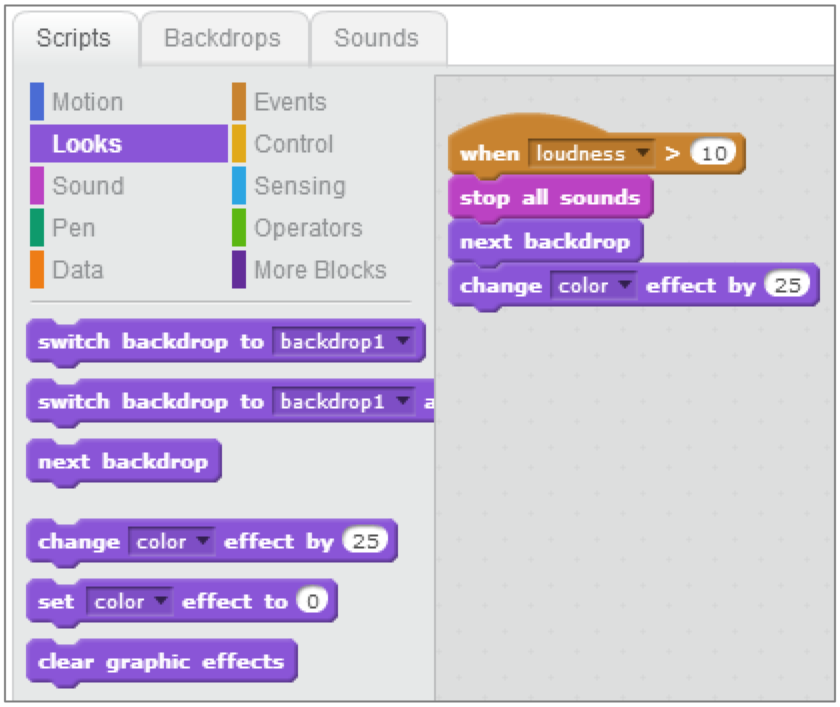


Figure 15

blocks in both the palette and in the scripts pane. Although palette restrictions override block restrictions, the block restriction system retains state through toggled palette restrictions. For example, if a block is restricted, then the palette is restricted, and subsequently unrestricted, the block restriction system retains the state of the originally restricted block. As in the case of the palette restriction system in Design mode, the block color is modified when excluded to provide feedback, as shown in Figure 12, but the blocks remain available for use. When in Play mode, the excluded blocks are a lighter shade of grey, as depicted in Figure 13, to indicate that the user no longer can drag the blocks into the scripts pane. Similarly, the context-menu is disabled for excluded blocks in Play mode, both in the palette and in the scripts pane, to prohibit restriction modification during play, and to prevent duplication of excluded blocks. Additionally, the "duplicate" menu item is unavailable for any stack containing excluded blocks in Play mode, as shown in Figure 14, to ensure the player cannot produce excluded blocks in the scripts pane. When the application is not in Design or Play mode, all blocks revert to their original Scratch look and behavior, as shown in Figure 15.

The primary classes modified to implement the block restriction system are PaletteBuilder, BlockMenus, Block, and BlockIO. Since PaletteBuilder generates the blocks for the palette selected by the PaletteSelector, it stores a dictionary of block labels and Booleans indicating restricted state. When the user includes/excludes a block, this dictionary is updated, the palette rebuilds, and any matching blocks in the scripts pane update to reflect the altered color and behavior. Block restrictions are persisted to disk via ScratchStage, where PaletteBuilder's dictionary of block labels and Boolean's are transformed into a JSON representation and stored with other top-level project settings, as shown in Figure 16. The PaletteBuilder dictionary of labels and Booleans

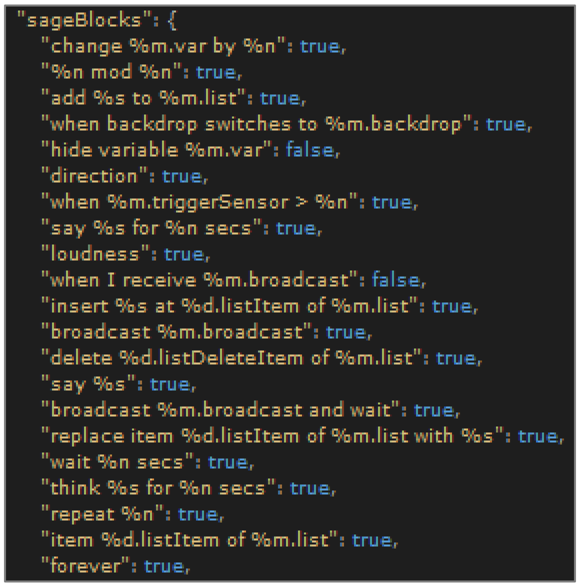

```
"sageBlocks": {
   "change %m.var by %n": true,
   "%n mod %n": true,
   "add %s to %m.list": true,
   "when backdrop switches to %m.backdrop": true,
   "hide variable %m.var": false,
   "direction": true,
   "when %m.triggerSensor > %n": true,
   "say %s for %n secs": true,
   "loudness": true,
   "when I receive %m.broadcast": false,
   "insert %s at %d.listItem of %m.list": true,
   "broadcast %m.broadcast": true,
   "delete %d.listDeleteItem of %m.list": true,
   "say %s": true,
   "broadcast %m.broadcast and wait": true,
   "replace item %d.listItem of %m.list with %s": true,
   "wait %n secs": true,
   "think %s for %n secs": true,
   "repeat %n": true,
   "item %d.listItem of %m.list": true,
   "forever": true,
```

Figure 16

are similarly updated upon project load from JSON, and the Booleans are reset to true when the user starts a new project. Subsequently, the palette selectors, palettes, and scripts pane update to correctly reflect the newly loaded state of the block restriction system.

7. CHALLENGES & INCOMPLETIONS

Despite the limited project scope, implementing this feature set presented four notable challenges, and two items remain incomplete.

7.1 Four Challenges

The first challenge involved properly installing and configuring the development environment in order to successfully compile the Scratch codebase. Eclipse, equipped with the FTD plugin for ActionScript, Flash, and Flex, served adequately as a text editor that properly recognized ActionScript keywords and offered formatting assistance, but I did not successfully operationalize it as a compiler and debugger. As recommended in the Scratch documentation, instead I installed Ant and modified configuration files and environment variables in order to create a reliable build process. While workable, this arrangement was less than ideal, since it led me to rely on alerts and notification windows in order to debug the voluminous errors encountered. Ant's output did offer helpful error messages with references to the files and line numbers that generated the errors, but since compilation usually consumed 25-30 seconds, the workflow of adding alerts, compiling, and running, proved less efficient than stepping through code in a debugger. I'm content with my decision to skip past resolving this problem early in the project timeline, because otherwise I might have run short of time to implement the proposed functionality, but I'm confident that if I revisit this issue I can overcome the difficulty and properly stand-up a development environment that allows me to compile, run, and debug Scratch from Eclipse.

The second major challenge involved determining which version and libraries of ActionScript were viable for use in Scratch development. I discovered that widely-used control

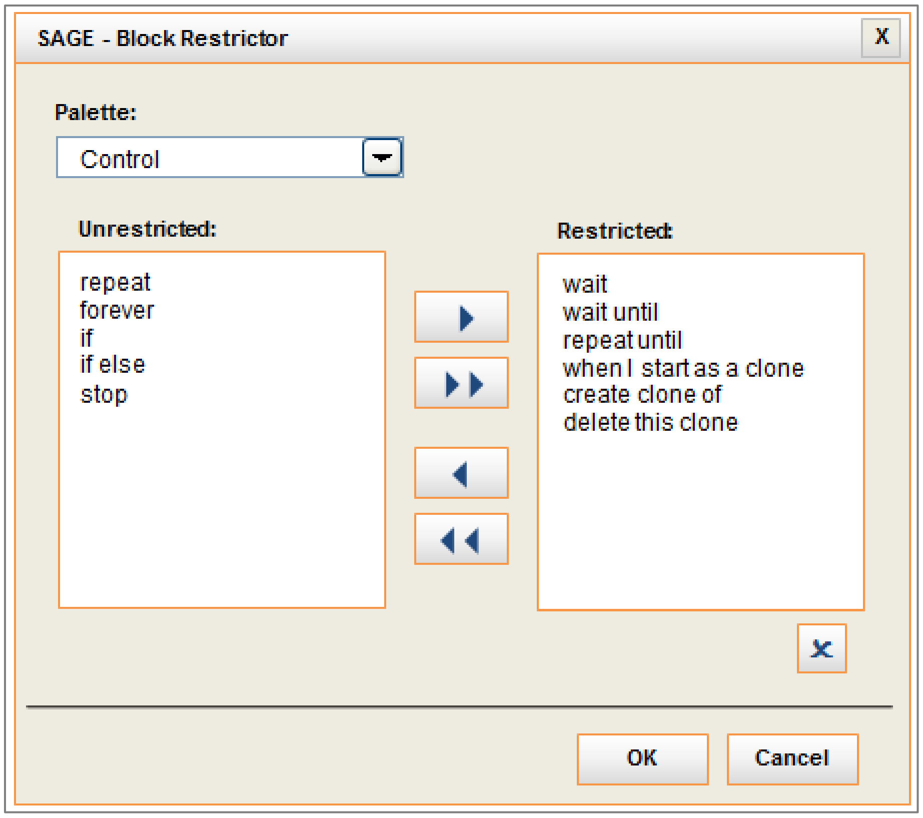


Figure 17

packages, such as MX and Spark, are not both freely available and compatible with the codebase. This realization helped me understand why the Scratch developers implemented classes I would otherwise expect to access via an API, such as IconButton, which can serve as a checkbox provided adequate image assets. The deficit in UI infrastructure led me to alter significantly the design for the block restriction system, since developing my own DropDownList and multi-select ListBoxes proved too lengthy a detour from the proposed project goals. Although the current design using context-menus might better fit with Scratch workflows than the original pop-up window proposed, shown in Figure 17, the interface now suffers from the lack of a multi-select feature. The anticipated forthcoming addition of restriction presets, templates, and examples, should reduce the impact of this interface deficit, but not resolve it fully.

The third notable challenge derived from the inconsistent object persistence pattern in the Scratch codebase. For example, blocks in the various palettes are not constructed once, and then displayed when the palette is selected. Instead, the blocks in each palette are constructed afresh every time a palette is selected. As a consequence, adding a property, such as "isRestricted" to a Block object does not make as much sense as one might presume, since the state promptly would be destroyed. The predominantly nicely-organized, object-oriented codebase unfortunately is used in a transient manner that prevents as much encapsulation as desired.

The fourth and last sizeable challenge relates less to the current project than it does to future work. The most recent codebase downloaded from GitHub, compiled without modification excepting configuration, does not produce a .swf Flash file that matches the functionality of the application hosted on the Scratch website. Important features, such as populated Sprite libraries, and the capability to fully save and load projects with blocks in the scripts pane, do not operate correctly. Now that I've become relatively well-versed in the codebase, I suspect I'll manage to resolve these problems, but I did find the discrepancy between the "advertised" and "found" functionality disappointing.

7.2 Two Incompletions

Certainly, from the broader perspective of long-term SAGE development, a massive quantity of required functionality remains to be designed, implemented, field-tested, evaluated, revised, redeployed, and so forth. A narrow comparison of the final project proposed and the implementation completed, however, reveals only one noteworthy shortfall. The block restriction system does not support the "Data" and "More Blocks" palettes, which consists of buttons instead of typical Scratch blocks. I began building a custom context-menu class to apply to these buttons, and was able to generate sprites that could appear in the correct locations, but ran out of time to complete the implementation. The "More Blocks" palette requires additional work beyond the custom context-menu, since it enables the creation of blocks, and support for block restrictions of user-created blocks demands substantial, but manageable, special-case handling.

The second incompletion relates to my response to feedback in the progress report. I indicated that if time allowed, I would attempt to include palette and block restriction presets, as well as embedded instructional material, in order to assist teachers, who likely have minimal computer science experience. While these features, as well as example projects and step-by-step guides, remain crucial for SAGE, they did not fit within the final project timeline.

8. LEARNINGS

The goals of this project were to become familiar with the Scratch codebase, which consists of over 35 thousand lines of code, to develop a baseline proficiency with the language in which Scratch is implemented, ActionScript, and to implement two of SAGE's design tactics, the palette and block restriction systems. Although these aims were modest, learning occurred throughout the pursuit of each of the three goals, and I'm glad to have pushed through the learning curves associated with standing-up the Scratch codebase and familiarizing myself with the patterns and nuances of ActionScript and Scratch. Implementing the palette and block restrictions systems required tracing code throughout the many packages in order to add the many key statements and functions that did not concentrate in the more substantially modified classes, such as PaletteSelector and PaletteBuilder. As a result, I have become relatively fluent in the majority of the code, and feel much better equipped to implement additional SAGE features now than I did prior to starting the final project. I'm most pleased, though, with having devised and validated the process of transposing learning theories into learning principles, design tactics, and implementable designs realizable as software tools for GBL. At the beginning of the semester, I struggled to determine how to synthesize all of the research for the survey paper into cogent GBL designs. I'm happy to have overcome the start-up friction, built momentum, and initiated a robust design and development process that will lead to sustainable SAGE progress.

9. CONCLUSION

To summarize, the functionality implemented includes initial SAGE Design and Play modes, and palette and block restriction systems that afford teachers, as game designers, the capability to scaffold CT learning outcomes. By actualizing one of the gameful design tactics highlighted in the midterm paper, this feature set supports the situated meaning learning principle and begins to enable the creation of goal-based scenarios and story-centered curricula. A few of the next steps for SAGE include employing the remaining design tactics identified in the midterm paper in order to design and develop many additional features; clearly, SAGE is far from a game system at this point. After completing design proposals this summer and continuing development this fall, I'm hoping to undertake User Centered Design cycles at the flagship Computer Clubhouse in Boston, with the goal of exposing SAGE to appropriately-aged children willing to inform and perhaps participate in some aspects of SAGE's iterative improvement and extension. Additionally, I intend to think more deeply about how to embed assessment and adaptive difficulty within SAGE. Both of those elements are not only extremely important for student learning, but also they offer opportunities to explore CS-centric techniques which might help fortify the research agenda from a CS perspective. Many techniques in learning analytics and educational data mining research present possibility, and I'm eager to identify which elements might most practically enhance this GBL system for CT. Before too long, though, as earlier noted, I suspect we'll want to evaluate progress, consider substantial redesign, and possibly redevelop SAGE for tablets. Thank you for the constructive feedback throughout the semester; as you have advised, I plan to concretize the theories, principles, and design tactics organized this semester into implementable designs and GBL software tools in the many months ahead.

10. REFERENCES

Action Message Format. http://en.wikipedia.org/wiki/Action_Message_Format
Apache. http://httpd.apache.org/
Ant. http://ant.apache.org/
CSTA, "CSTA K-12 Computer Science Standards: Mapped to Common Core State Standards," http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_Standards_Mapped_to_CommonCoreStandardsNew.pdf
CSTA Standards Task Force, "K–12 Computer Science Standards," Computer Science Teachers Association, 2011. http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf
Flash Player. http://www.adobe.com/products/flashplayer.html
Flex SDK. http://flex.apache.org/
Real Time Messaging Protocol. http://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol
SAGE 0.1 Demo. https://www.youtube.com/watch?v=EW4njO2ogig
SAGE 0.1 Source Code and Virtual Machine. https://drive.google.com/open?id=0B5VvJWwvpBFyZGxGMGdrOEs5TXM&authuser=0
Scratch. http://scratch.mit.edu/
Scratch Modification Wiki. http://wiki.scratch.mit.edu/wiki/Scratch_Modifications
Scratch Source. https://github.com/LLK/scratch-flash
ScratchJr. http://www.scratchjr.org/