Week Eight Milestone Report

Capstone Project

159.356

20th September 2015

Aidan Houlihan, Ashraf Alharbi, David Dudson, Josh Baker & Mitchell Osborne

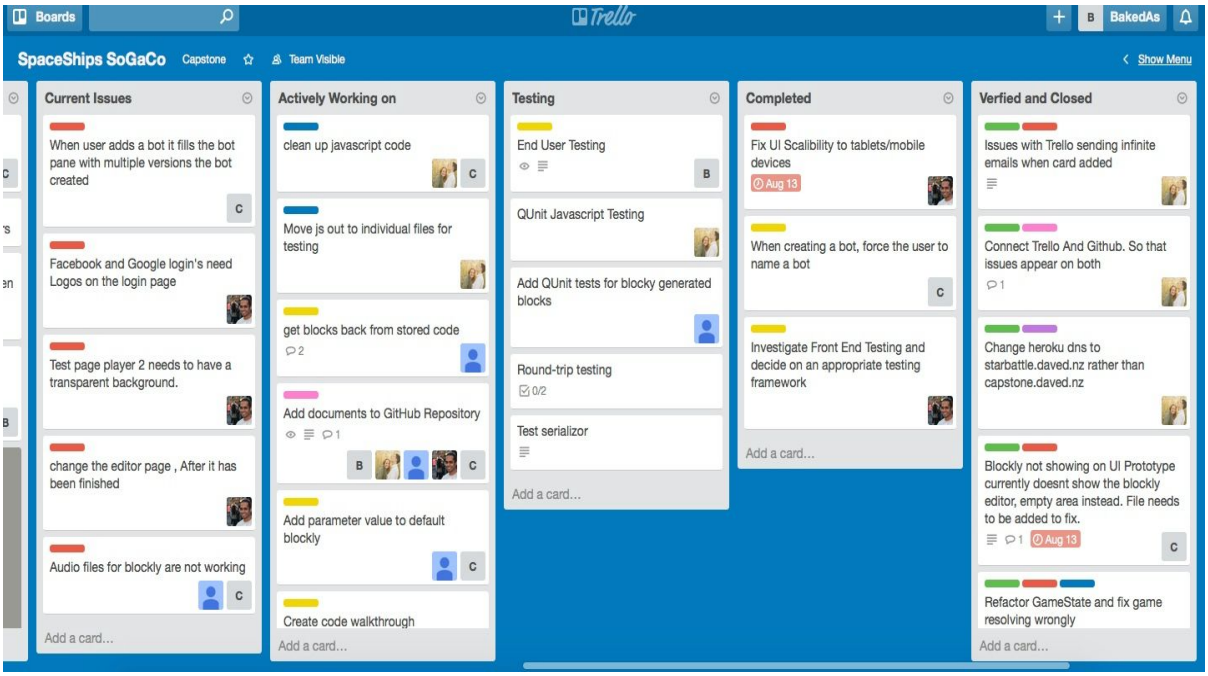
Table of Contents:

Process:

Updated Project Plan:

Throughout the evolution of the project the modules have become more reliant on one another and more integrated. To ensure that all members of the group continue to further the project as a whole, I have continued to update each member's goals and requirements on a weekly basis. Doing so has allowed all of the members to clearly understand what they need to complete each week because the four modules have become one. Each week, the current open tasks are assigned to the appropriate member and then given a priority for each of the members to fulfil (often the higher the priority, the sooner it is required to be completed). As each member works on the project, they may have to create a new task or update their current task. In this case, the issue is then added to our board on Trello and assigned to the appropriate member at the beginning of the following week during our meeting. Some tasks are smaller than others and therefore completed sooner, this often ‘frees up’ a member to work on another task or help another member with their current task.

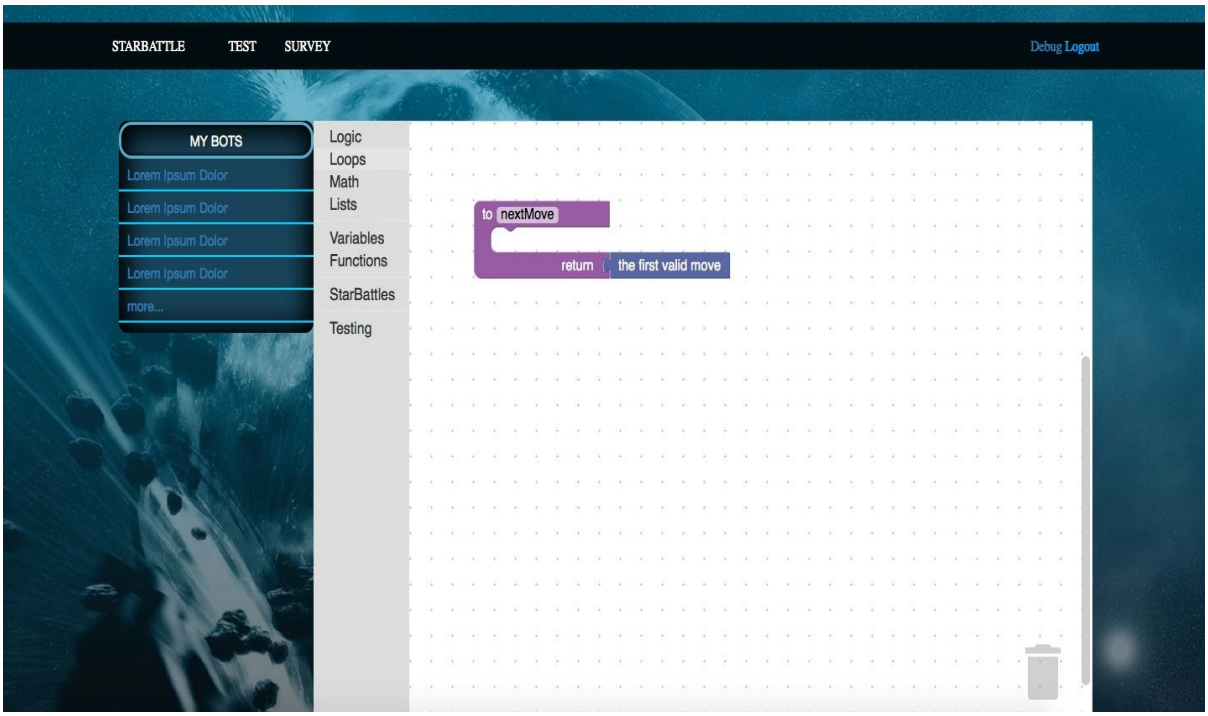
The project management and planning tool *Trello* has been instrumental in documenting the group’s current tasks, features and issues in our project. To mark the completion of a task, a group member to whom the task has been assigned to will move the task from its designated list to the ‘completed’ list on our group board. To further verify the issue has been completed, the project coordinator will then individually check each completed card and ensure that the work has been completed. Trello also keeps a track of where a card started along in the process and where it gets moved to, to show the progression of work and how a task came to completion.



Architecture:

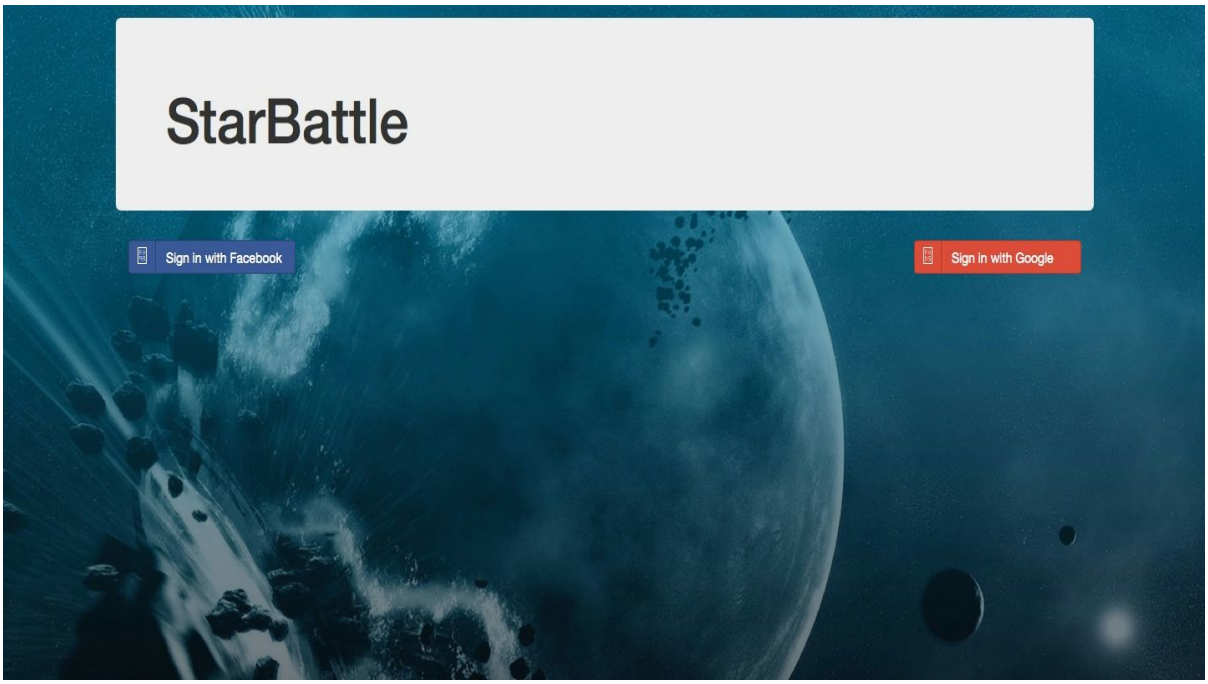
As the project has progressed, the four defined modules (The Game, User Interface (JSP), User Interface (CSS/Styling) and Visual Language) have been merged into one module through the build system. Having them initially separated allowed for faster development of the application as the workload was spread evenly. Now that the application is running as a whole, the team members work on the same processes and tasks together. Each member provides the unique knowledge of their own module to allow the modules to be integrated together.

The Blockly module contains the visual language which allows a user to create a bot modify them and delete them if necessary. The user will create a bot by dragging and dropping a block into an editing area, the module takes this input and converts the block into usable Java code. The opposite occurs when the user edits a saved bot. The module must store the information saved by the user on the SoGaCo platform.



The game logic module involves the logic for a game to run on the application. It takes the Java code and executes the function created by the user when they save the bot. It parses a JSON object to the server which the server must read in and generate a serialized version of the input from the Blockly after it is converted to Java.

The user interface contains both of the Blockly and game logic modules but also interacts with the SoGaCo platform. When opening the application, the user is required to sign up to the platform using either a Facebook or Google account to login. After logging in, the user creates a bot from the blocks given in the editor page. A sample piece of code is given to the user to modify further. After creating a bot the user can save it and then navigate to the test page where they can run the bot against other bots created by other users or predefined within the API. When a bot is saved it is stored on the SoGaCo platform through the user interface.



The SoGaCo platform holds the game logic for running a game and interacts with the user interface to run the game and animate the game being played. Before the game is rendered on the front end, the API knows which bot will win the game and returns the serialized version of the game to the front end to display.

Project Infrastructure:

During our first setup of the project we said we would use Google Groups, Facebook chat and GitHub as our means of communicating, uploading code and documents. We found this to be too cumbersome requiring the uploading of a document in one place then reloading it in another, it also had the potential to have some important documents misplaced. Since week 5, we have changed to using Facebook chat exclusively for discussion on the project. We often update each other on our progress or ask other members for help. We are now using GitHub as a repository of for our code and documents and we are using Trello as our ‘development board’ which contains all of our current issues. All of the information on the Google groups has been transferred to the GitHub repository. This has made consolidation of information simpler and communication within the group easier.

For the duration of our project, trello has been used as our main issue tracking system. It provides a simple visual display of the issues or features as well as allowing them to be categorised, assigned to members and commented on. When a team member encounters an issue or feature to be implemented, they are free to add it as a card to the board and label the category it falls under (issue, feature, requirement, refactoring, documentation, completed and minor issue). These issues are then reviewed at the start of the following week by the team coordinator and then reassigned as necessary. This is done to ensure that the team members do not get overwhelmed with the number of issues/features assigned to them. It also balances out the workload for the other members that have fewer issues assigned to them. There will always be some features or issues which can only be completed by a specific team member. In the event of this, the workload will be shuffled between team members until a balance is found. Once an issue or feature has been completed by a team member, the issue can then be labelled ‘complete’ and moved to the completed list. The list of completed issues/features is then verified by the team coordinator and finally moved into the verified and closed list.

We have used trello as a way of expressing ideas and potential features; this means that we have a variety of card list types from potential features to issues, testing and backlogs. These list types help to identify what has been done, what was not done and what needs to be done.
For our project we have used GitHub as our main code repository. There have been a total of 2 branches off the master branch excluding spikes, one for each core module; toebes-blockly and webUI, most of the recent work to the core infrastructure has just been developed on master as it has mostly been fixing minor issues. There have not been many feature implementations recently, all of the work has been related to core requirements and testing. No further branching has occurred.

The commit timeline shows a steady number of commits per day averaging about 17 commits in total per day. Each member has committed a variable amount of times as some modules require less committals than others, David has the most with ~65 commits, Aidan follows closely with ~60 commits, Mitchell has ~40 commits Josh has ~5 commits and Ashraf has ~3 commits. These numbers are to be expected as David has to commit changes to the server configuration and regularly to ensure that the application continues to work, Aidan has been working with David integrating the game logic with the server, Mitchell had to complete his research into the Blockly code before starting on writing the API, Josh mostly works behind the scenes and only needs to upload documents to the repository and Ashraf has been doing large amounts of work then submitting all at once. This is ok as the user interface is not required for the application to work.

Quality Assurance:

Quality Analysis:

Part of making an application viable for end users is to test the modules which make up the overall application and verify they work as specified. To ensure the quality of our application we have focused on testing the application's individual modules and how they interact with the other parts of the application. We began by testing the individual units that made up our modules, and then we tested the unit's interactions with each other to ensure that we have a working module. This was done for each of the larger modules before they were tested together as a whole working application. Each team member was responsible for their own module and testing individual units.

For the visual language Mitchell focused on testing the Blockly code to ensure it output the correct Java code. This was an important part of the application as the code that encodes the blocks needs to be valid and work correctly with the other blocks and output the correct Java code. As this testing focused mostly on the output of the system, a black-box style of testing was used. QUnit was the framework

chosen for this style of testing because it gives an easy to use environment which displays results clearly, it is powerful in its ability to test JavaScript and has a large user base and thus a lot of support. Mocha and Jasmine were also considered as testing frameworks however these have less support as they are newer than QUnit and Mitchell felt more comfortable with QUnit as he was more familiar with the framework. QUnit also uses test driven development whereas Jasmine uses behaviour driven development, a methodology we were not using. QUnit's simplicity made for a cleaner test framework which is ideal for blocky (the main library we are using for our blocks). This is because the Blockly blocks have a very specific set of checks that need to be done, for example, you need to check the inputs and outputs of the blocks are of the appropriate type. You also can easily test the generated code from a Blockly block is correct with a simple string comparison. To execute the tests, an HTML page is created with two QUnit files located within the header (QUnit.js and QUnit.css). On load, the tests written within the page are executed and an HTML page displays the results of the tests executed. The output of each block was tested as well as the default bot that was created. The written tests cover the following: Java generation from blocks, input and output connections of blocks, the default bot generation and the input connection type of the nextMove function block.

Some parts of the JavaScript on the frontend remain untested, this includes things like HTTP requests, DOM updates and JSP variables. When we began testing this section we found that we had pushed forward with development to get the application working. This meant that there was no structure to this section, so parts of it were untestable in the current state. To actually test this there were 3 forms of testing required, DOM tests (which are partly covered by Selenium), HTTP mocks, and JSP tests. The JavaScript was also in script tags on the JSP's so there was no ability to test the JavaScript code independently of the JSP. David has been working on making testing much easier in the future and simpler to both develop and test.

The front end has been tested by Ashraf using the Selenium in browser testing platform. The tests mostly focused on testing whether or not the HTML and CSS elements load into the webpage. This is important because if a change occurs that

should result in an HTML or CSS element not loading correctly, we can see where the test failed and fix the issue. Selenium as a platform was chosen because it provides a lot of documentation and allows for automation of tests using Java code. Automation is very helpful as it allows the test to be run on build which can show us errors in the HTML or CSS. Other HTML and CSS validation has also been done to validate the actual content of the pages using w3c validator. To run the tests, you need to install Selenium as a plugin into your browser. You then click on the record button, access the browser and then click the elements on the page. Once testing has been completed stop recording actions and review what the results of the test were.

As the backend of the code is just as important as the front end code, the same test driven development strategy applies. We used JUnit to test the backend Java code generation from Blockly as well as the game logic on the server side.
JUnit was chosen as a testing framework as it has plenty of libraries, support and is what we are familiar with. We also believe that the game code needs to be tested thoroughly at the unit level and JUnit allows for this level of testing. The back end tests currently cover the game logic, checks that when a bot is created it is valid and checks for copying such as use of shallow or deep copies. We plan to do further testing on the application (front and backend) in the coming weeks.

In regards to integration testing and round trip testing there are 2 things that need to be completed before this is possible. Firstly we have to be able to get the Blockly xml representation back from stored code, that way we can validate the full save/load cycle. Secondly we need the JavaScript code to be tested, this allows us to verify that when playing a bot game the data that is returned is reliable.

All build scripts and commits to the git repository have been automatically tested by TravisCI. When a commit occurs, TravisCI will try to build the project and run tests on the submitted code. This includes commits to all branches, not just the master branch. This ensures that the code pushed to the repository is valid and works with

our application. If the code is invalid the build fails. On a successful build of the master branch, the code is automatically shipped to a live server on heroku.

Other Quality Assurance:

Initial end user tests have been completed. These were done as a gauge of how the application currently is for the end users. It asked some basic questions about layout and usability as well as the respondent's details so that we could follow up their responses. The user tests involved three ‘students’ who were given a set of instructions to follow and then asked how easy the tasks were to complete. The questions and the responses can be found in the attached appendices. As some of the application is not yet fully functional, the responses were limited and critical, this however provides a point of reference for us to work on the application and change depending on the user’s feedback. The end user tests involved a short walk through for the users to create a bot and run a game. However as the application is not yet fully complete there was not much the users could edit. A comprehensive code walkthrough will be created to ensure that when the application is launched the user will be able to create a bot of their own with ease. It will explain how to create, save and then run a game with their own created bot versus the premade bot or against a friend’s bot. Some navigation tips will be added to ensure that the user knows where to go without getting lost. All documents have been uploaded to the git repository under the documents folder.

Throughout development, the application has been tested using various browsers as well as different media such as desktop computers, laptops and tablets. So far we have found no issue with using the application on Chrome, Safari, and Mozilla Firefox. Internet Explorer 9+ will be tested in the near future. The application works well on all screen sizes greater than 13 inches; however some scalability issues have arose with the use of the application on a tablet or a screen smaller than 10 inches. These issues will be updated and changed in an upcoming iteration of our application (scrum meeting).

Product:

Application link: starbattle.daved.nz

The application allows the user to do the following:

- Login to the application using social logins such as Facebook or Google
- Create, edit and save a bot
- Run the saved bot against another bot or one of two default built in bots

These executables work only on a local server at the moment but will be pushed to live within the coming weeks. The current product will be demonstrated later this week.

Appendices:

Project link:
starbattle.daved.nz
GitHub:
https://github.com/
Blockly:
https://developers.google.com/blockly/
Facebook:
https://www.facebook.com
TravisCI:
https://travis-ci.org/
Java:
https://www.oracle.com/java/index.html
W3C Validator:
https://validator.w3.org/
HTML 5:
http://www.w3.org/TR/html5/
CSS:
http://www.w3.org/Style/CSS/Overview.en.html

Initial End User Testing Group One Responses

Timestamp	What's your name?	How easy was it to login to the website?	What do you think about the layout of the editor page?	Create and save a bot	Play a game	How did you find creating a bot to play?	How did you find playing the bot you created?	Do you have any other comments about this application?	What browser did you use?	What type of device did you use?
18/09/2015 13:53:56	Francis	Easy, just one click. I used Google.	Simple. It showed all options however i didn't know why a java code editor was underneath the one we were meant to use. Looks good.	done	done	Easy, it was just one click.	I didnt play the bot, i just watched.	It was good.	Mozilla Firefox	Desktop / Laptop
18/09/2015 14:26:44	Jack	Easy. Facebook.	Looks good.	ok	ok	Easy.	Easy. I didn't play though the computer did it for me.	No	Chrome	Desktop / Laptop
18/09/2015 14:08:09	James	It was ok. I used facebook but it was slow.	Looks good. Navigation at top is good.	.	.	There wasn't much to create, as the predefined bots were already done for us.	Looked ok.	Fix scalability for smaller windows.	Chrome	Desktop / Laptop