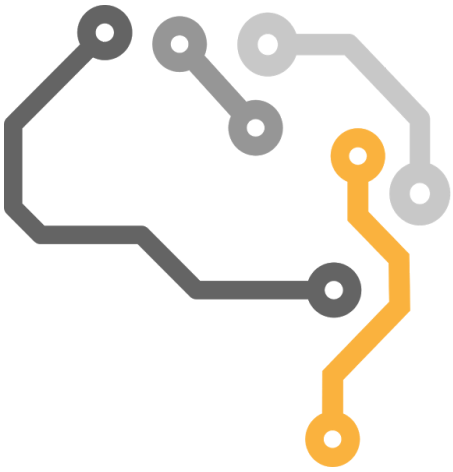
Requirements Specification

Date: 12/6/2017

Project sponsor: Barbara Jenkins

Team faculty mentor: Ana Paula C. Steinmacher



BRAINSTIM STUDIOS

Vincent Messenger (Lead)
Anderson Moyers
Nathan Franklin

Accepted as baseline requirements for the project:	
Client:	Team Lead:

Table of Contents

1. Introduction

Alzheimer's Disease, or AD, is a progressive form of dementia which gradually destroys mental function and memory. It often manifests as short term memory loss like forgetting minor details in early stages and it progresses to pervasive, long-term memory loss like forgetting essential functional tasks and loved ones. In the last stages of AD, cognitive function declines until bodily functions are impaired, ultimately leading to death. As of 2015, there were an estimated 29.8 million people suffering worldwide from AD. It is the sixth leading cause of death in the U.S., a new case is diagnosed every 66 seconds and more than 5 million Americans live with the disease at a cost of $259 billion per year. Without any treatment, those numbers are projected to explode to 16 million Americans with the disease, at a cost of over $1 trillion a year, by 2050.

Research indicates that fortunately, regular cognitively stimulating interactions can reduce the risk of Alzheimer's Disease. This research has precipitated interest in playing brain stimulation games to keep the brain healthy and game companies have started researching and designing games for this purpose. Lumosity, an online site, is an example of a gaming platform that offers this type of cognitive gameplay. Lumosity relates their game design to studies conducted on how humans learn: the idea is that by giving users a fun way to challenge their brains, users can keep their brains healthy and reduce the symptoms of degenerative brain diseases.

Barbara Jenkins, our sponsor, has created a fast-paced word game called WordScuffle that incorporates social gameplay in order to provide users with maximum potential for increasing their brain health. The game generates random letter sets with which the user will have three minutes to construct as many words as possible. Users, or "players", construct words in a grid-like fashion, which allows words to overlap. Once a game is finished, a player's score is calculated and they can compare their score and words with other players of the game.

There are different game modes that present users with different scoring systems. This forces users to adapt the way they think to the challenge that is presented. The game will generate ten letter sets per gametype every day that each user can complete. Once a given letter set has been completed, users can then compare their scores with other users of the game. On top of this, the user will have the option to play unlimited practice games, where unique letter sets will be generated at the beginning of each game. However, these practice games are not eligible for community comparison.

WordScuffle currently takes place with a pencil and paper, time and scores are manually kept, and results are compared through email. It takes considerable time to tally up scores, scores and results are viewable by players before they may have finished their own tileset, and there is enough entropy in the game's workflow that more time is spent with minute tasks of

gameplay than playing the game. Because much of the gameplay requires “manual” human processing, there are numerous chances for error.

Our team BrainStim Studios is working with Barbara to realize this game as a web application and resolve these workflow problems to make the game more fun, more interactive, less tedious, and even more socially stimulating. Our web application will offer automatic, integrated word validation which will reduce misspelled words. A score calculator will also be updated as users construct words onto their board. Scores will be maintained in a database, where players can retrieve scores and results from other players. Word validation combined with more robust scorekeeping will eliminate human error and reduce entropy in the game. To boot, our scoring system will improve competitiveness because it prevents players from seeing results before they have finished their own set. To enhance social stimulation, we will provide players with a way to create communities with other players, so they can filter high scores to just those they wish to see.

In this requirements specifications document, we go into great detail about the requirements that we have discussed with our sponsor that will need to be satisfied in our finished product. The technologies we will be using to produce the finished product were carefully picked out so that the project requirements can be met. These requirements have been classified as functional, non-functional, or environmental. With a clearly defined set of requirements, we will be able to use this document as a contract with the client for completing the project. If our software meets all the agreed upon functional requirements and performance requirements, the client should be happy with the finished product we supply them with in May 2017. The requirements will allow us to stick to a plan for the development of the software, and the client should not be surprised by the software we present to them. By thoroughly discussing with the client what they want us to produce, we will have a set of requirements that need not change as the software is constructed.

2. Problem Statement

Although there are existing platforms which offer cognitive gameplay, these platforms do not adequately address the social components of cognition in preventing the onset of AD. Lumosity’s games for example offer no wider social context: gameplay is solo, players have little chance for interaction and players cannot learn from the results of others and compete.

WordScuffle offers a chance to have fun with wordplay while specifically addressing the social brain-flexing that other platforms lack; the game however is currently played with pencil and paper, time and scores are manually kept, and results are compared through email. It takes considerable time to tally up scores, scores and results are viewable by players before they may have finished their own tileset, and there is enough entropy in the game’s workflow

that more time is spent with minute tasks of gameplay than playing the game. Because much of the gameplay requires “manual” human processing, there are numerous chances for error.

Our team BrainStim Studios is working with Ms. Jenkins to resolve these workflow problems to make the game more fun, more interactive, less tedious, and even more socially stimulating. Our web application will offer automatic, integrated word validation which will reduce misspelled words. A score calculator will also be updated as users construct words onto their board. Scores will be maintained in a database, where players can retrieve scores and results from other players. Word validation combined with more robust scorekeeping will eliminate human error and reduce entropy in the game. To boot, our scoring system will improve competitiveness because it prevents players from seeing results before they have finished their own set. To enhance social stimulation, we will provide players with a way to create communities with other players, so they can filter high scores to just those they wish to see.

In the following sections we outline the overall picture of how we will translate the workflow of Ms. Jenkins’ current implementation to software, the key requirements we must fulfill in order to successfully complete our implementation, and how the technologies we have chosen will meet our needs in this process. We begin with our Solution Vision in the next section.

3. Solution Vision

In order to solve the outlined issues, our team is working with Ms. Jenkins to realize WordScuffle as a mobile-friendly and highly responsive web application. The team will develop a server to perform tasks related to user management and gameplay. The team will also create a frontend client that will allow users to play games, manage their user data, and participate in community aspects of the game.

3.1 Server

The team will be using Node.js to create a server to perform tasks related to user management and authentication. This server will communicate with Firebase to allow users to create accounts and login to their accounts.The server will also use Firebase’s Realtime Database to store gameplay data such as user data, community scores, and gameplay configuration.

This server will also perform tasks related to gameplay. The server will be responsible for generating and serving random tilesets that players will use to construct their word grids. It will also perform the word validation as players construct their words, as well as scoring for each

word that is validated. Once a user has completed a challenge, the server saves the word grid and score into the Firebase database.

3.2 Frontend Client

The team will be using Angular 4 to construct a highly responsive user interface that will allow users to use the various features offered by WordScuffle. The server will serve randomly generated tilesets to the frontend client, which players will use to construct words onto a play grid. Once the game time limit has passed, the frontend client will send the constructed grid and score to the server to be saved.

This frontend client will also allow the user to manage their account data and participate in community gameplay functionality. From the frontend client, the user will be able to change the personal information associated with their account. They will also be able to form communities with other users in order to compare scores and constructed word grids.

3.3 Workflow Diagram

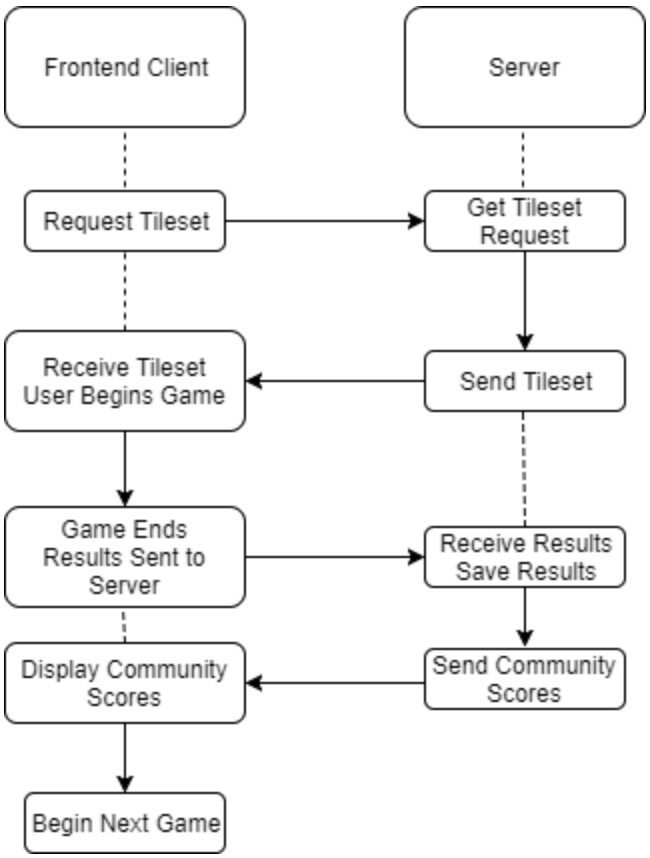


Figure 3.3. Workflow diagram.

Figure 3.3.1 shows an example of a potential workflow for our solution. In this example, the frontend client requests a tileset from the server. The server responds with a randomly

generated tileset, which the frontend client uses to start a new game for the user. Once the user finishes the game, the frontend client sends the results to the server to be saved. After the save has happened, the server then responds with the community results for that particular letter set. Once the community results have been displayed, the user can begin another game.

4. Project Requirements

To acquire project requirements for our implementation of WordScuffle, our team has been meeting bi-weekly with our client for approximately three months. During our first month of meetings we played WordScuffle to gain basic ideas about how the game is played and we discussed basic requirements for the implementation such as Mrs. Jenkins' desire for the game to be a Web 2.0 application and to be mobile friendly.

In our last month of meetings, we focused on honing domain-level requirements to reach main functional, non-functional and environmental requirements. We approached requirements acquisition by examining the way the game is currently played manually and then conceiving of a comparable workflow for our web application implementation of WordScuffle. We used the workflow of how a user will interact with the web application to determine our functional requirements. We were then able to extract non-functional and environmental requirements based on what we determined we must have functionally.

The main domain-level requirements determined by our project sponsor are the following. First and most importantly we should be able to play the pen and paper version of WordScuffle with our new implementation. Our implementation should secondly be a Web 2.0 Application which combines with our third requirement that it be playable on mobile devices, especially iPad. For our fourth domain-level requirement, we should include an administration console so that an administrator can change global settings for our WordScuffle once it is up and running, manage data from the game, and manage accounts and communities. Fifth and finally, we must design our web application with monetization in mind.

4.1 Functional Requirements

We have determined the following nine functional requirements which are necessary to implement WordScuffle as a web application:

FR-1: Construct Words
FR-2: Keep Score
FR-3: Track Time

FR-4: Validate Words
FR-5: View/Compare Results
FR-6: Participate in Communities
FR-7: Manage User Profile
FR-8: Administration Console
FR-9: User Authentication
FR-10: Game Rules

FR-1: Construct Words

Constructing words is one of the main functional requirements of playing a round of WordScuffle. In order to make WordScuffle playable on mobile devices, we will implement tile movement with drag-and-drop functionality. A user will receive a bank of tiles from which they can drag tiles onto a grid to form words. The grid will be at least 13x13 but should be designed to accommodate spaces up to 16x16 (grid size will be statically defined by an administration setting and the grid will be generated accordingly). Both the grid and tiles will adjust automatically in size according to screen size and the size of both the grid and the tiles must correspond to one another at any given time. The user should be able to drag tiles to the grid and back. Words can be constructed vertically and horizontally.

FR-2: Keep Score

Keeping score is another functional requirement of playing a round of WordScuffle. As words are formed on our gameplay grid each round, a user's active score for that round is always displayed. The score will be incremented for each correct word that is validated. If letters are dragged from the grid and this causes words to become invalid, the score will be decremented and displayed accordingly. When a round is complete, our web application will automatically save the user's score for that round in a cloud-based database. As a user plays more rounds of WordScuffle, their score history will continue to accrue in our database.

FR-3: Track Time

When a user begins a round of WordScuffle, our web application will begin timing the round and it will display the time throughout that round. The default time for each round, as determined by Ms. Jenkins, is three minutes. The global setting for the length of a round will be editable on the administration console. When time is up, the game is over and our application will exit the wordplay screen. Time tracking will be implemented in a way that cannot be manipulated by a user during gameplay.

FR-4: Validate Words

Word validation occurs throughout each round of WordScuffle. When a singular non-adjacent tile is placed on the board it will not be validated and will automatically be

assumed to be an invalid word. This is because our implementation and our dictionary do not accept one-letter words. Word validation for a given array of tiles on the board will begin when a user completes the placement of a second adjacent tile and will continue to occur after each successive tile placement within a particular array. Word validation will also occur after a tile is removed from an array on the board, as long as the array remains at least two tiles long. Each time an array of tiles is changed, (whether it is one tile long or two or more tiles long), our gameplay grid will provide visual feedback to the user by changing the appearance of the entire tile array to indicate that it is either an invalid or valid word.

FR-5: View/Compare Results

An instance of a gameplay result includes a user's final score and word configuration for a round; this data is saved after each round of WordScuffle. Users will have access to view results independently of playing a game, available at any point from the main menu after they are logged in. A user will also have the opportunity to view results after finishing a round of WordScuffle. When a user views their gameplay results they will be given these results in relation to their community members' score data (if they are participating in a community). Viewable gameplay results include historical results occurring previously to the current day as well as a ranking with results for the current day relative to other members of the community.

FR-6: Manage Communities

After a user logs in, they will have access to a link in our applications main menu for accessing community information. This link will take them to a main section that gives them the option to view, join, or leave a community. A user will also be able to view basic information like the community name and number of members from their user profile if they have joined one. Any functionality outside of viewing the community in their user profile will redirect the user to the main community management page.

FR-7: Manage User Profile

When a user signs up for WordScuffle, they will be prompted to create a basic user profile with first and last name, email address, and a password. Further customization is not required to participate in communities or gameplay, but we will provide options for further personalization. Similar to managing communities and viewing gameplay results, a user will be able to access their profile for viewing or editing from the main menu at any point after logging in. From the user profile page, they will have the options to create an avatar, view personal information, manage their membership type (we are designing our web application with the possibility of monetization in future implementations), distinguish which information is viewable by others, and view basic information about the community they have joined.

FR-8: Administration Console

Ms. Jenkins has requested an administration console which enables an administrator to manage variables about the game. We will provide authentication access to the console through the same login portal created for all users of the web application. An administrative user will have an additional option (beyond what a non-administrative user will have) to access the console from the main menu. The administration console will include options to remove or "ban" users from the web application in the case of use violation. It will include options to change global gameplay variables like the number of tiles generated per round (this affects the creation and layout of the game-grid as well as tile set generation), the number of rounds playable per day (this affects the number of tile sets generated per day), the time allotted per round and the score weighting. There will be separate global variables specific to the two different game modes.

FR-9: User Authentication

Logging in or authenticating is vital to our implementation because many of our other functional requirements require user authentication. For instance, providing an administration console requires our web app to differentiate between users to give administrative access to users only when appropriate. Participating in communities, score tracking, and comparing scores also requires the ability to remember a user's settings and previous history which is not possible without authentication. Creating and managing a user profile is also directly dependent on the ability to authenticate a user.

FR-10: Game Rules

Ms. Jenkins has supplied us with specific rules for the wordscuffle gameplay. There are general rules that apply to all gameplay, like every user getting the same generated tileset because users will compare scores for the specific tilesets, and there are specific rules that make the two game types unique.The player will have the ability to view the game rules for the different game modes that are offered.

4.2 Non-functional Requirements

For non-functional requirements, our team has focused on the following:

NFR-1: Letter tile generation
NFR-2: Letter tile attributes
NFR-3: Word validation attributes
NFR-4: Device Compatibility
NFR-5: Reliability of Server

NFR-1: Letter tile generation

Tile set generation is a vital mechanism which happens on our server to support gameplay in our web application. A pre-defined matrix of letters will be created statically on our server. Each day, our server will generate a randomized set of letter tiles from this pre-defined matrix. The *number of sets* generated corresponds to the number of rounds playable by each user for that day and the *number of tiles* within a set corresponds to how many tiles a user will be given for each round of WordScuffle. All functional requirements related to gameplay are directly dependent on letter tile generation; these include *FR-1 Construct Words*, *FR-2 Keep Score*, *FR-3 Track Time*, and *FR-4 Validate Words*. These are directly dependent because a round of WordScuffle is not possible without generating the necessary tiles. Letter tile generation is also related to the functional requirement *FR-8 Administration Console* because an administrator will dictate the way generation occurs through the console.

NFR-2: Letter tile attributes

In order to most naturally map the experience of creating words from tiles on a game-grid to real-world experiences and interactions, we have determined that tile manipulation should fit within several constraints. This requirement is a quantitative component of our functional requirement *FR-1 Construct Words*.

First, when a user touches a letter tile it must provide feedback that it has been “picked up” by the user’s pointer (could be a mouse, finger or any other pointing device) within 500ms. Secondly, as the pointer moves, an area within the tile should encompass or overlap the area directly below the center of the pointer at all times. This is necessary to provide ongoing feedback that the pointer is in contact with the tile. Thirdly, when the user places a letter tile on the game grid, the tile should snap to a grid pattern that aligns tiles vertically and horizontally: this means that if a user does not align the tile perfectly within the boundaries of a grid square that the tile snaps to a grid square that corresponds to the majority of where the tile overlapped. The grid will provide a “ghosting effect” to accurately show where a tile would be placed if the user dropped it as they drag it over the grid. This same ghosting effect will occur in the tile bank container to show where the tile will be placed if the user drags a tile back to the bank.

NFR-3: Word validation attributes

To make gameplay interactive and accurate, our word validation must be responsive and accurate. This requirement is a quantitative component of our functional requirement *FR-4 Validate Words* and it is associated with other functional requirements such as *FR-1 Construct Words* and *FR-2 Keep Score*.

To ensure word validation does not impede gameplay in a timed, short-span environment we have determined that one complete instance of word validation must occur within two seconds. This two seconds encompasses the entire feedback process from the moment at which a tile visually indicates that it was dropped onto the game grid to the moment at which an array on the board visually indicates that it represents an invalid or valid word. The

process of checking our server's local cache and then sending and receiving a verification to an external resource must occur within remaining time not spent creating feedback.

With respect to industry standards, our word validation will accurately reflect what is contained in the Scrabble$_{TM}$ dictionary. This means for instance, that our gameplay will not "accept" a word that is length one.

Both of these quantitative attributes affect *FR-1 Construct Words* because they affect the effective speed at which a user can interact with the game as whole and construct words on the grid. This is extremely important in an environment where gameplay is short. They directly affect *FR-2 Keep Score* because score cannot be accurately kept if validation is not both timely and accurate.

NFR-4: Device Compatibility

Based on project requirements defined by our project sponsor, our implementation of WordScuffle should be both playable on mobile devices and a Web 2.0 application. This means that our implementation must be a web application that is fully scalable and playable on everything from mobile devices to desktop browsers and it must render similarly visually across a variety of browsers. For browsers we expect to support 95% of the American population. We expect to support a screen size down to 320px wide.

NFR-5: Reliability of Server

Because our web application will be served, we have a non-functional requirement to provide reliability and a contingency plan in case of failure. Given a failure of our server, we must automate rebooting and automate notifications with server status for quick follow up from a support person or developer. Providing redundancy to restore settings from a catastrophic failure is vital. If such a failure occurs, users should be given feedback within several seconds that the application's backend is out of service. Whatever gameplay has occurred client-side should be saved until the server is back up to avoid user frustration.

4.3 Environmental Requirements

Based on our implementation, these are the external prerequisites necessary (which are out of our personal control) for our web application to work:

ER-1: Internet Access
ER-2: Supported Device
ER-3: External Resource Reliability

ER-1: Internet Access

In order for a user to access our web application they must have internet access. This environmental requirement is inherent in the nature of our implementation as a web application and there is nothing we will implement to mediate this requirement.

ER-2: Supported Device

Having a supported device is another user-based prerequisite similar to *ER-1 Internet Access*. We will support devices down to 320px wide because this is fairly standard minimum screen size and few smartphones currently exist which are narrower. However, there are still devices on the market that could feasibly access our web application someday but for which our web application will not be designed: the Apple Watch is an example of such a device because of its size. Our application will detect screen size when it is accessed by the user and if screen size does not fit within our scope, we will notify the user with a simple page that WordScuffle cannot be accessed on their device and provide the system requirements of playing. A similar process will be used for unsupported browsers: we will support browsers contained in 95% of American users and users using unsupported browsers will receive the same notification page provided for unsupported screen sizes.

ER-3: Resource Reliability

If our server cannot communicate with our word validation API but our authentication/database service is still working, we will still fulfill all of our functional requirements except for those needed to play a game of WordScuffle: *FR-1 Construct Words*, *FR-2 Keep Score*, *FR-3 Track Time*, and *FR-4 Validate Words*. Although some of these requirements can be fulfilled separately they all work in conjunction to make a round of WordScuffle possible and so we will not be operating any of these functionalities if our validation API is not working. In this case we will allow all other functionality but prevent users from starting a WordScuffle round. We will notify them when they attempt to start a round that it is temporarily unavailable because of a service failure.

If our server cannot communicate with our authentication/database service but our word validation API is still working, we will only be able to fulfill the functional requirements *FR-1 Constructing Words*, *FR-2 Keeping Score*, *FR-3 Tracking Time*, and *FR-4 Validating Words*. All of these functionalities can be performed with our server and the validation API. *FR-2 Keeping Score* will be truncated in the sense that we will not be able to save scores after a round is completed and so therefore there will be no “state” for any round that a user completes during an outage like this (state would also be affected because our application will not be able to authenticate users either). During an outage like this, we will still display a main page notifying users that WordScuffle is temporarily unavailable. This is because our team has determined that providing an application with negligible, stateless functionality is as bad or worse than notifying users that WordScuffle is temporarily unavailable.

If our server cannot communicate with **both** our authentication/database service and our word validation API, we cannot directly fulfill any of the functional requirements for our application. Therefore, if our server fails to communicate with both external resources, our web

application is effectively unusable. We will display a main page notifying users that WordScuffle is temporarily unavailable.

5. Potential Risks

Throughout the requirements acquisition phase, the team has identified and considered two main risks. The first risk comes from the fact that our chosen method for word validation is an API with a 5000 request limit for free plans. The second risk relates to server security. The team needs to come up with a way to ensure that secured areas within the application remain unbreached by unauthorized users.

5.1 API Request Usage

After conducting thorough research and consideration, our team decided to make use of the WordGameDictionary.com to power our applications word validation functionality. This API is free, reliable, and fast, so it fits in perfectly with the rest of our system architecture. However, there is the fact that this API has a 5000 request limit for free developer accounts. Because of our environmental requirements to keep our solution costs low, this could be an issue for us. It is important to note that the team doesn't suspect we will ever hit our API request limit throughout the scope of this project, but if the project ever scales to a greater user base, there could be increased cost associated with hosting.

The team plans to solve the issue of our 5000 request limit by using some implementation of server-side caching. The team doesn't have all the details of this caching algorithm determined at this point, however we have determined that caching of requests could reduce the number of requests that leave our server every day. An example of such functionality can be seen in the following example.

Suppose two users make a request to the server to validate the same word. When the server gets the request from the first user, the server will check its local cache to see if that request has been made before. Since this is the first time a request for this particular word has come in, the server routes the request to the word validation API. Once it receives the response from the API, the server saves the results of the request in its local cache and sends the results to the requesting user. Now when the second user makes a request to validate this same word, the server will check its local cache, determine that it already has the validation results from this particular word, and send the results of this request back to the user. In this case, only one request was sent to satisfy two validation requests. This is just one example of a caching algorithm that prevent the team from ever hitting our 5000 request limit.

5.2 Server Security

Because our solutions features secured areas such as an administrative console, our team needs to make sure that these areas remain secured to people that should not access them. For instance, we need to make sure that non-administrative users stay out of the administrative console.

By having the user client program running in the user's web browser send the unique user account token to the Node.js server when requesting web pages, the node.js server can verify the user's account type in the firebase database. Once the user account has been verified as having permissions to view the requested web page, the web page data can be sent to the user client. For administrative changes to WordScuffle rules or altering account information for other user accounts, the node.js server will act as a 'middleman' to verify the client has the authorized level of power to make such changes to the database.

6. Project Plan

Throughout November, we have started working with the different technologies that were chosen to implement the game of WordScuffle. Tasks were initiated in an order based on how they interface with each other, starting with Angular.js as the main platform for the game.

When the class schedule is released next semester, we will be able to make a more detailed gantt chart for the rest of the project. At the moment, we have set a few major milestones for our project.

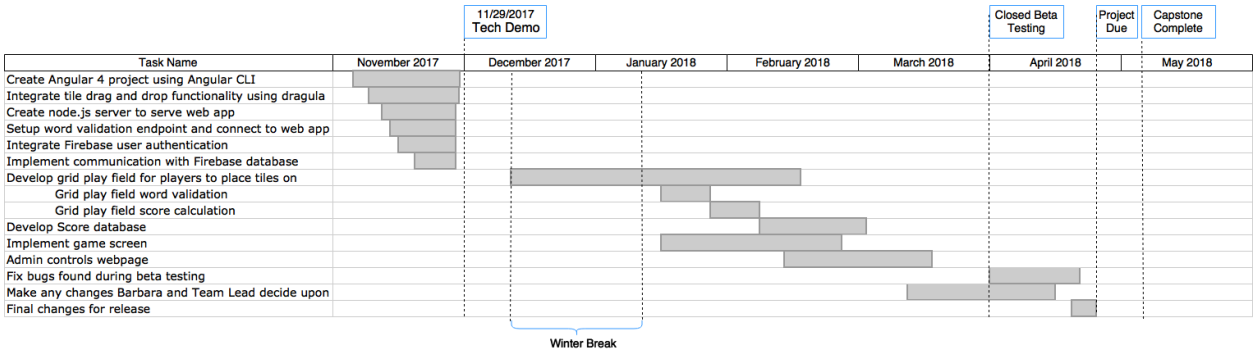


Figure 6.1. Gantt chart for developing Wordscuffle. A larger copy of this gantt chart can be found in the appendix.

6.1 Tech Demo

To prepare for the tech demo, we plan on having all chosen technologies working together to give a general idea of gameplay and confirm the technologies will work together.

6.2 Winter Break

The exact level of productivity for this project over winter break is not certain. Technically, there will be more time to work on the project while classes are not in session, but we may have plans other than working on the project. We will stay in contact with each other over this time, but project productivity will be limited. Any time to work on the project over winter break will focus on the grid playfield.

6.3 Closed Beta Testing

The plan is to get the whole game done by the end of March, so we can release it 'private beta' style to Ms. Jenkins and some of her friends. We will expect feedback from them, and will be able to put all programming efforts into any problems they have or changes they want that fit within our 'contract'. It is important that some of the people in the closed beta testing are unfamiliar with the game, because players experienced with the game rules may have a bias for ease to learn the game and navigate the website.

6.4 Project Due

Near the end of April, the project will be due. Exact date is yet to be discovered. Depending on project progression and actual project due date, closed beta testing may happen sooner than noted.

6.5 Capstone Completion

At the end of the semester, when we have completed the project, we will consolidate a list of accounts and passwords linked to the technologies used for creating WordScuffle. We created a developer email to be used for all needed developer accounts used for creating WordScuffle, so that easy transfer of developer accounts to Barbara's possession is possible. Instructions for any important administration control not implemented into the front-end of the website will be written and given to Ms. Jenkins.

7. Conclusion

In conclusion, our team BrainStim Studios will be developing a web application implementation of WordScuffle, a word game designed to incorporate cognitive and social gameplay to help reduce the symptoms of Alzheimer's Disease. Our web application will also resolve several key workflow problems associated with the current pen and paper implementation of the game on which our web application is based. We have had many discussions with our sponsor Barbara Jenkins to create our set of requirements based off WordScuffle gameplay, expected website workflow, and website administration functions. Our web application improves on the pen-and-paper version of the game by:

- Automating scoring, allowing players to focus on word combination

- Improve social aspects by controlling tileset generations and high-score viewing, as well as implementing an ability for users to form communities
- Providing word validation to lower confusion and eliminate scoring mistakes

Our nine main functional requirements are as follows: constructing words, keeping score, tracking time, validating words, allowing users to view and compare scores, allowing users to join or create a community, allowing users to manage their account, easy to use/access admin console, and user authentication in the form of logging into accounts. These functional requirements are coupled with the non-functional requirements of tileset generation, tile set attributes, word validation attributes, device compatibility, and server reliability. The environmental requirements are that the game will require the users have internet access, the game will only run on devices with certain hardware specifications, and the reliability of our chosen servers is out of our control.

To address our key requirements for our web implementation of WordScuffle, we will be combining Dragula (a Javascript-compatible drag-and-drop library) with a Node.js-served Angular front-end. We will supplement these technologies with a word validation API from WordGameDictionary.com (to get fast validation from the Scrabble$_{TM}$ dictionary) and the database and authentication services offered by Firebase, a mobile development platform backed by Google. We are confident that we will be able to use these technologies to successfully create a web game to help prevent the onset of Alzheimer's Disease.

8. Appendix

8.1 Gantt Chart