JAVA – GRAPHICS ASSIGNMENT REPORT

INTRODUCTION

This report will be written as step-by-step process, which will help to show how I made certain decisions and how the project progressed over time.

BACKGROUND INFORMATION

League of Legends is a popular free to play competitive multiplayer game. Teams of 5 face each other in 20-50 minute matches where each player must choose a 'champion' to play as. Players can compete against each other in a ranked ladder system with the aim of attaining higher rankings. As of April 2017 there are 134 completely unique champions, each with 4 key active abilities/spells and one passive ability.

The game has a very steep learning curve, particularly due to the sheer volume of information a player must learn to play the game optimally. Since there are 134 champions in game, newer players will frequently face champions they have not seen before and will therefore may have no information on what an enemy champions spells may do and how to play around them. At a higher level, players will generally be aware of the major details of all champions in the game, but are unlikely to know the more subtle but still impactful details such as cooldowns (time before the spell can be used again) and damage values of spells.

Riot Games, the developers of League of Legends, have a free and publicly available API for accessing data about the game (https://developer.riotgames.com/api-methods/). The API is split into different methods for accessing different data (see Figure 1). Examples include information about a players account (match history, ranking etc.), information about players in a current ongoing game and all the stats for all champions in the game.

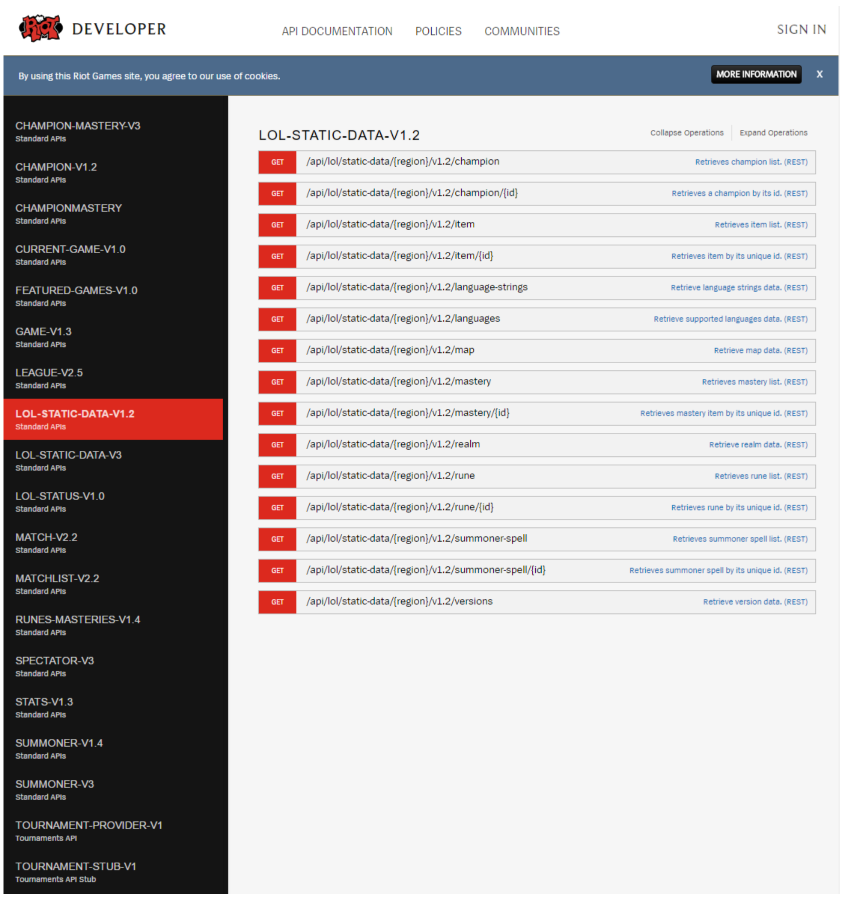
INITIAL PITCH

For this assignment, my plan is to create an application that can access this Riot Games API to retrieve images and data about each champion, then display this in an easy to access format. In this way a player could use this application in the background while in a game to learn about a champion they don't recognise, or to find out the cooldown of a certain enemy spell.

This project will introduce me to several new java techniques. Basic Networking and internet access will be core to the project, and the data is returned from the API in JSON format so I will need to learn how to retrieve and decode this. I am also planning to use FXML and CSS with JavaFX. For my Oxo assignment I found containing all style information within a CSS file kept the code much neater and made styles significantly easier to tweak (the CSS file can be edited without recompiling the Java files). Using FXML to define the tree graph seems like a logical extension of this but for layout instead, as trying to determine which object sits within which container (and so on) is very time-consuming when written in the code.

This should also be a nice project to add features incrementally. I will break the project down into stages, each adding one or several different features. The first stage could simply be a grid of all the champion names downloaded from the API, then perhaps images could be added, the a search feature, then another javaFX scene for viewing more detailing stats about each champion.

In Figure 2 on the following page I show my initial prototype for the layout of the GUI.



DEVELOPER API DOCUMENTATION POLICIES COMMUNITIES SIGN IN

By using this Riot Games site, you agree to our use of cookies. MORE INFORMATION X

CHAMPION-MASTERY-V3 Standard APIs
CHAMPION-V1.2 Standard APIs
CHAMPIONMASTERY Standard APIs
CURRENT-GAME-V1.0 Standard APIs
FEATURED-GAMES-V1.0 Standard APIs
GAME-V1.3 Standard APIs
LEAGUE-V2.5 Standard APIs
LOL-STATIC-DATA-V1.2 Standard APIs
LOL-STATIC-DATA-V3 Standard APIs
LOL-STATUS-V1.0 Standard APIs
MATCH-V2.2 Standard APIs
MATCHLIST-V2.2 Standard APIs
RUNES-MASTERIES-V1.4 Standard APIs
SPECTATOR-V3 Standard APIs
STATS-V1.3 Standard APIs
SUMMONER-V1.4 Standard APIs
SUMMONER-V3 Standard APIs
TOURNAMENT-PROVIDER-V1 Tournaments API
TOURNAMENT-STUB-V1 Tournaments API Stub

LOL-STATIC-DATA-V1.2 Collapse Operations Expand Operations

GET	/api/lol/static-data/{region}/v1.2/champion	Retrieves champion list. (REST)
GET	/api/lol/static-data/{region}/v1.2/champion/{id}	Retrieves a champion by its id. (REST)
GET	/api/lol/static-data/{region}/v1.2/item	Retrieves item list. (REST)
GET	/api/lol/static-data/{region}/v1.2/item/{id}	Retrieves item by its unique id. (REST)
GET	/api/lol/static-data/{region}/v1.2/language-strings	Retrieve language strings data. (REST)
GET	/api/lol/static-data/{region}/v1.2/languages	Retrieve supported languages data. (REST)
GET	/api/lol/static-data/{region}/v1.2/map	Retrieve map data. (REST)
GET	/api/lol/static-data/{region}/v1.2/mastery	Retrieves mastery list. (REST)
GET	/api/lol/static-data/{region}/v1.2/mastery/{id}	Retrieves mastery item by its unique id. (REST)
GET	/api/lol/static-data/{region}/v1.2/realm	Retrieve realm data. (REST)
GET	/api/lol/static-data/{region}/v1.2/rune	Retrieves rune list. (REST)
GET	/api/lol/static-data/{region}/v1.2/rune/{id}	Retrieves rune by its unique id. (REST)
GET	/api/lol/static-data/{region}/v1.2/summoner-spell	Retrieves summoner spell list. (REST)
GET	/api/lol/static-data/{region}/v1.2/summoner-spell/{id}	Retrieves summoner spell by its unique id. (REST)
GET	/api/lol/static-data/{region}/v1.2/versions	Retrieve version data. (REST)

Figure 1 - The RIOT Games API reference website

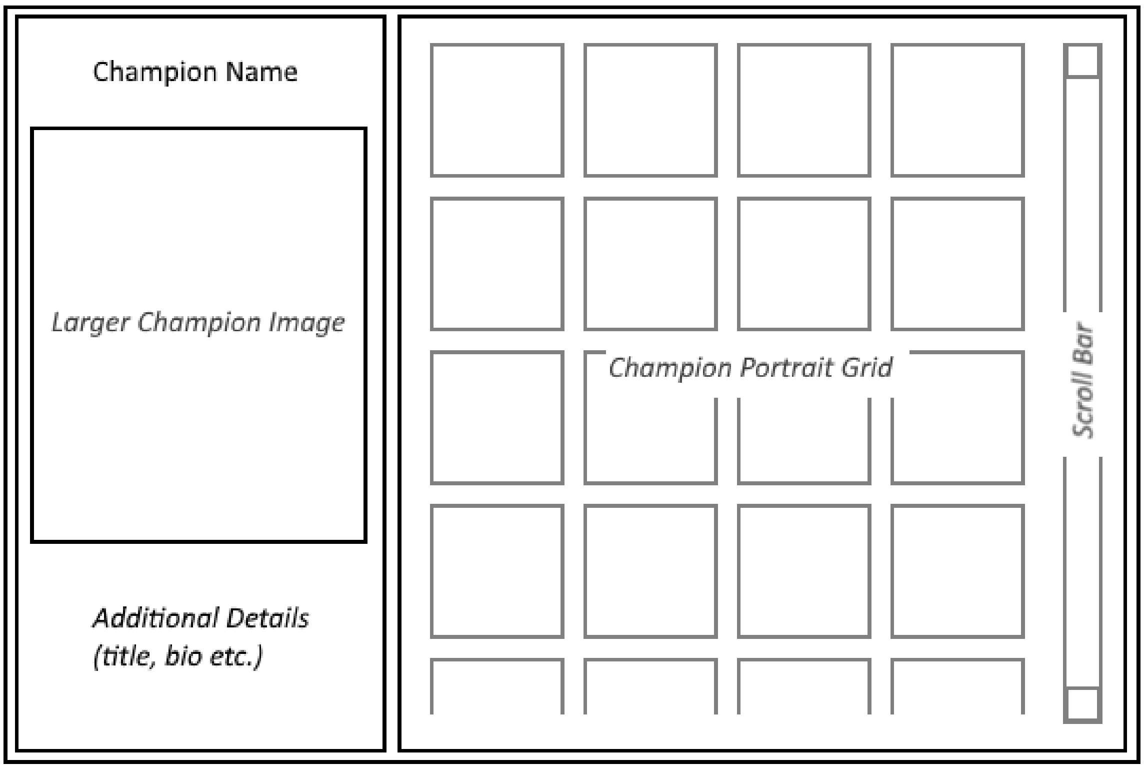


Figure 2 - Prototype GUI layout for the program

FIRST STAGE – SIMPLE JAVAFX WINDOW AND API ACCESS

CURRENT PROGRAM STATE

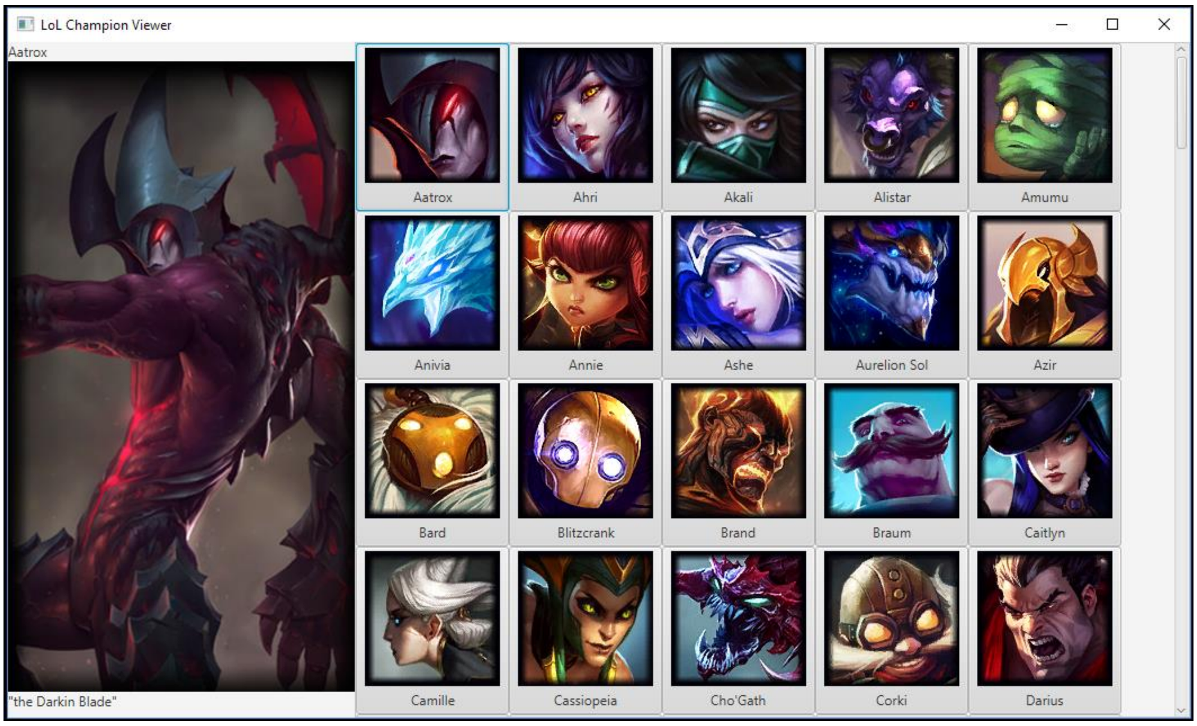


Figure 3 - Program state after stage 1.

- Two classes created, LoLStatViewer & RiotAPIData
- LoLStatViewer is the javaFX class:
    - Sets up all the graphical and interaction elements of the program.
- RiotAPIData is the API access class:
    - Encapsulates all code relevant to accessing the API, including converting the data into a JSON objects.
    - An external JSON parsing library, JSON-java (https://github.com/stleary/JSON-java), is used by this class.
    - All data is accessible to external classes only through getter functions. Each of these convert the JSON object data into standard java types (strings, lists or maps).
    - This class should only go out of date once the API methods or versions are changed. Although League of Legends has regular game patches roughly twice a month, the API keeps itself up to date. When retrieving image URLs, this class calls the API to access the most recent patch version.
    - Since the most time-consuming processes of this program will be API calls, this has been kept to a minimum. This class contains several private variables for storing data that get initialised in the class constructor, so that getters can return information from here rather than making an API call each time.

REMAINING ISSUES

- The program start-up is very slow – this is due to having to access 130+ separate images to display the champion portraits. It will be necessary to include concurrency/threading so that the program displays a loading bar during this process, rather than freezing or not even displaying a window for several seconds.

- Currently the grid of champions is organised alphabetically by key rather than name. While most champion keys are 1:1 with only spaces and punctuation removed, one (and possibly more) bizarre cases exist where the internal name is different. Since the champion “Wukong” is referred to internally as “MonkeyKing”, this champion appears in the incorrect place in the grid. While this is only a very minor issue, it may be worth addressing later.
- For some reason only the portrait image URLs allow version numbers in them. This gives us the strange scenario where a champion portrait displays as the most recent image, but when their portrait is clicked the larger image on the left displays an image from an older version (the champion “Galio” is a good example of this). For now, I will leave this as is but look into a solution later on.
- Currently the UI has no layout settings or styles defined so looks very unpolished. This will be added in a later stage.
- No testing has been implemented yet, this should be added in the following stage.

SECOND STAGE – TASKS, STYLES & TESTING

CURRENT STATE

The current program state is displayed in Figure 4. Not much additional functionality has been added, but several problems from the previous version have been addressed. A button for ability details has been added but does not yet have any action mapped to it. The slow start-up and UI hanging issues were fixed by moving the image loading out of the JavaFX thread into a separate task, the loading screen is displayed in Figure 5. I also spent a significant amount of time working on the styles of the program, creating a 114-line CSS file to deal with all the individual elements. More information is detailed in the following sections.

Another interesting point is that two new champions were released in the game between stage 1 & 2. This demonstrates the main benefit of using APIs rather than an alternative method: my program has updated to display the two new champions (totalling 136) with no additional work required. The only scenario where more work would be required is if the API methods in use were changed or depreciated, which would not likely occur for a significant amount of time and additionally comes with a pre-warning period of several months.

IMPLEMENTATION

- For the image grid task, the grid initially starts as a VBox with the progress bar and some text inside. The progress bar is bound to the 'progressProperty' of the image grid task so it can display the current progress.
    - The 'setOnSucceeded' function for the image grid task updates the borderpane center area to the grid. This allows the grid to be created behind the scenes, then shown only once it has been fully populated.

- Styles for almost every element in the program were added into a CSS file.
    - This includes more basic styles such as setting all VBox alignments to centre, along with more complex changes such as buttons styled with layers of gradients and removing the buttons and arrows from the scrollbar (since they are generally unused).
    - If I was to work on the styles again, it would be worth adding functionality to update the CSS during runtime. While recompilation is not required, restarting the program each time the CSS is slightly edited is fairly time consuming. If it is possible, it would be very useful to simply have a button to reload the CSS file and instantly update with any changes made.

- The issue mentioned in stage 1 where the left panel larger images were out of date was fixed. After asking a question on the API forums, it turns out putting an extra '/' into the URL (example below) returns the image from the latest patch (it is becoming apparent that the API has a few nonsensical quirks that must be discovered and worked around).
    - "img/champion/loading//aatrox_0.jpg" returns the most recent image.

- Testing has been implemented into the RiotAPIData class. LoLStatViewer has not been tested because it is purely JavaFX/graphics, so cannot be auto-tested.

REMAINING ISSUES

- The UI hangs occasionally when clicking on one of the champion buttons, presumably because it must load the large left panel image each time. It will be worth moving the left panel updating into a separate thread in the next section.
- All of the remaining issues from section one have been fixed, except for the out of alphabetical order problem with one or two champion keys.

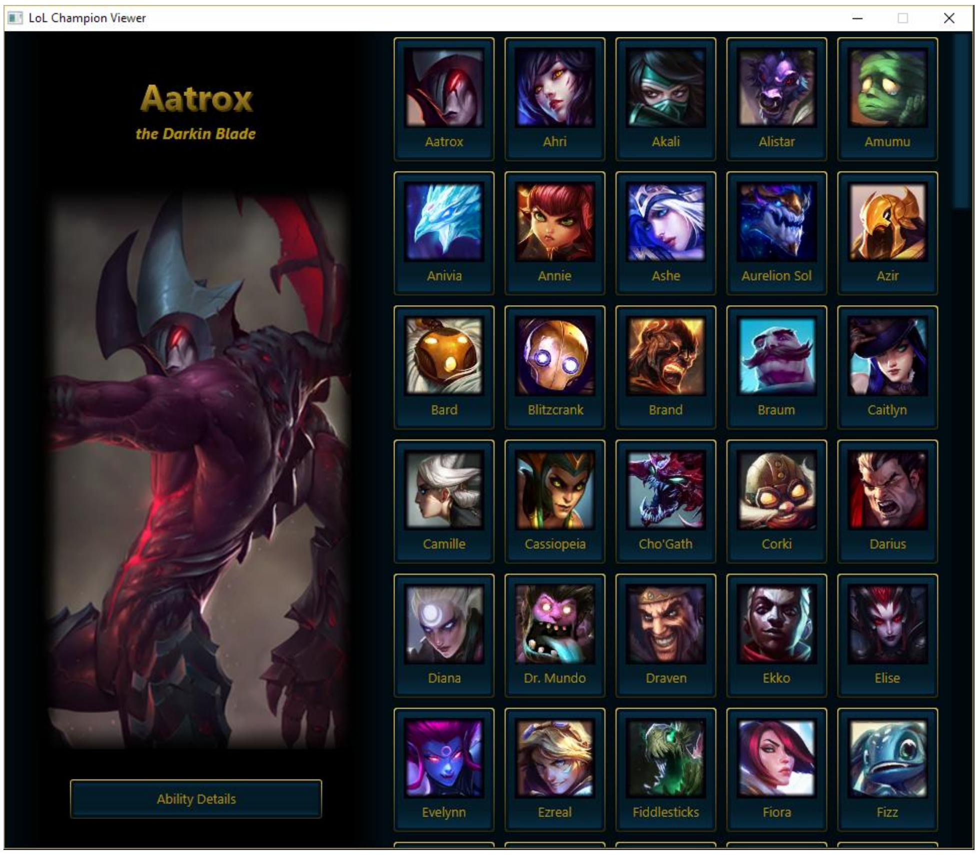


Figure 4 - Program state after stage 2.

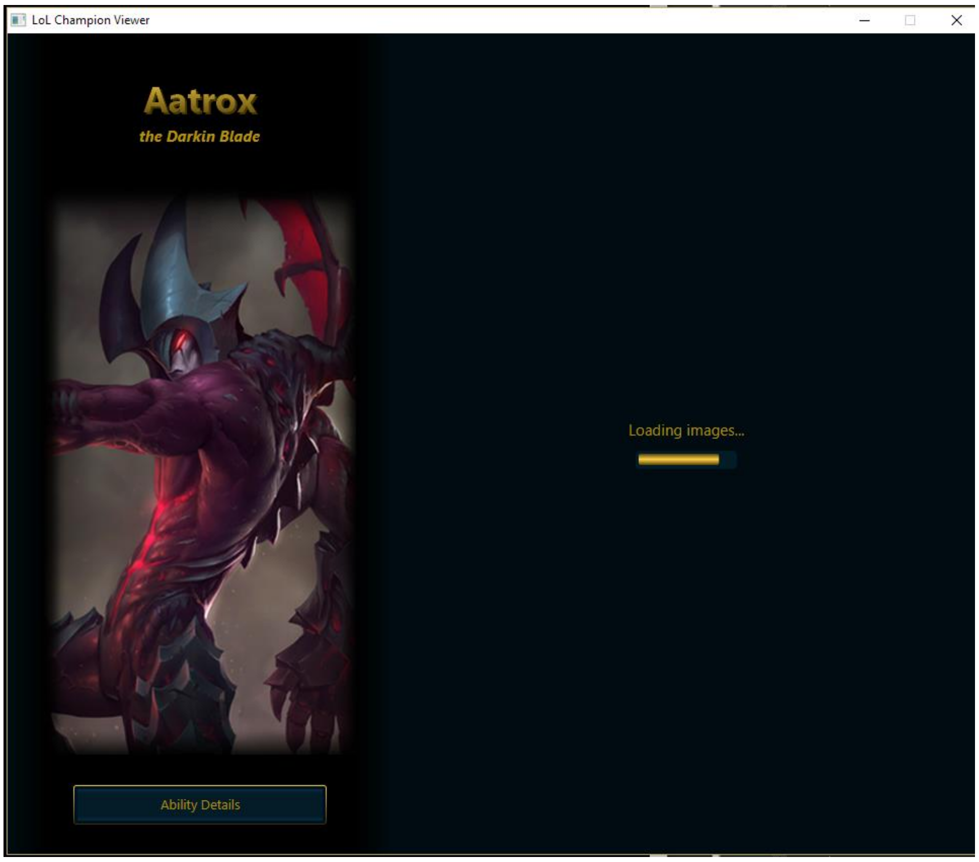


Figure 5 - Program loading the grid images, with a bar to indicate progress.

THIRD STAGE – REFACTORING AND ANIMATIONS

REFACTORING LOLSTATVIEWER

I performed a large refactor of the LoLStatViewer class to try and cut down on the size and make it more readable. As much as possible of the JavaFX scene setup was moved into a new class, SceneSetup.java. All control and modification of the application has been left in the original class. SceneSetup gives public access to LoLStatViewer for important nodes that need to be modified in the control logic, such as buttons and the image and labels in the left panel.

TRANSITION ANIMATIONS

Several animation transitions have been added:

- The first transition was added to fix the UI hang (due to image loading) when switching between champions on the grid. The current image fades out immediately, then a task is set up so when the new image is loaded it will fade into view. This setup should look reasonably smooth regardless of image load time.
  To allow the images to fade in and out at the same time (rather than the current image fading completely, being replaced then fading back in), a new node setup was required for the image in the left panel. Instead of a single ImageView a Group is used which contains two ImageViews stacked on top of each other, one for the previous image fading out and one for the current image fading in.
- The second transition was added to transition between the main view (with the grid and left panel) to the detail view (not yet created). To transition to the detail view, the left panel slides left and the grid slides right, both fading out, followed by the detail view fading into view (shown in Figure 1). The reverse occurs when returning to the main view.

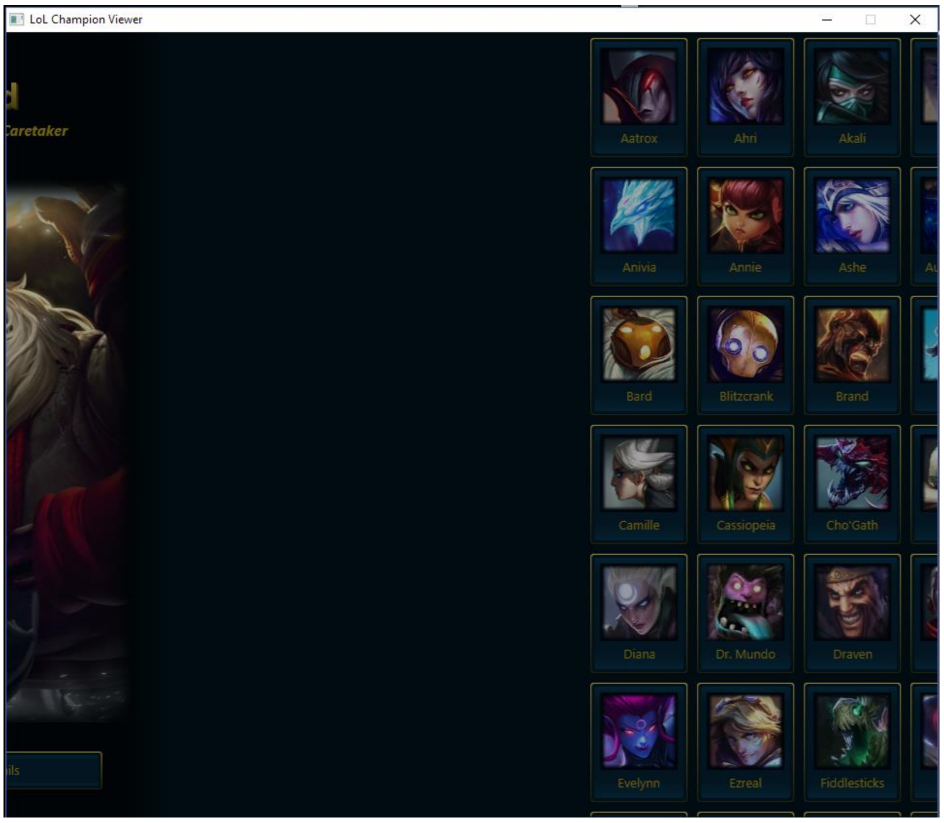


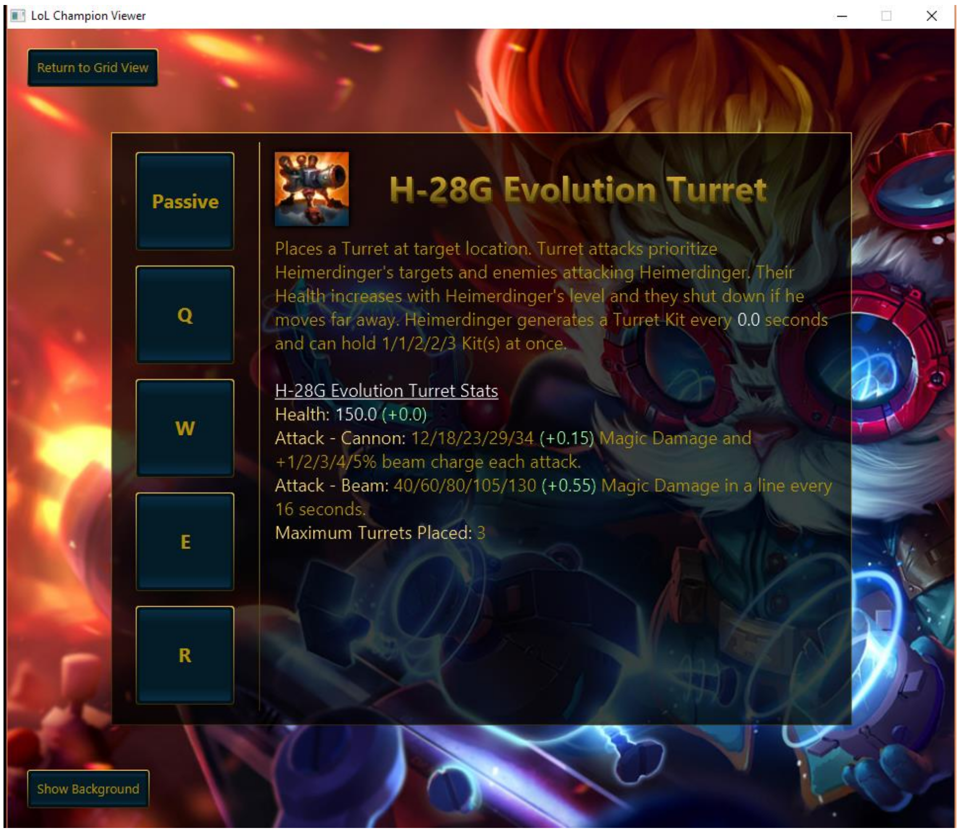
Figure 6 - Transition from the main view into the detailed view.

FINAL STAGE – CREATING THE DETAIL VIEW

FINAL PROGRAM STATE

For this final version of the program the detailed view has been fully implemented, shown in the Figure X, X, X. The user can choose between the 5 abilities using the vertical buttons on the left, which will display the relevant ability title, icon and description in the right panel. The ability description is formatted with different styles in different parts, this process is explained in the following section.

The background uses the champion splash art, which is loaded from an URL in a separate task and fades into view once loaded. The show background button can be used to fade all the other visible nodes so the background can be viewed, and this can be reversed by pressing the same button. This functionality is displayed in



H-28G Evolution Turret

Places a Turret at target location. Turret attacks prioritize Heimerdinger's targets and enemies attacking Heimerdinger. Their Health increases with Heimerdinger's level and they shut down if he moves far away. Heimerdinger generates a Turret Kit every 0.0 seconds and can hold 1/1/2/2/3 Kit(s) at once.

H-28G Evolution Turret Stats
Health: 150.0 (+0.0)
Attack - Cannon: 12/18/23/29/34 (+0.15) Magic Damage and +1/2/3/4/5% beam charge each attack.
Attack - Beam: 40/60/80/105/130 (+0.55) Magic Damage in a line every 16 seconds.
Maximum Turrets Placed: 3

Figure 7 - The detailed view, displayed a formatted ability description.

DESCRIPTION FORMATTING

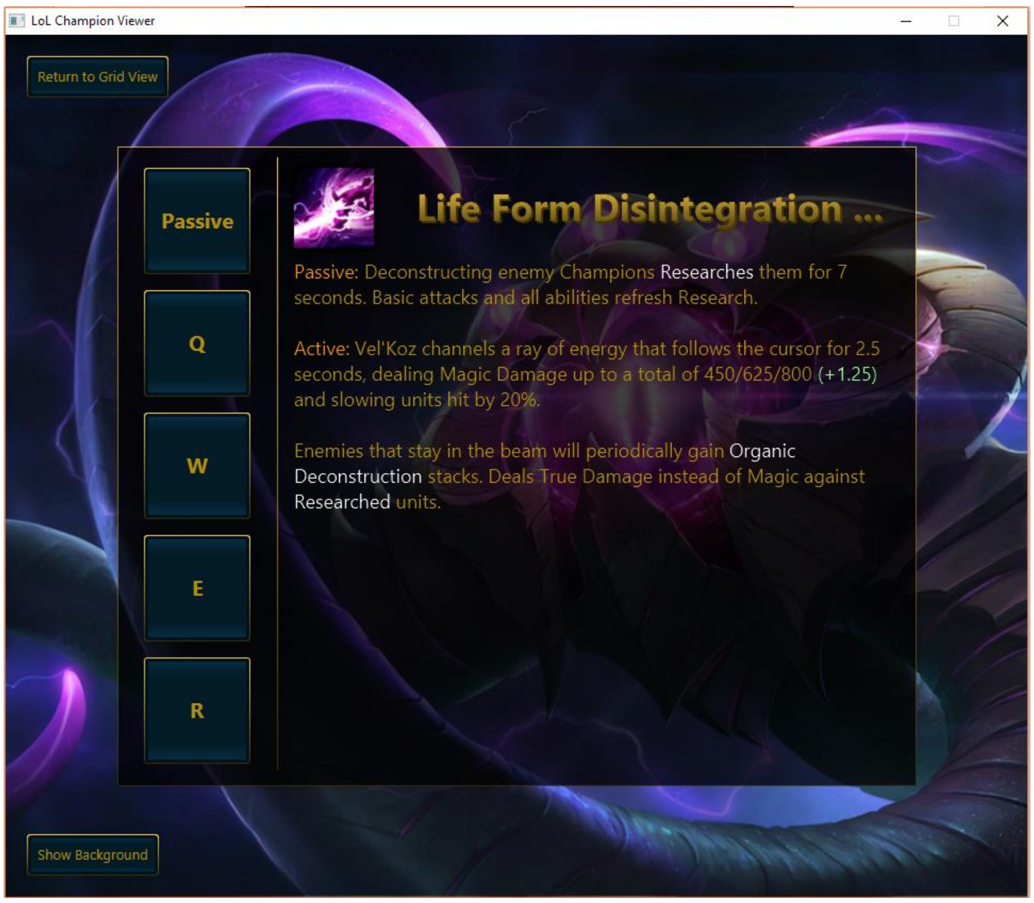
By far the most complex part of this section (and probably the entire assignment) was formatting the ability descriptions supplied by the API. Basic one or two-line descriptions with no formatting or numbers are made available, but I decided to use the more complex 'tooltip' when possible, to provide more detailed information. These detailed tooltips have different styles for individual words and sentences, so a javaFX label cannot be used: a TextFlow must be used, with each differently styled string added as a separate javaFX Text object. A large part of the complexity in formatting derived from the highly inconsistent data provided by the API, which meant a huge number of edge cases had to be catered for. Some examples of inconsistency include:

- Passive abilities do not provide tooltip information. Additionally, a random selection of abilities across different champion shave tooltip information missing.

- HTML tags were invalid in some places, such as close tags before open tags.
- Different styles of HTML tag usage is used: the majority of tooltips use <span color=”#000000”> for colours, but passive abilities use <font color=#000000> instead and a small number of champions have unspecified colour tags such as <abilityPower>.
- The tooltips include placeholders which have keys to match up to specific data values (provided in the same JSON object). Many of the placeholders have missing data values or incorrect numbers.

Since the focus of this program was not a HTML formatting, I chose to write only a simple HTML parser. This doesn’t match close to open tags, so will just assume the next tag after an open is the respective close. The parser functions by splitting the string into array where “<” characters are found, so each new tag has a separate string. The tag data is extracted and stored including hex colour values where specified, then the tag is removed from the string. Additionally, any placeholder text from the API (in the forms {{ a1 }}, {{ e1 }} & {{ f1 }}) are replaced with their respective data values, or simply removed where data is missing. While the execution is fairly simple, the parser is able to parse the majority of tags correctly, and removes all tags it doesn’t recognise, so the end result is always presentable to the user. Additionally, some functionality was included for nested tags (shown in Figure 7, underlined and coloured text). This works when 2 layers of tags are nested directly, such as “<u><font>text</font></u>”.

I created the description formatter as a separate helper class, DescriptionFormatter.java, which is used by the main LoLStatViewer class.



Life Form Disintegration ...

Passive: Deconstructing enemy Champions Researches them for 7 seconds. Basic attacks and all abilities refresh Research.

Active: Vel'Koz channels a ray of energy that follows the cursor for 2.5 seconds, dealing Magic Damage up to a total of 450/625/800 (+1.25) and slowing units hit by 20%.

Enemies that stay in the beam will periodically gain Organic Deconstruction stacks. Deals True Damage instead of Magic against Researched units.

Figure 8 - Another champion ability displayed in the detailed view.

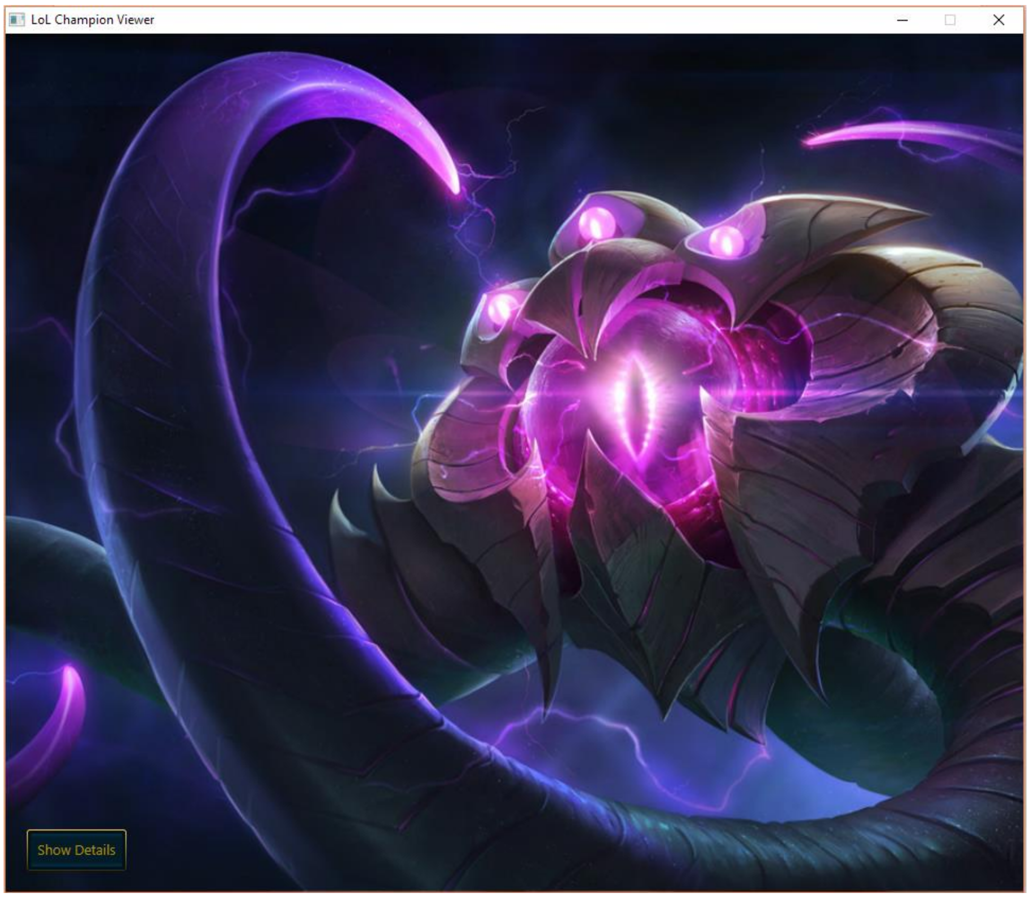


Figure 9 - The show background button has been pressed, revealing the full background image.