Tauresium Test Plan

This document will detail the different tests made on the Tauresium program after the first working build was completed. All examinations in this document fall under one of three types, acceptance, edge case and erroneous, which are defined as the following:

* Acceptance – Input that is to be expected from an average gameplay session.
* Edge Case - Input that lies on the boundary of what is expected to be a reasonable parameter.
* Erroneous – Input that is specifically meant to be unacceptable and handled with an appropriate error response.

Using these three types of testing, this document will review each major feature of Tauresium to ensure the submitted release of the project will not contain any major errors – this by extension includes the API, as all functions on the website make use of the API. For the purposes of this log, the following assumptions will be made:

* The user using a modern browser, ideally chromium based or similar. Any issues that arise from using an outdated or unsupported browser are likely to be client-side only and the responsibility of the user to maintain.
* The user does not modify the source code in any way except to make intentional erroneous inputs or HTTP requests. This means any issues particularly concerning presentation that occur when the original document is modified are not to be examined.
* The user does not have any form of administrative privileges or access to the backend database – this discludes tests that would not be possible in a regular game session – such as the forced deleting of an account when a session is active.
* All inputs are assumed to be sequential, this means tests that concern the timing of inputs will not be considered, such as if two players attempt to take the same location at once. This assumption is applied due to the lack of resources available to conduct said tests.

It should also be noted that all API keys or other identifying data may not be included with the database provided, as these tests were conducted on a different device to ensure the tests did not influence the demo video that was being recorded concurrently, hence if any of these tests are to be attempted, certain parameters may need altering to fit the circumstances of the database in use.

# Test 1 – Page accessibility (Logged Out, Acceptance)

*Description*:In this test, I attempted to access the pages naturally accessible by logged out users, these include 5 pages – The index, the new world creation menu, the country creation menu, the API page, and the tutorial page.

*Result*: All pages listed were successfully accessed using their buttons provided. Pages such as the API page did not include the information not applicable to a logged-out user, notably the API Key. This feature works as intended.

Successful

# Test 2 – Page accessibility (Logged Out, Edge Case)

*Description*: This test concerned the accessing of pages not normally provided to users through direct means such as menu buttons – but should still nonetheless be accessible by logged out users. This includes: The session success page, the login failed page and the error page.

*Result*: Both the login failed page and general error page were accessible through URL entry, as expected. Once accessed, the session success page caused a redirect to the error page, due to no parameters supplied in the URL – an expected result – attempting to provide this page with correct parameters as would be usually applied by the website allowed for the correct loading of the page. This feature works as intended.

Successful

# Test 3 – Page accessibility (Logged Out, Erroneous)

*Description*: This test covers the access of session specific pages – notably the Main page, provinces page, session stats page and two event pages. These pages should cause a redirect to the error page when accessed, as they are only meant to be usable by logged in players.

*Result*: All pages successfully redirected to the error page when accessed, a process which occurs through the backend, meaning the page resources should not have been loaded before the redirect occurred.

Successful

# Test 4 – Page accessibility (General, Erroneous)

*Description*: This test will cover the accessing of files that are not meant to be accessed via the user but exist within the scope of the website and therefore can be loaded. This includes files like the top bars which are embedded within pages when appropriate, as well as scripts normally used by the website.

For the purposes of this test, any errors are only of concern if they cause changes to be made to the database or allow access of unintended resources. As these pages are not meant to be normally viewed on their own, any client side issues that occur and do not persist between pages are the responsibility of the user.

*Result*: When attempting to use scripts, two incidents of note occurred. Firstly, when the login verify script was entered, the page attempted a redirect to an error page within the scope of the Scripts file, which did not exist. Secondly, when the PageUpdate script was accessed, a PHP error occurred when the script attempted to include a file that did not exist – this is a notable issue but one that has no impact on anything but the client accessing the page, and therefore the fixing of this issue is low priority.

When loading the PageElements files, JavaScript issues occurred and resources such as the stylesheet were not correctly loaded, but again this is to be expected when these files are accessed in this way. Similar issues also occurred when loading PageElements files.

In conclusion these issues are not dangerous but fixing these issues should be considered if possible, but this is not a high priority task, due to the nature of how these issues can be accessed.

Low Priority/Minor Issues

# Test 5 – Login (Acceptance)

*Description*: Entered a valid login already stored in the database into the top bar login page as well as the login page accessed when an invalid login is provided.

*Result*: Both pages were able to successfully load the player into their game and store their session variables as expected.

Successful

# Test 6 – Login (Edge Case)

*Description*: Attempted to log into accounts using incorrect case in logins, which is expected to not permit a user to login.

*Result*: All attempts failed to login, entered logins must be in the same case they were originally created in.

Successful

# Test 7 – Login (Erroneous)

*Description*: Attempted to login with no parameters supplied, only a username or password supplied, an invalid username with a valid password and vice versa supplied, and input of a large number of characters, as well as input of non-Latin characters.

*Result*: All logins failed as expected, when a large number of characters were entered the response was slower than other responses, but the login still failed.

Successful

# Test 8 – Create a world (Acceptance)

*Description*:Attempted to create a world with the following parameters:

* World name: Geb
* Map Type: Earth
* Game Speed: Normal

*Result:* The world was successfully generated and the world code “**HHZLQT1FLNLS5MT9**” was provided.

Successful

# Test 9 – Create a world (Edge Case)

*Description*: Attempted to create a world with a duplicate name, Geb, and a game speed of slow, using the map type earth. This should not cause issues, as the unique identifier for the world is the world code, not the name, and the game is set to support all four speeds.

*Result*: The world was generated as expected, providing the world code “**ARQP89DDUA68HF71**”. Checks of the database of this and the previous test showed that both worlds existed in memory.

Successful

# Test 10 – Create a world (Erroneous)

*Description*: Attempted to create a world using numeric characters and with no inputs provided for other properties

*Result*: The page redirected to the world creation screen with three error messages describing the issues that had occurred – the non-alphabetical world name and the lack of inputs for other fields.

Successful

# Test 11 – Create a world (Erroneous 2)

*Description*: Used the developer console to remove restrictions on word length on the client side, as well as allow the selection of invalid properties. The submission was as follows:

* World Name: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
* Map Type: TBA
* Game Speed: e

*Result*: Three errors occurred as expected, one mentioned the invalid name of the world, as it exceeded the character limit, and the other two errors noted the invalid map type and game speed. The latter two errors may only occur if the user modifies the code in this way, as there is a separate message for missing inputs and there are no invalid inputs that can be selected on the page. As with the last test, neither world appeared in the database.

Successful

# Test 12 – Create a country (Acceptance)

*Description*: Created a new country in the world “**HHZLQT1FLNLS5MT9**”, using the following parameters:

* Country Name: England
* Password: 1234
* World Code: HHZLQT1FLNLS5MT9
* Government Type: Democracy
* Colour: Red

*Result*: The country was successfully generated and added to the world. Upon generation, “England” was granted the province “**Europe\_Croatia**” and all starter buildings were constructed. Logging into this account was successful and showed the aforementioned information.

Successful

# Test 15 – Create a country (Edge Case)

*Description*: Attempted to create a new country using the maximum number of characters possible as the nation name, as well as a single space as the password. All other fields were valid.

*Result*: The inputs provided caused the program to redirect to the error page, rather than redirect to the same page with error messages appended. This only occurs when all basic checks have passed but the input provided has caused another error. After retesting this, the error seems to have come from the password. This issue may require fixing, but it should be noted that no modifications were made to the database before the error occurred, meaning the only consequence of this problem was the redirect to the error page rather than to the create a country screen.

Low Priority/Minor Issues

# Test 14 – Create a country (Erroneous)

*Description*: Attempted to create a country without any parameters supplied

*Result*: The user was redirected back to the create a country page with error messages supplied that told them that the information was missing.

Successful

# Test 15 – Create a country (Erroneous 2)

*Description*: Entered invalid characters into the username, password, and world code boxes, with valid government and colour inputs.

*Result:* input of username and world code issues was detected, but invalid passwords were permitted. Like in the former edge case, the country was not appended to the database but did cause a redirect to the general error page, rather than the intended account creation page. This means the password input is liable to errors, which are likely occurring when the password is hashed. This issue may need resolving.

Low Priority/Minor Issues

# Test 16 – Create a country (Erroneous 3)

*Description*: Entered a name in use by a player as the nation name, a world code of a full world, and a colour in use by a player in said world. These inputs should cause issues, as the nation name is intended to be a unique identifier, worlds should not permit new users after reaching their capacity, and colours must be unique in a world.

*Result*: Nation name and world name were rejected, and the error was reported appropriately. Due to failing the world name check, the colour code was not checked, further testing showed that entry of a new country with a duplicate colour was not permitted by the system in a non-full world.

Successful

# Test 17 – Create a country (Erroneous 4)

*Description*: Used developer console to allow inputs that exceed the nation name size limit and the world code size limit, as well as permitting the entry of invalid government types and colours.

*Result*: All inputs were reported as invalid when the player was redirected to the country creation page, including the country colour (as it checks if the colour is valid before checking if it is used within the world code provided). No new data was added to the database as a result of this, meaning the country creation did not succeed, as expected.

Successful

# Test 18 – Page accessibility (Logged In, Acceptance)

*Description*: Players with an active session should be permitted to access certain pages, including all pages accessible to logged out players as well as the main pages, province page and event pages. This test was conducted using both the buttons available on the website, as well as URL inputs.

*Result*: All pages were accessible by the logged in player. It should be noted that URL inputs redirected to the error page when no parameters were supplied for the provinces and event results pages, this result is expected.

Successful

# Test 19 – Page accessibility (Logged In, Edge Case)

*Description*: Logged in players may be able to access pages originally intended for logged out players – there should be no issues that arise from these pages being accessed, however it is worth testing to be sure.

*Result*: When accessed using a logged in account, no issues arose from loading the pages, including the “login failed” page, which is not usually accessible under normal circumstances, but does not present any problems. Attempting to use the login failed page to login to a second account while previously logged in does not seem to cause any issues, as the session variables used to store the players state are all overwritten when the login script is accessed, meaning the session is treated the same as if it was logged out and back in.

Successful

# Test 20 – Page accessibility (Logged In, Erroneous)

*Description*: While there are no pages that a logged in player cannot access, this does not account for events that may occur when a page is loaded but the player is logged out through external means (For the case of this test, by opening a second tab and logging out). This test will attempt to access certain high-risk scripts or pages while the session is unloaded in a separate tab, the subjects being: Events, annexation, and building.

*Result*: When attempting to answer an event while the session is removed, the event is not completed, and the user is redirected to an error page. Similar events occur when an annexation is attempted, both for the cases of peaceful annexation (culture and economic) as well as military annexation. Interestingly, the same is not true for buildings. When a building request is made while the session does not exist, the building is constructed before the player is redirected to an error page, with the appropriate balance still being deducted from the player and the building bonuses applied – it even prevents building from occurring when the player does not have the required influence. This is likely because the session variable API key is being loaded when the provinces page is being accessed – so the AJAX call is using the stored value for the API key, which allows it to conduct the request while the session does not exist server side any longer. This issue is minor at best, as it does not get in the way of players interacting with the game, nor does it allow for players to exploit the game in any way, therefore it is likely this issue will not need to be fixed.

Low Priority/Minor Issues

# Test 21 – Event generation

*Description*: This test has no type as it is a feature outside the control of the user. Event stacks are generated based on world speed and use the last online time to calculate the event stacks that should be added, storing a decimal value equal to the amount of time passed divided by the event speed – this means if 45 minutes pass for a 30-minute event speed game, the events should be incremented by 1.5 event stacks. These stacks are limited to 5 at any time and cannot be less than 0. To test this, I reset the event stacks to 4 and let a minute pass on a 30-minute event timer, which should result in an increase of 0.03333 event stacks.

*Result*: The event stacks were set to 4.03278, which, an increase of 0.03278. This is almost exactly 0.03333, with the error being only 0.00055 stacks missing due to the decimal limit of 5 decimal places – which is a small enough margin of error to not be impactful to a user’s experience. This feature seems to work as intended with minimal issues, and even works while the server is offline – as the database stores the last online time of a player and compares that value against its own time.

Successful

# Test 22 – Event loading (Acceptance)

*Description*: For this test I will attempt to load a new event from memory using an event stack, if this functions correctly it should remove an event stack and load a random event into the game, with three relevant options.

*Result*: The game decreased the players event stacks by 1 and loaded the event ‘Deicide’ with its three relevant options. Leaving this page and opening the event page again loaded the same event without decreasing any stacks, demonstrating that once an event is loaded into memory the player can open it again at any time.

It should be noted that with event loading, there is instances where the developer console will receive a failed GET request. This is an intended feature, the webserver first sends a GET request to the server to see if an event exists, with a failed request signifying a new event needs to be loaded by using a PUT request. The response code for this operation is an error response code, and thus some consoles will report it as an error, despite it being an intended feature.

Successful

# Test 23 – Event loading (Acceptance 2)

*Description*: For this test I attempted to load an event from the button that appears on the event completed screen (which only is visible if the player has 1 or more event stacks left), to ensure that this button also properly functions.

*Result*: A new event ‘Munch Squad’ was loaded for the player, leaving this page and re-entering it kept the event as with the first event load test. Attempting this test again but without pressing the ‘next event’ button allowed the player to load a new event by pressing the event button on the top bar, as is expected.

Successful

# Test 24 – Event loading (Edge Case)

*Description*: This test concerns the case in which a player attempts to load an event when they have exactly one event stack, the bare minimum amount that a player can have before an event can be loaded.

*Result*: The event ‘Deicide’ was successfully loaded despite the player having just barely enough stacks to load the event – demonstrating that the event loading system does not have any issues when dealing with exact values.

Successful

# Test 25 – Event loading (Edge Case 2)

*Description*: This test examined the case in which a player exits the event page after using their last event stack to load an event – giving them 0/5 event stacks. If the website functions as intended, the player should still be able to access the event page to resume their previously loaded event.

*Result*: When exiting the page, the top bar event button was still available despite the lack of event stacks. When pressed, the player was allowed to resume their event as normal, after answering the event button was removed as the player now had no event stacks remaining nor any event loaded.

Successful

# Test 26 – Event loading (Erroneous)

*Description*: For this test, I used up all the event stacks loaded by the player before attempting to load a new event either through any buttons available or the load event URL.

*Result*: Once the final event was completed, the button allowing a player to use their next event did not appear, and neither did the top bar event button. Manually entering the load event URL caused a redirect to the error page and did not load an event for the player nor use any event stacks, an expected effect when a player attempts to cheat the system by loading an event without having the necessary events.

Successful

# Test 27 – Event answering (Acceptance)

*Description*: For this test I loaded the event ‘A New Form of Warfare’ with an account set to 1 in all focus point generation categories as well as 0 influence for said categories. I then selected the answer ‘Modernising the nation is certainly important, but the costs outweigh the benefits’, which is an option set to grant the player +0 culture generation, +0.03 economic generation and -0.03 military generation. Additionally, as the event is worth 60 influence, the player is expected to have the following statistics on selection of the aforementioned option:

* Culture Generation: 1
* Economic Generation: 1.03
* Military Generation: 0.97
* Culture Influence: 60
* Economic Influence: 61
* Military Influence: 58

*Result*: When answered, the player received the exact bonuses specified by the description, exactly as expected. Additionally, it should be noted that since the influence are stored as integers, new influence is always rounded down to the nearest integer, hence why economic influence (which would normally receive 61.8 influence) only received 61 points.

Successful

# Test 28 – Event answering (Acceptance 2)

*Description*: This test took place immediately after the last and concerns the second event loaded after a player has received their rewards, and if the rewards received are added accurately. The next loaded event was ‘Creature Comforts’ and the selected option was ‘Cull the population to acceptable levels’, which is worth -0.01 culture focus, +0.03 economic focus and +0 military focus, with the event itself being worth 25 influence. Adding these stats to the previously attained values, the player should have the following:

* Culture Generation: 0.99
* Economic Generation: 1.06
* Military Generation: 0.97
* Culture Influence: 84
* Economic Influence: 87
* Military Influence: 82

*Result*: All values change as expected, demonstrating that the bonuses provided by events do change over time, and previous changes do impact new point gains – if the original 1.03 economic generation were applied, the player would only have received 25 economic points from this event, leaving their total at 86 points, hence proving the new additions to the influence count are influenced by the new generation levels.

Successful

# Test 29 – Event answering (Erroneous)

*Description*: For this test, I loaded the event ‘Rosebud’ and edited the first options tag in the developer console to rename it as ‘Option 4’, then I attempted to press the button to see what would occur as a result of this.

*Result*: Pressing the button with an invalid name redirected the player to the error page as intended. When resuming the game, I attempted to load the event again to see if the event had been answered regardless, but the event was still loaded into memory, and no influence or generation changes had been applied, leading me to believe the issue was caught before any errors could occur.

Successful

# Test 30 – Culture annexation (Acceptance)

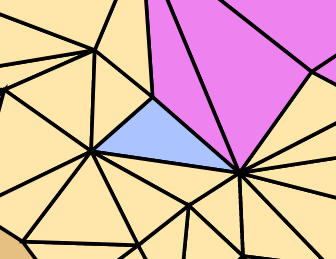
*Description*: Attempted to annex a location adjacent to a province owned by a player using culture influence. The location used for this was ‘**China\_SouthTibet**’, which had an adjacency with the player in question. This location was stated to cost 57 culture influence, and the player had a balance of 84 influence, thus meaning they should have 27 influence upon annexation.

Figure 1 China\_SouthTibet

*Result*: The location was successfully annexed by the player, adding the location to their nation. After annexation occurred, the players influence balance was set to 27, demonstrating that the correct amount of influence had been deducted.

Successful

# Test 31 – Culture annexation (Edge Case)

*Description*: Attempted to annex the location ‘**China\_WestTibet**’, a location worth 75 influence points, while the player attempting the annexation had exactly 75 cultural influence. This should allow the annexation to occur while leaving the player with a balance of 0 culture influence.

*Result*: The location was successfully annexed, and the cultural balance of the player hit 0 as expected.

Successful

# Test 32 – Culture annexation (Erroneous)

*Description*: Sent a cultural annexation request to the province ‘**China\_EastXinjiang**’ while having an insufficient balance to claim the province. The player should not be able to take this land, as they do not have sufficient influence to do so.

*Result*: On accessing the provinces page for the region, the button to attempt culture annexation was greyed out, with click events turned off for the button. I used the developer console to reenable the click events and attempt to send the annexation HTTP request, this resulted in a redirect to the error page as the script had detected the issue, and on accessing the map page, the land was not taken nor had there been any changes to the culture influence balance.

Successful

# Test 33 – Culture annexation (Erroneous 2)

Figure 2 SEAsia\_BurmaNorth

*Description*: For this test I attempted to annex the location ‘**SEAsia\_BurmaNorth**’ while having 999 cultural influence, more than enough to take any province on the map, but without any adjacencies to the location. This should pass the check for costs, but not the adjacency check, and therefore the annexation request should be denied.

*Result*: Much like the last test, as the location was not accessible, the button for the province was greyed out, preventing the player from sending an annexation request without changing the tag properties in the developer console. When this was reenabled, I sent an annexation request which resulted in a redirect to the error page, as the web server had noted that the location was not adjacent to any province.

Successful

# Test 34 – Culture annexation (Erroneous 3)

Figure 3 Occupied China\_EastGuangdong

*Description*: In this test I granted the location ‘**China\_EastGuangdong**’ to another player within the session, and attempted a cultural annexation of the location, this should not be possible, as only military annexation is permitted to take other players owned locations.

*Result*: As usual, the button for cultural annexation was not available due to the province being owned, but by editing the HTML in the developer console I was able to send a request to take the location, which once again ended in a redirect to the error page. It should be noted that the annexation button used is only forming a HTTP request to be sent to the API, hence the results for these requests would be the same when done through the API.

Successful

# Test 35 – Economic annexation (Acceptance)

*Description*: This test involved the use of economic points to annex locations in a local region, for this I used the previously mentioned location ‘**SEAsia\_BurmaNorth**’ which, while not adjacent to a province, exists in a coastal region the player has access to. This should mean economic influence can be expended to take the location, as the player has enough economic influence to do so.

*Result*: When accessing the province page, the economic button was available, and could be pressed to send an annexation request for 96 economic points, out of the players 100 economic points. When pressed, the player was granted the location and retained 4 economic influence as expected.

Successful

# Test 36 – Economic annexation (Acceptance 2)

*Description*: For this test, I granted a player a number of coastal locations in the Indian ocean region, allowing them to become dominant in the region. This meant the player should now be able to economically annex locations in the adjacent region of the East African coast (‘**Africa\_Somalia**’)**.**

*Result*: The button to take the location was available as expected, and annexing the location proved successful. Additionally, as expected, the cost of the location was increased by the regions ‘far away’ penalty, which adds an additional cost to sending economic annexation requests to far away locations. Finally, after annexing this location, the player now notably had 1 out of 4 coastal provinces needed for dominance in the East Africa region.

Successful

# Test 37 – Economic annexation (Edge Case)

*Description*: For this test, I attempted to take a local coastal region while having exactly the required amount of economic influence, if successful, the player should be able to take the location while retaining only 0 economic influence. For this test I used the location ‘**Madagascar\_Boeny**’, which has a cost of 76 economic influence.

*Result*: The player was able to take the location and was left with 0 economic influence as expected, similar to the results of the edge case cultural annexation.

Successful

# Test 38 – Economic annexation (Erroneous)

*Description*: In this test I attempted to send an annexation request to a coastal region while having insufficient economic influence to do so, much like the related cultural annexation tests, I used the developer console to allow me to create this request and the requests for all future erroneous economic annexation requests.

*Result*: Attempting to annex the location resulted in a redirect to the error page as expected, and the location was not taken nor was any economic influence deducted from the player’s balance.

Successful

# Test 39 – Economic annexation (Erroneous 2)

*Description*: This test attempted to annex an inland location using economic points, which should not be possible. To ensure this test is not biased by coastal region or point cost, the location chosen, ‘**Africa\_Zimbabwe**’ was a location within the players coastal region, and said players economic balance was set to 999, higher than any province should ever cost.

*Result*: As expected the annexation request led to an error page response and did not result in any changes to the map or point balance, demonstrating that players may not annex non-coastal locations using economic influence

Successful

# Test 40 – Economic annexation (Erroneous 3)

*Description*: This test concerned the sending of economic annexation requests to far away locations when the player does not have dominance in an adjacent coastal region. For this test I granted the subject player coastal land in the west Africa node, granting them 1 of 5 locations needed to travel to the adjacent South America East or Mexico node. This should not allow them to use economic influence to travel to this location.

*Result*: The player was unable to annex the location, as before, changing properties in the developer console to submit the request caused the error page to be displayed and no changes to be made, preventing the player from circumventing the game rules to their own advantage.

Successful

# Test 41 – Economic annexation (Erroneous 4)

*Description*: In this test I attempted to annex an enemy occupied location using economic points which, like cultural annexation, should not be possible – only military influence can be used to take other players provinces. This test used the previously mentioned location of ‘**China\_EastGuangdong**’, as was tested for cultural occupied annexation.

*Result*: As with similar tests, the player could not take the province, with attempts through console causing a redirect to the error page. This should mean that all possible avenues for exploits with economic annexation are covered.

Successful

# Test 42 – Military annexation (Acceptance)

*Description*: This test will concern the use of military annexation to annex an adjacent, unoccupied location. As military annexation can be done both through fulfilling the cultural annexation requirements or the economic annexation requirements, this test should demonstrate that an adjacent province can be taken using military points.

*Result*: Taking the location ‘**China\_EastXinjiang**’ using military points was permitted, due to the adjacency of the location to existing borders. This came at an additional +20% cost due to the lack of occupation in the region, and -5% due to adjacency. This meant the total cost came to 99 military influence, which the player was able to use to annex the location, and which was subsequently removed from their balance.

Successful

# Test 43 – Military annexation (Acceptance 2)

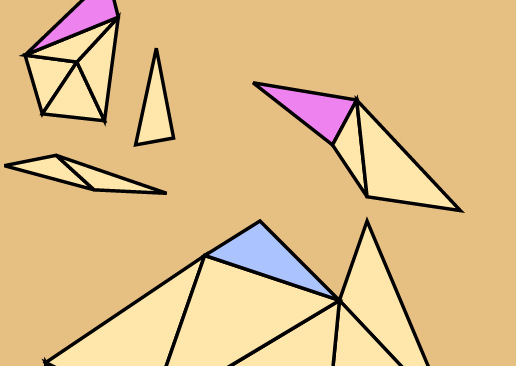
*Description*: As previously mentioned, military annexation is expected to work either through adjacency or sea routes, hence this test will ensure military points can be used to annex locations within a local region (a region that has at least one coastal city within the area). For this test I used the location ‘**Australia\_NorthernAustraliaNorth**’, which is present within the south east Asian region, of which the subject player has a coastal region within.

Figure 4 Australia\_NorthernAustraliaNorth

*Result*: The annexation request was successful, costing 117 military power for the player and granting them the location in exchange for that influence cost.

Successful

# Test 44 – Military annexation (Acceptance 3)

*Description*: This test will determine if the player can use military power to annex far away regions - locations which are adjacent to a location a player has dominance in, but which said player does not have any coastal provinces within. To this I granted the player 5 provinces in the West Africa region, which grants them coastal dominance in said region, and allows them to use their influence to annex the South East America coastal region.

*Result*: The player was able to annex a location within the South East America coastal region, costing an enormous 226 military power to do so. This is because using military power to take an province on a far away location costs an additional +100% military influence.

Successful

# Test 45 – Military annexation (Acceptance 4)

*Description*: In this test, the player will attempt to annex an adjacent occupied location using military power – this should be permitted, as players can take occupied locations using military influence. To do this I will be using the ‘**China\_EastGuangdong**’ previously used to test if a player could annex an occupied location using cultural influence or economic influence.

*Result*: The player was able to finally annex the location, taking it from its previous occupant after expending the necessary points to do so. This demonstrates military annexation Is possible for adjacent provinces.

Successful

# Test 46 – Military annexation (Acceptance 5)

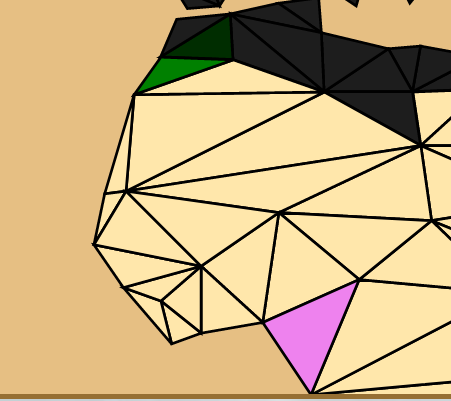
*Description*: This test concerns occupied military annexation within local regions, specifically within the West African region. To do this, I attempted to annex the location ‘**Africa\_WestSaharaState**’ which is present within the region, under the occupation of green.

Figure 5 Africa\_WestSaharaState

*Result*: The province was able to be annexed as expected, granting the player the land, and deducting the appropriate military influence, with a 20% bonus cost for not having adjacencies.

Successful

# Test 47 – Military annexation (Acceptance 6)

*Description*: This test will verify that the player is able to use military influence to annex occupied locations in far away regions, for the purposes of this test, I gave the green nation the previously occupied location in south America, making the location once again considered a ‘far away’ region for the pink country. This test should allow pink to take the location with an added cost.

*Result*: The pink country was able to annex the aforementioned location successfully, as usual with the correct province granted and the appropriate cost removed from the players military balance, with the previous mentioned +100% cost for far away regions being applied again.

Successful

# Test 48 – Military annexation (Edge Case)

*Description*: Like the other annexation edge case tests this test will examine if the player is able to annex a province while having the exact number of points required for said action. For this test, I granted a nation all but one province in the Indian subcontinent and used the last province (owned by another nation) as a point to attempt to invade from.

*Result*: Interestingly, this seems to be the sole annexation case in which the edge has an issue. When attempting to annex a province worth 84 military influence, the button to attempt annexation was unavailable, meaning at the very least the front end is not correctly handling exact values for military annexation.

Additionally, when I granted the player one more military point, they were able to annex the location and were left with 1 military point left, meaning the actual cost deducted is correctly displayed. However, when the front end is circumvented using the developer console to reenable annexation buttons, the annexation request still fails, redirecting to the error page as if the request were invalid.

Gameplay Issue

# Test 49 – Military annexation (Erroneous)

*Description*: This test will determine if military influence can be used to acquire provinces when the player has insufficient points. As with other tests of this type, this will be done by editing the HTML to allow the clicking of the annexation button, to send the request – this same effect can be done by manually submitting a POST request.

*Result*: Attempting to militarily annex a location with insufficient points will cause a redirect to the error page as expected, and will not permit the annexation to occur, nor will points be deducted.

Successful

# Test 50 – Military annexation (Erroneous 2)

*Description*: This test will concern the use of military influence to attempt to annex far away coastal regions without first having sufficient coastal power in an adjacent region. To do this, I will attempt a military annexation of the province ‘**SouthAmerica\_NorthChile**’ which is not in an adjacent coastal region to any provinces owned by the subject nation. This annexation attempt should fail.

*Result*: Once again using developer console to reenable pointer events, sending the annexation request to take a province in a region a player has no access towards causes a redirect to the error page, with no provinces being taken nor any points lost. Additionally, this test was repeated on both an owned and unowned province to ensure there are no differences, both of which had the same result.

Successful

# Test 51 – Military annexation (Erroneous 3)

*Description*: In this test, I will use the API directly to attempt an annexation request on a province owned by the player making the request. This is not possible in normal circumstances on the website, as the annexation page is replaced with the building page when it detects it is owned by the connecting player.

*Result*: The server responded with the default response for an invalid annexation request, indicating it detected the player was attempting to annex their own province. While the error message could be changed to reflect this, since it is not is possible in normal gameplay, it is not a priority to add this. However, since the annexation request was not successful, no provinces changed nor did any point balances.

Successful

# Test 52 – Military capacity (Acceptance)

*Description*: This test will attempt to exceed the military influence capacity of a player using events, to ensure the player is not able to get a balance over their limit. To do this their military influence will be set to one below the maximum and then an event will be completed, this player should then only reach their capacity.

*Result*: The event attempted to give the player an additional 46 influence points but only added enough to hit the capacity, hence meaning the capacity is enforced by the web server. However, the event screen does still say it added the original amount of military influence, rather than the amount adjusted for the capacity – while this is a small issue it could be worthwhile to modify this to reflect the real number of points added.

Successful

# Test 53 – Province page loading (Erroneous)

*Description*: The province page requires the name of a province to be passed to load information, this is set as a variable within the URL, and therefore the name can be easily modified by a player. This test will be used to examine the cases in which the province URL is modified to include no parameters as well as invalid province names.

*Result*: Both tests resulted in a redirect to the error page, as is expected. However, it is interesting to note that the redirect using no parameters was instant, whereas the false parameter test redirect was not. This is because the GET request for the URL parameters is done through PHP, whereas the search for the province name is handled using AJAX, hence the request for a redirect is conducted shortly after the loading of the page – this does not seem to cause any issues.

Successful

# Test 54 – Main map visibility

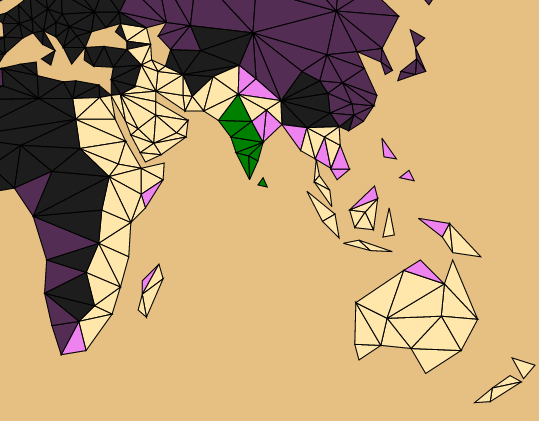
*Description*: This test concerns province visibility and texturing on the main page – all owned provinces should be highlighted in their respective national colour, and provinces that a nation is not adjacent to, nor has any coastal adjacency or coastal provinces within, should be greyed out to signify the lack of importance to the player. This test will determine if the view system is correctly implemented by limiting a certain players provinces to the India region and examining the view available to said player – if the functionality works as intended, this player should gain visibility of the east African region, the south east Asian region and any adjacent provinces.

Figure 6 The visibility provided to green.

*Result*: Once restricted to india, the green player was only able to view regions relevant to them, including the east african, indian and south east asian regions. Additionally, as can be seen at the top of the nation on the supplied figure, the player was able to view some adjacent locations that would not be normally visible if visibiltiy was restricted to solely coastal regions.

Successful

# Test 55 – Main map province costs

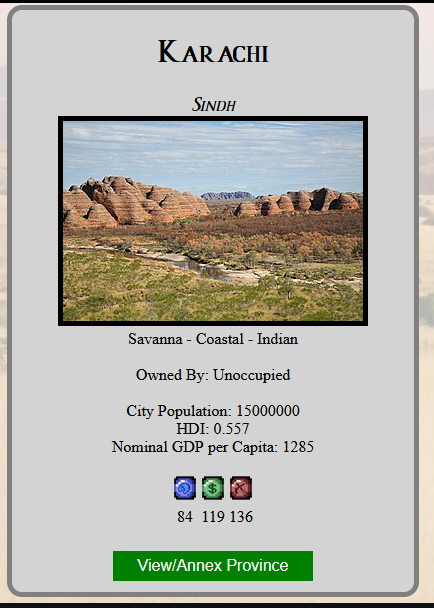
*Description*: When selecting a province on the map, an AJAX request is made to permit the player to read the province cost before entering the province page, this request should return accurate results, which include any modifiers that would be applied, such as military modifiers etc. This test will ensure the values stated on the short province view screen are accurate.

Figure 7 The shorthand province view window

*Result*: The values of a selected province are identical to that provided within the province view page, and even are not displayed when the province selected cannot be taken via that method. It should be noted that this system has a flaw in that any province not annexable by the player displays the message “inaccessible”, even for their own province, potentially confusing newer players.

Successful

# Test 56 – Construction (Acceptance)

*Description*: Building a province in a location owned by a player allows players to increase certain statistics and is highly beneficial to the late game operation of a session. Player should be able to construct buildings in their owned locations provided they have sufficient points and have previously constructed prerequisite buildings. This test will determine if the player is able to complete the action of constructing infrastructure in their own province as intended.

*Result*: The province first selected for this task was the location ‘**India\_Rajasthan**’, which is a cultural province, and hence has access to both culture and military buildings. On testing, the player was successfully able to construct buildings of both types, provided they had the valid points required. This test was then repeated on the location ‘**India\_SriLanka**’, a military province with access to both military and economic buildings. Tests here once again permitted the construction of buildings in this location as expected.

Successful

# Test 57 – Construction (Edge Case)

*Description*: This test will be used to determine if constructions can be completed when the player has the exact number of required points for a construction to be made, as the aforementioned similar issue with military has brought into question if this problem exists elsewhere within the game.

*Result*: All tests on each point type returned successful, with the player able to construct the buildings when they had exactly the correct number of points, and the post-deduction point balance for all three categories being 0 influence.

Successful

# Test 58 – Construction (Erroneous 1)

*Description*: If a player has an insufficient balance of influence to construct a building, they should not be able to complete the construction, nor should any influence be deducted. This test will attempt construction of buildings when the player has an invalid number of points available.

*Result*: Any construction attempts made with insufficient influence balance caused the page to be reloaded with an error message telling the player their point balance is insufficient to complete the transaction – after this, no points are deducted. This test was repeated with all three-point types to ensure the result remained the same, and all tests resulted in the player being unable to construct the buildings.

Successful

# Test 59 – Construction (Erroneous 2)

*Description*: The API for this website grants players the ability to submit construction requests manually, whereas through the website the HTTP requests are constructed automatically. This means while the result for the point balance will be the same for both, there is the possibility to create illegal requests through the API – for this test, I will attempt to construct a building using the API that is of an invalid type – as in a cultural province, you may not construct economic buildings, this test will attempt to send an economic build request to a culture province using the API. To do this, the following request (with type POST) was sent:

***http://127.0.0.1/Tauresium/TaurAPI/Building/India\_Rajasthan/4M2UU5PQEVFLF14N/E***

*Result*: The request returned an error code 400 with the appended message “Invalid building”, specifying that the supplied building type (E for economic) was not a possible building type within the location.

Successful

# Test 60 – Construction (Erroneous 3)

*Description*: In previous builds of the program, an unseen error existed in which the player was able to construct buildings that did not exist by sending an API build request to a province with the supplied influence type having all possible buildings completed, this did not cause any bad effects to the program, as the invalid buildings had no properties or cost, but was still fixed to avoid any other related issues occurring. This test will ensure the measures provided were sufficient to prevent this error, to do this I will max out the buildings available in a location, before sending a request to construct a new building using the following request:

***http://127.0.0.1/Tauresium/TaurAPI/Building/India\_Rajasthan/4M2UU5PQEVFLF14N/C***

This request will attempt to construct a building in the province that should not exist.

*Result*: The server responded again with a “400 Invalid building” error, demonstrating that building could not be constructed, as is expected, additionally, checking the database ensured that the stored building parameters were not invalid, and had stayed at its correct ID – C4. This test was then repeated for all three influence types, all of which had the same result.

Successful

# Test 61 – Construction (Erroneous 4)

*Description*: Due to building POST requests being user specific, an API key is required, signifying the player who is attempting the request. In this test, I will attempt to create a build request using the API key of another player to a province that they do not own. This should not be possible – only the building owner should be able to submit build requests, and therefore sending the API key of another player should not allow the request to pass. To do this I used this POST request:

[***http://127.0.0.1/Tauresium/TaurAPI/Building/India\_MadhyaPradesh/YG5YDTTTHM819CAK/C***](http://127.0.0.1/Tauresium/TaurAPI/Building/India_MadhyaPradesh/YG5YDTTTHM819CAK/C)

The provided province is a province owned by green, whereas the API key provided is owned by the pink nation, this request should therefore not succeed.

*Result*: The API responded with “400 You are not the owner of this location” – demonstrating that the API is able to determine that the player with the provided API key should not be permitted to perform this action. In addition, the building was not constructed and neither players influence balances were modified in any form – showing that the construction request was not completed.

Successful

# Test 62 – Military capacity bonus

*Description*: When a player constructs a building, they will receive bonuses to certain statistics, one of which is their overall military capacity. This test will examine the before and after military capacity of a player when they create a new building providing this statistic. This will be done by constructing a “**Military Base**” (**M1**) on a location, which provides +10 military capacity. Before starting this test, the subject player had a military capacity of 405.

*Result*: Upon construction, the players military capacity was incremented to the value of 415, demonstrating that the capacity increase is properly implemented for the player.

Successful

# Test 63 – Military capacity bonus 2

*Description*: When a province is taken, all constructions in the location should be taken with it – hence, when a player loses a province, the provinces initial military capacity bonus and the bonus from any buildings should be lost. This test will see one player take a province with the aforementioned “**Military Base**” (**M1**) construction, to observe the effects on the military capacity of both players after this action. The province itself provides an initial bonus of 15, which is then incremented by 10 due to its constructions, giving it a total military capacity bonus of 25 – as such, the annexing player (with their current military capacity being 1835) should receive a new total of 1860 military capacity, whereas the original owner (with their capacity of 415) should have their capacity lowered to 390. Additionally, the original owner’s military influence was set to 415, to see if there are any issues that occur when a player’s military capacity is reduced past their current influence.

*Result*: Upon annexation, the annexing players capacity was set to the expected 1860, and the capacity of the original owner was lowered to 390, both properly representing the transfer of 25 military power. Additionally, the original owners current military influence was then lowered to 390, demonstrating that the lowering of military capacity will deduct military influence if required. It should be noted however, that this effect will only take effect the next time a request is made, or the account is active on the website – therefore if a GET request is made to the original country, it may show their unadjusted military influence, but this should automatically be changed whenever any request is made, and therefore cannot be abused in any way.

Low Priority/Minor Issues

# Test 64 – Build cost modifier

*Description*: Another statistic that can be improved by buildings are the costs of future buildings to the location, players may wish to prioritise buildings that lower future building costs to avoid wasting influence if possible. This test will ensure the building cost reduction is applied both:

* After the building is constructed
* At the correct amount
* With all other build cost modifiers applied additionally to create the total reduced cost.

To do this, I will first construct the building “**Cathedral**” (**C1**) which will cost 50 cultural influence to build, I will then construct the “**Military Base**” (**M1**) which, while originally costing 50 military influence, should now cost 45 military influence due to the 10% cost reduction of the previous building. After this, I will construct the “**Train Station**”(**C2**), followed by the “**University**” (**C3**), the latter of which provides a -25% build cost, then I will construct the “**Airfield**” (**M2**) which, while originally costing 100 military influence, should use the reduced cost of 65 military influence due to the 35% cost reduction.

*Result*: The first construction cost the expected amount, 50 culture influence, signalling that the build cost modifier is not appended until after the construction is complete, after this I attempted to create the military base as previously mentioned, which had the reduced cost of 45, which was correctly deduced from the balance with the adjusted price. After this, I then built the other two culture buildings granting me -35% build cost, and then used my remaining military influence to buy the airfield, now worth the expected 65 military influence, which was correctly deducted at this correct price. It seems the build cost modification works entirely as intended.

Successful

# Test 65 – Local defensive strength

*Description*: The last available construction modifier, local defensive strength, refers to the additional military penalty cost for attempting to annex the location – this statistic is strong, and most buildings that grant this effect do so in large amounts – in some instances, I have observed province costs growing to roughly 400 military influence to take, due to the player constructing all the possible defensive buildings in that location. This test will determine if the defensive strength modifier is correctly applied, by noting the before and after military cost of a province when the local defensive strength is incremented. The province used for this, while having no defensive strength modifiers, cost a total of 98 military points, and while the exact order at which modifiers are applied might stop the result from being an exact double of this (as the building used, “**Airfield**” (**M2**) should provide +100% cost), any significant increase in cost will be counted as a success.

*Result*: After constructing the +100% defensive strength building, the originally 98 military cost provinces incremented to a cost of 166, which is a roughly 70% cost increase, as previously mentioned, the exact order in which cost modifiers apply will stop any changes from being exactly 100%, especially as the original cost already had military modifiers applied (due to adjacency and other factors) which would be not changed by the new construction modifier, therefore this will be counted as a successful test.

Successful

# Test 66 – Local defensive strength 2

*Description*: This test will determine if the defensive strength bonuses are added together, this test will use the previous test province and construct additional defensive strength buildings to examine any changes to cost, this test will be subject to the same rules as the former, with any significant increases in cost being counted as a success. To do this, the building “**Cathedral**” (**C1**) Will be constructed, which applies a +20% defensive strength modifier.

*Result*: After construction, the cost to take the location increased to 180 military influence, which is a roughly 83% increase from its original cost of 98, and an 8% increase over its former value of 166. This is an expected and successful result, and interestingly it should be noted that the percentage increase reported in the province screen is an accurate representation of the real modifiers applied to the military cost, with the original cost of the province in question, “**India\_MadhyaPradesh**”, being worth a base of 68 military influence – with the province screen reporting a 165% increase in cost, which when calculated equals the expected 180 military influence (when rounded down).

Successful

# Test 67 – Session stats page

*Description*: This simple test is just used to ensure the session details screen correctly loads player details and will be tested by changing the number of occupied provinces of a player to ensure the values correctly update as expected. The session stats page stores some statistics of each player in the game, with more details available using the API. To do this test, I used a player with 79 provinces, 1835 military capacity and 3 of 4 provinces needed for dominance in the East African coastal region, I then annexed a new coastal location in the East African region, which should increment the provinces number to 80, the military capacity to 1850, and set the player to dominant in the East African Coast region.

*Result*: All values were changed to their expected value, and the same occurred when testing the taking of a province via military annexation, verifying that the statistics on the page are valid.

Successful

API-Centric Tests

This segment will focus on the API and the parameters sent to it, using the same acceptance, edge case and erroneous model as the previous section. This segment will cover many of the same features previously discussed in this document, but with manually made HTTP requests – which open the project up to many other vulnerabilities in terms of error checking. For more information regarding the API and the structure of TaurAPI requests, see the report which contains a list of all valid requests and their parameters.

# Test 68 – Empty request

*Description*: This test concerns the sending of an empty request to the API server, with no additional parameters provided aside from the inclusion of the API tag, TaurAPI, in the request. This request is expected to provide some form of failure response, as some form of identifying parameters must be provided to the API to construct a query.

*Result*: The API returned the header “405 Method Not Allowed” with no additional JSON responses attached. This response was the same regardless of the method supplied (If valid)

Successful

# Test 69 – Invalid method

*Description*: TaurAPI only accepts four method types – GET, POST, PUT and DELETE. For this test, I will construct a basic API request using an invalid method type. The request provided, a country request, will be an entirely valid request but will have the method “NICE”. This request should fail and return an error response code.

*Result*: The API returned a “501 Not Implemented” response, demonstrating that the method type supplied is not an accepted request type.

Successful

# Test 70 – Invalid command

*Description*: This test will discuss the result of sending a request of any type using an invalid or unsupported command, “Egg”. This should return a negative response as the use of an invalid command should not be permitted.

*Result*: The HTTP request received a response code of “501 Not Implemented”, as intended. Additionally, any further parameters supplied did not impact the result in any way when tested.

Successful

# Test 71 – GET Province (No parameters, Acceptance)

*Description*: In this test I will create a HTTP request with the command “Province” and no additional parameters. The province command is one of the only requests that supports returning all values, many others do not have this functionality implemented for security purposes, and upon the sending of this request, all provinces should be returned in JSON format, with their basic (not world dependent) properties provided.

*Result*: The header for this response was a basic “200 OK” response, designating that the request had been accepted, and the JSON response provided a list of all provinces in the game, as well as the details that would be provided for each when the province page was accessed during normal gameplay.

Successful

# Test 72 – GET Province (Province ID, Acceptance)

*Description*: This test will utilise the ability for the API to return basic information regarding a specific province, by the supplying of a valid ID as the first parameter. For this test, I will use the location‘**India\_SriLanka**’ as the parameter.

*Result*: This request returned a 200 OK response, and the JSON provided described the basic details of the location. These details are the same ones included with the no-parameters variant of the province command, demonstrating the API is able to return a specific value of the province set on request.

Successful

# Test 73 – GET Province (Province ID, Edge Case)

*Description*: This test will attempt to supply other identifying information as the province ID when submitting a request, notably the capital city of the location. This request should be permitted for this variant of the command and should display the same results as if the province ID itself was supplied. For this test, the parameter provided will be the city name of ‘Kotte’.

*Result*: A 200 OK response was returned, and the information provided was the same as that of ‘**India\_SriLanka**’, as the city “Kotte” is the capital city for this location. This demonstrates the API is able to use the name of a city as a result rather than the ID, though it should be noted this is the only case in which use of a city name is permitted rather than a province ID.

Successful

# Test 74 – GET Province (Province ID, Erroneous)

*Description*: In this test I will supply an invalid province ID name as the parameter for this request, this should return a failure response as the location specified cannot be within the dataset. For this test I will provide the parameter “AAA” as the province ID.

*Result*: The server responded with a “400 Not Found” response, signifying that the requested parameter cannot be found within the provinces dataset. Additionally, no JSON response was included.

Successful

# Test 75 – GET Province (Province ID + World Code, Acceptance)

*Description*: When provided with a world code, the province command should return all relevant provincial information, but with the inclusion of world specific information such as the constructions at the location, as well as the current owner. For this test I will use the ‘**India\_SriLanka**’ location along with the world code ‘EPTR65E23EJ4HFTZ’, which should return a location occupied by the country “ADMIN” with tier 1 military buildings and tier 2 economic buildings.

*Result*: The server responded with a 200 OK response and returned all the information of the province along with the world specific information – the owner and the buildings, as well as any bonuses supplied by the constructions in the location. This is the expected outcome of this request.

Successful

# Test 76 – GET Province (Province ID + World Code, Edge Case)

*Description*: As previously mentioned, this request should not be able to use any other identifying information such as capital city names as a parameter – this test will use the ‘Kotte’ location as the parameter, as well as the same world code as the last test, to discover the response received when this is attempted.

*Result*: This request returns a 200 OK response, however, the JSON response is notably erroneous, with a returned value signalling an error has occurred in the API script, this is likely due to the possibility of an invalid province ID being supplied is not correctly handled. This result can be assumed to be the same for any invalid province ID, regardless of it is an existing capital city.

API Issue

# Test 77 – GET Province (Province ID + World Code, Erroneous)

*Description*: As the use of an invalid parameter has been discovered to cause issues, this erroneous test will solely refer to the use of an invalid world code, as well as a world code that does not have any occupiers in the region, hence giving the province no world specific information to refer to.

*Result*: When supplied with an invalid world code, the server sent a 200 OK response, and provided the basic JSON of the province, as would be returned when only the province ID was supplied as a parameter. Doing this with a valid world code but one that does not have any modifications to the specified location provided the same results.

Successful

# Test 78 – GET View (No Parameters, Erroneous)

*Description*: This test will check the results of attempting to use a GET request on the view tag, used for returning information like province visibility and province owners, when no parameters are provided. This should not be an accepted request, as the View command is player specific and should return information based on the supplied player API key.

*Result*: A “404 Not Found” response was received, with no JSON information attached, this is because the API is designed to handle requests with no parameters supplied.

Successful

# Test 79 – GET View (API Key, Acceptance)

*Description*: This test will concern the use of the view command when supplied with a valid API key, this should return a list of provinces with visibility information and ownership information supplied. This data is based on the world the player inhabits as well as their own provinces – players will be able to view locations they have any link to – coastal or adjacent.

*Result*: When supplied with a valid API key, a 200 OK response was received, as well as a json response detailing all provinces on the map, with appended information such as their visibility, owner, and colour (as determined by the owner)

Successful

# Test 80 – GET View (API Key, Erroneous)

*Description*: In this test I will perform the View request again but supply an invalid API key – this should have the same results as the no parameters test, as the API should discover that the supplied API key does not exist within the database.

*Result*: As expected the response received was a “404 Not Found” response, with no additional JSON returns, demonstrating that this feature works as intended.

Successful

# Test 81 – GET Country (No Parameters, Erroneous)

*Description*: The country GET request can be used to examine details concerning a specific user by supplying their name as a parameter, however, this feature cannot be used to display all users within the database for security reasons (some players will wish to keep their world codes private and doing this would permit any user to read all the world codes in the game), therefore providing no parameters should cause a failure response.

*Result*: This method caused a response of “405 Method Not Allowed” with no additional JSON response. This demonstrates that the API will not permit users to access the information of all registered players of the game, as is expected.

Successful

# Test 82 – GET Country (Country Name, Acceptance)

*Description*: This test will attempt to use the country method to return information relating to a specific user, namely “ADMIN”, using their name as a parameter. This should be accepted, and should return various user information, such as their name, colour, and others. It should be noted however that this request should not return certain information such as the hashed password or API key of the user.

*Result*: The call was successful, returning a 200 OK response with appended JSON information including all the non-sensitive information about a player. This request also appends information such as coastal dominance and the list of all owned locations, for use within the session statistics screen.

Successful

# Test 83 – GET Country (Country Name, Erroneous)

*Description*: In this test I will attempt to provide an invalid country name as a parameter for the GET country request. As the player’s name provided, “B5D”, does not exist, there should be a failure response from the server.

*Result*: This request resulted in a “404 Not Found” Response, demonstrating that the name supplied does not exist within the dataset, and therefore there was no information to supply in response to the request. As with other failure responses, this HTTP request provided no JSON response.

Successful

# Test 84 – GET Country (API Key, Edge Case)

*Description*: This test will attempt to use the country API method with the API key, returning country information based on the supplied key. This is not a feature I have implemented, but due to the set up of the API, may be one that is possible, therefore this test might have unexpected results.

*Result*: The API returned a 404 Not Found response, as it interpreted the supplied API key as a country name, and therefore did not find any results for the request. This result is neutral, as while it did not work, it was not an intended feature and did not cause any other issues, and therefore does not need changing.

Successful

# Test 85 – GET Country (Country Name, Edge Case)

*Description*: In this request, I will provide a valid country name but with invalid case. As case is important for other features such as login, it is possible that this may cause issues if the API does not consider case in the request. For this test I will send the parameter “admin”, which is the lowercase equivalent of the user “ADMIN”, which ideally will not be interpreted as a valid country name.

*Result*: The request received a 200 OK response and the information for the user “ADMIN” was supplied as a result. Originally this was considered to be a severe issue – as this could mean a user may be able to gain access to an account by creating an account with a similar name but with different case, as the API would not be able to differentiate the two. However, this is actually not the case, as when creating a country, the API will check for duplicates regardless of case, and therefore creating a nation with the same name but with different case is impossible unless the player has access to the database itself. It still cannot be ignored that this presents a problem where some aspects of the website include case sensitivity and some do not, but this issue was not as catastrophic as originally expected.

Low Priority API Issue

# Test 86 – GET World (No Parameters, Erroneous)

*Description*: As with the country list, there should not be a way for players to list all available worlds. Players are expected to share their world codes through their own means, not just join random available games. This test will attempt to provide no parameters to a world GET request in order to attempt to list all available worlds.

*Result*: Similar to the country tests, this request returned a “405 Method Not Allowed” response with no included JSON data, demonstrating that the command to access all available worlds is not accessible.

Successful

# Test 87 – GET World (World Code, Acceptance)

*Description*: The only identifying information regarding a world is its world code, even things like world names can be duplicates, therefore the GET world request will only accept a world code as a parameter. This test will ensure this feature works by providing a valid world code as a parameter, after which the server response should detail information about the world, including information on its occupants.

*Result*: The request returned a 200 OK and the JSON data response included all relevant world information including occupant details, as is expected.

Successful

# Test 88 – GET World (World Code, Erroneous)

*Description*: This test will cover the case in which an invalid world code is provided in a GET world request, which should clearly result in a failure response.

*Result*: As with similar cases the server responded with a 404 Not Found response and provided no JSON response.

Successful

# Test 89 – GET World (World Code, Edge Case)

*Description*: While this likely cannot cause any issues, I am curious if the case issue previously mentioned in this document is present in other parts of the program, particularly in the world code. This test will use a valid world code (usually in full upper case) with lower case letters in place of the upper-case letters to verify if this is a consistent issue.

*Result*: As with the country name, the API also interpreted a lower-case variant of the world code as a valid code, though this has no potential avenues for abuse, as the world code cannot normally generate any lowercase letters. This does mean that it is likely all parameters are not-case sensitive however, so this must be considered in the future.

Successful

# Test 90 – GET Building (No Parameters, Acceptance)

*Description*: Unlike country and world, the building method can be used to return all applicable buildings by providing no parameters, as it contains only information that is intended to be accessible by any user. This test will try and use this method to return all the buildings in the game, in the same manner as was done for provinces.

*Result*: The server responded with a generic 200 OK response, and the JSON response provided contained a list of all 15 buildings implemented in the game, as well as their bonuses and names.

Successful

# Test 91 – GET Building (Building Name, Acceptance)

*Description*: This test will attempt to access the build method by sending a HTTP request to the API requesting the details for a specific building, namely “**Airport**”, which is the building at id C4. This should return a valid response with JSON including the details for said building.

*Result*: When provided with a building name, the API returned a 200 OK response with JSON data that detailed the information regarding the specified construction

Successful

# Test 92 – GET Building (Building Name, Erroneous)

*Description*: This test will attempt to send a request incorporating an invalid name of a building as a parameter, which should produce a failure response.

*Result*: The API responded with a “404 Not Found” response and as usual had no additional JSON data. This is an expected result from this test.

Successful

# Test 93 – GET Building (Building ID, Edge Case)

*Description*: There are two identifying parameters for a building, the name itself and the internal ID of the building, this test will attempt to use the building ID as a parameter in place of the building name. This is not expected to work, as the API does not need to search by building ID during the use of the front-end website, hence the feature was not implemented, though it should still be tested in case anything unexpected occurs.

*Result*: The API responded with its usual response when provided with invalid data, “404 Not Found” with no JSON data included. This demonstrates that the API is not able to search based on ID data, but this is not considered an issue as this functionality is not necessary.

Successful

# Test 94 – GET Government (No Parameters, Acceptance)

*Description*: Like with the buildings, government type information is intended to be accessible by all users, and hence it should be possible to perform a full return of all government type data by providing no identifying parameters to the API. This can be done by using the same methods as other requests, providing no parameters aside from the method type, Government.

*Result*: The API returned a 200 OK response and provided the basic stats of all government types implemented in the game, with information like its title and base focus values incorporated in its JSON.

Successful

# Test 95 – GET Government (Government Name, Acceptance)

*Description*: As with other request types, the government method can be used to return information for specific government types by including the name of the government as a parameter in the HTTP request. This should only return statistics for the government type specified. This test will use the type “**Sultanate**” as a parameter, and all returned results are expected to be related to this government type.

*Result*: A 200 OK response was received with included JSON data describing the base stats of the Sultanate government type, as was expected.

Successful

# Test 96 – GET Government (Government Name, Erroneous)

*Description*: For this request, I will use an invalid government name as a parameter in order to attempt to query the database for a non-existent form of government. As usual, this is expected to provide a 404 response, as no information can exist for the provided parameter, “AAA”.

*Result*: As expected the server responded with a 404 response due to there not being an entry in the dataset for the government type “AAA”. Additionally, as usual, there was no JSON data included.

Successful

# Test 97 – GET CoastalRegion (No Parameters, Acceptance)

*Description*: Once again coastal regions are expected to be publicly accessible information and therefore when no additional parameters are supplied the server will default to displaying all the coastal regions within the game. This test will determine if this feature is properly functioning.

*Result*: The server responded with a 200 OK response with JSON data for all coastal regions in the game included as anticipated.

Successful

# Test 98 – GET CoastalRegion (Region Name, Acceptance)

*Description*: This test will use the ability of the API to return information for a specific target coastal region as demonstrated with government forms and buildings, by sending the region name (specifically the region ID, not the coastal title that is displayed on the session stats screen) as a parameter for the HTTP request. For the purposes of this test, I will use the parameter “**Mexico**”, referring to the most useful coastal region in the game.

*Result*: As per usual, the server granted a 200 OK response with the JSON data for the coastal region of Mexico included as the JSON response data.

Successful

# Test 99 – GET CoastalRegion (Region Name, Erroneous)

*Description*: This test will follow the same pattern as previous similar methods, using an invalid coastal region name as a parameter to see if the API is able to recognise this problem and respond with a 404 response or similar.

*Result*: The Server responded with a generic 404 Not Found response, stating that the identifier requested did not return any results from the database.

Successful

# Test 100 – GET Cost (No Parameters, Erroneous)

*Description*: The cost method requires a province ID and a country name and returns the costs of taking the location as based on statistics like the buildings in the region and adjacency bonuses. This method should not accept any parameters less than the country name and province ID, and therefore passing no parameters at all should return some form of failure response.

*Result*: As expected, the COST request with no provided parameters returned a 404 Not Found response, with the JSON data being empty. This was one method I was unsure about in terms of response to invalid parameters, so it is nice to see the API successfully handles this request.

Successful

# Test 101 – GET Cost (Single Parameter, Erroneous)

*Description*: While no parameters will return a 404 response, it is important to note that results for passing a single parameter may not be the same. Additionally, this test will only cover providing a country name, as you cannot format a HTTP request in a way that ‘skips’ a parameter. For this test I will use the valid country name ‘**ADMIN’** as the sole provided parameter.

*Result*: This test had mixed results, with the error code successfully reporting “404 Not Found”, however the JSON response included an error message referring to an undefined variable, likely due to the fact that no province ID was supplied.

API Issue

# Test 102 – GET Cost (Country Name + Province ID, Acceptance)

*Description*: This test will use the cost method as intended, by providing the province ID and country name as parameters for the request. This should return the cultural, economic, and military cost of a location for the specified player. This test will use the aforementioned ‘**ADMIN**’ account as the country name, and the location “**China\_Yunnan**” as the province ID.

*Result*: The API provided a 200 OK response with JSON data containing accurate results for the cost of the location in each influence category, as well as brief descriptions of the circumstances for the cost, such as adjacency and other parameters.

Successful

# Test 103 – GET Cost (Country Name + Province ID, Edge Case)

*Description*: This test will test the case in which a player attempts to read the cost of their own province, which is expected to return three “Infinite” cost values. This test will use the province “**SEAsia\_BurmaNorth**” – a location owned by the user ‘**ADMIN**’.

*Result*: The API returned 200 OK and three infinite values for the location, as well as some brief messages describing the location as “Owned by another player”, except in the case of military, in which is states “This is your own province” – a message that would not be seen during normal gameplay. Regardless this test has proved successful, as the returns did not contain any errors.

Successful

# Test 104 – GET Cost (Country Name + Province ID, Erroneous)

*Description*: This test will cover providing an invalid country name as a parameter for the cost method, once again using the location “**China\_Yunnan**”, though this time with the country name “**OneOhFour**” – a country that is not a registered user. This test is expected to provide a 404 Not Found response.

*Result*: This request provided a 404 Not Found response as anticipated, as well as – thankfully – no JSON response of any form.

Successful

# Test 105 – GET Cost (Country Name + Province ID, Erroneous 2)

*Description*: In this test I will examine the use of an invalid province ID with a valid country name, the country name being “**ADMIN**”, and the province ID being “**China\_Egg**” – the latter not being a valid in game location.

*Result*: This test resulted in a 404 Not Found response with no appended JSON response.

Successful

# Test 106 – GET Cost (Country Name + Province ID, Erroneous 3)

*Description*: For this test I will combine the prior two tests to submit a request including both an invalid country name and an invalid province ID, which ideally should result in another 404 Not Found Error. For this I will use the username “**OneOhSix**” and the location “**China\_Egg**”

*Result*: As with both other erroneous requests, this request responded with a 404 Not Found response with no JSON data included, demonstrating that the API can deal with erroneous requests like this.

Successful

# Test 107 – GET Event (No Parameters, Erroneous)

*Description*: The event method is used to retrieve the currently accessed event of a country via their API key, but notably does not load a new event if one is not already in use. This first test will attempt the use of the event method without providing any API key, which should return a 404 error as the method is intended only to access the loaded events of players, not events themselves.

*Result*: The API responded with an unexpected header – “401 Bad API key supplied”, referring to the lack of API key provided in the query. This, while not anticipated, is still an acceptable result, especially as the response had no included JSON data.

Successful

# Test 108 – GET Event (API Key, Acceptance)

*Description*: This test will experiment with the use of the event method when given valid conditions – a valid API key for a player who has an event currently loaded. The provided API key was that of the user ‘**ADMIN**’, who had an event loaded at this point in time.

*Result*: The API responded with a 200 OK response and provided the correct details for the event, including descriptions of each of the options for the event.

Successful

# Test 109 – GET Event (API Key, Erroneous)

*Description*: In this test I will provide the same request as the previous test, but on a user without an event currently loaded. This is expected to form some type of failure response, either a generic 404 Not Found response or a response that is more specific about the issue.

*Result*: The API responded with a new variant of the 404 response – “404 Player has no active event”, a valid response for this request. Additionally, while there was some JSON data supplied, it simply said “null”, which, while unnecessary, has no ill effects on the use of the API.

Successful

# Test 110 – GET Event (API Key, Erroneous 2)

*Description*: In this final GET test, I will attempt to use an invalid API key as a parameter for the event request method – which should result in a 401 error as previously demonstrated.

*Result*: Just as expected the server responded with a “401 Bad API key supplied” error, and with no JSON data appended. This means this test was successful.

Successful

# Test 111 – POST Country (No Parameters, Erroneous)

*Description*: For this test I will attempt to use the create country method with no parameters in order to ensure it properly blocks any invalid requests – it should also be noted that these POST requests (country and world) are designed to return JSON parameters based on the errors that occurred, formatted for use as a URL modifier, the user is expected to interpret the error message by themselves.

*Result*: The request resulted in a 400 response with URL parameters in the header and JSON data signalling the errors that occurred with the request. It should also be noted that the API was unable to append a “MISSING” tag, which would usually be included for missing parameters, for unknown reasons, but instead used the default INVALID tag, however this is not an issue.

Successful

# Test 112 – POST Country (1-4 Parameters, Erroneous)

*Description*: All POST country method requests require exactly 5 parameters to complete the query, as the creation script cannot function with any less. This script will experiment with providing 1 through 4 parameters of valid quality – this latter condition is to ensure the results are not modified by a failure response to one of the parameters, as this is purely a test of the parameters required themselves.

*Result*: All tests returned their expected 400 response, each providing their own URL error parameters describing the invalid responses given. As with the last test, no new records were appended to the database, proving that this response did not create a country.

Successful

# Test 113 – POST Country (5 Parameters, Acceptance)

*Description*: This test will ensure that a country is able to be created and logged into when all 5 valid parameters are provided in the request, this should result in a new country being created in the database. For this test I will use the following parameters:

* Country Name: Iseb
* Password: Goob
* World Code: EMFK7T4E4817ZEFC
* Government Type: Tribe
* Colour: ff0000

*Result*: The server returned a 200 OK response, and the data (the country and its first province) were appended to the database. Attempts to login to this new account were successful when provided the same country name and password as specified, and interestingly the case was maintained for this request, as the login was not usable without the capitalisation seen above, which calls into question the use of capitalisation in API requests.

Successful

# Test 114 – POST Country (5 Parameters, Edge Case)

*Description*: In this test I will attempt to create a POST request providing duplicate fields where duplicates are not permitted – the username and colour fields. Additionally, I will also test the case in which a world code with full players is used, which should similarly result in a failed test. It should be noted this refers to three disconnected tests, the three will not be tested simultaneously.

*Result*: Each test returned the expected 400 response, demonstrating that these key fields do not accept duplicate inputs.

Successful

# Test 115 – POST Country (5 Parameters, Erroneous)

*Description*: This test will focus on the government type and colour fields, which only accept certain values as inputs, therefore I will test each by providing inputs that are not included as valid responses. All other parameters will be valid within the HTTP request, and each test will be conducted separately using the following values:

* Government type: Hivemind
* Colour: 4A412A

*Result*: Both requests returned the expected 400 response designating the supplied inputs as invalid – demonstrating these fields will only accept their expected input values.

Successful

# Test 116 – POST Country (5 Parameters, Erroneous 2)

*Description*: As the majority of test cases for the country input have already been covered using the front end, this test will simply examine the case in which all 5 parameters are incorrect to fully ensure that all error checking is properly enabled through the API. If this test fails, its URL string return should detail each of the errors that occurred, which will demonstrate the error checking is supplied using the API rather than the country creation script used before calling the API.

*Result*: The API responded with a 400 response with the appended data defining each of the 5 parameters to be invalid, demonstrating that the error checking is completed through the API as is required.

Successful

# Test 117 – POST World (No Parameters, Erroneous)

*Description*: Much like the country method, the majority of the world creation functionality has been tested earlier in this document, which means this segments error checking should only be to test inputs not possible without direct HTTP requests – the first of which being a lack of any parameters provided. This would usually be handled through the world creation screen by appending “NULL” as the parameter, which the API would then interpret as a missing response. This means only the API can be used to send no parameters at all, hence this test.

*Result*: As with the country method, failed requests are send using a 400 code with the URL parameters defining the errors appended to both the header and the JSON. This is what occurred in this case, though of note is the fact that the world creation was able to recognise missing parameters and provide the MISSING response, rather than the default INVALID response.

Successful

# Test 118 – POST World (1-2 Parameters, Erroneous)

*Description*: This test will examine the case in which a player submits only one or two parameters, though these will be valid for reasons discussed in previous tests, which should still result in a failure response.

*Result*: In both cases the API detected that not all parameters required were provided, and therefore did not permit the world to be created, resulting in a 400 error with the URL parameters included as expected.

Successful

# Test 119 – POST World (3 Parameters, Acceptance)

*Description*: This test will handle the case in which all 3 parameters are sent with valid values in the HTTP request, which should result in the creation of a new world. For this test I will use the following parameters:

* World Name: Egg
* World Type: Earth
* World Speed: Quick

*Result*: The API responded with a 200 code with the heading message requesting the user examine the JSON response for the world code, which included the world code of a newly generated world (which was verified to exist within the database, with the parameters specified included). This demonstrates the world POST request is usable.

Successful

# Test 120 – POST World (3 Parameters, Erroneous)

*Description*: This test will experiment with providing all three parameters will invalid inputs, to ensure the API will discover the issues and deliver a failure response in return.

*Result*: The API discovered all three erroneous inputs and returned a 400 response. Additionally, this response disclosed the URL return in the JSON rather than both the header and the JSON, which while inconsistent with the previous method is still not an issue.

Successful

# Test 121 – POST Event (No Parameters, Erroneous)

*Description*: Moving onto the event POST request, the API requires both an API key and an option number in order to respond to an event, hence it must be tested the response in which neither an API key nor an option number is selected.

*Result*: The error code 401 was received, defining the API key supplied as invalid. While this is clearly not the only issue with this request, the API seems to perform an API key check first and prioritise responding to the API key error.

Successful

# Test 122 – POST Event (1 Parameter, Erroneous)

*Description*: This test will examine the case in which an event method is sent with a valid API key but no option number to a player without an active event. Both the fact that there is no selected option number and the fact that the player in question has no event loaded should cause this request to return a failure response.

*Result*: The received response was a 400 error with the message “Player event ID is invalid, is there an event loaded?”, demonstrating that the API has at least recognised that the player has no active event.

Successful

# Test 123 – POST Event (1 Parameter, Erroneous 2)

*Description*: This test will serve as a response to the former test, attempting a POST event method while an event is loaded, but without providing an option number parameter. This is expected to return a failure response to the user, as the player is expected to select an option for this method.

*Result*: A 400 error response was received, with the appended message “Invalid Event Option”, demonstrating the API is able to recognise when an event option has not been provided.

Successful

# Test 124 – POST Event (2 Parameters, Acceptance)

*Description*: In this test I will use the event method as intended, on a player with a loaded event with a valid API key and a valid option number, to ensure the response is correct. The expected response should include the changes made to the player based on the response provided, and the input used to receive this response will be the following:

* API key: N5DAXWGKLQEGXMMC
* Event Option: 3

*Result*: A 200 OK response was received with the JSON data including details on each of the changes made as a result of this option being selected. It should be noted from this test that the event option should always be a number 1,2 or 3, as the option ID itself is not used in this method.

Successful

# Test 125 – POST Event (2 Parameters, Erroneous)

*Description*: In this test I re-attempted the previous test using the same inputs but without an event loaded, which should cause a failure response to occur, as the player should not be able to submit an answer to an event they have not yet loaded.

*Result*: the API returned the same error as received in test 122 – demonstrating that the API is able to recognise when a player has no event loaded and will prevent any actions being taken if the player does not have any events.

Successful

# Test 126 – POST Event (2 Parameters, Erroneous 2)

*Description*: For this test I loaded a new event onto an account and attempted to send a request using both an invalid API key as well as an invalid option number, with the expectation being that both should cause a failure return.

*Result*: As expected, the response received was a 401 error, with the error message defining the API key as invalid, demonstrating that the API key check is processed first. This test will be the last for the event method, as the validation of API keys has been demonstrated, and previous tests have shown the results of providing an invalid option as the response.

Successful

# Test 127 – POST Building (No Parameters, Erroneous)

*Description*: This test will concern the use of the building method without any parameters build supplied, much like the other POST tests. Ideally, this test should only result in a failure response and should not impact the game in any way.

*Result*: Similarly to other tests, the API first noticed the API key not corresponding to a player, and then sent a response stating that the provided API key was invalid. This is interesting, as it is not the first parameter in the query.

Successful

# Test 128 – POST Building (3 Parameters, Acceptance)

*Description*: In the previous testing on buildings, HTTP requests were used to demonstrate the use of certain invalid responses, due to this, the majority of API requests concerning the buildings POST request have already been demonstrated and tested and hence this test will only concern acceptance testing and parameter testing of the feature. The inputs for this test were the following:

* Province ID: China\_Jilin
* API Key: N5DAXWGKLQEGXMMC
* Point Type: M

*Result*: The API responded with a “200 Successfully constructed building” response, and the province and the players point balance were updated accordingly.

Successful

# Test 129 – POST Building (1-2 Parameters, Erroneous)

*Description*: In this test I will test the result of providing both one and two valid parameters to demonstrate that the API should submit a failure response as a result of these invalid queries.

*Result*: The first test resulted in a 401 Bad API key response, as no API key was supplied. Once this API key was supplied, the error changed to a 400 Invalid building type, demonstrating that the API can detect when invalid parameters are used. This will conclude testing on the building method as other cases of its use have already been demonstrated in previous tests.

Successful

# Test 130 – POST Province (No Parameters, Erroneous)

*Description*: Much like the building method, the majority of uses of the annexation method, “Province”, have been demonstrated in prior tests, hence these tests will only cover the API specific interactions that can be performed. This first test will attempt to provide no parameters to the HTTP request and is expected to result in a 401 response.

*Result*: While the API did respond with a 401 response, it did not specifically mention the API key as the cause of the error, but nevertheless did not change any database information and provided only a JSON response saying “INVALID”.

Successful

# Test 131 – POST Province (3 Parameters, Acceptance)

*Description*: This test will attempt to demonstrate the POST province method functions as expected, by providing the following valid parameters to the HTTP request:

* Province ID: China\_Liaoning
* API Key: N5DAXWGKLQEGXMMC
* Point Type: C

*Result*: The API responded with “200 Successfully annexed location” as a response, with JSON data stating “SUCCESS” as a code to be used by other scripts. Examining the map did prove that the province had been successfully annexed by the player with the API key, therefore showing this method is usable.

Successful

# Test 132 – POST Province (1-2 Parameters, Erroneous)

*Description*: This final province method test should demonstrate that the API can properly handle receiving less parameters than expected. As with similar tests, all these provided parameters will be valid to ensure other errors are not detected which may change the result of the test, and the test will be completed twice, once with one parameter and the second with both. Hence the provided parameters will be the following:

* Province ID: China\_NorthInnerMongolia
* API Key: N5DAXWGKLQEGXMMC

*Result*: Both tests provided the same “401 Bad Parameters” return, demonstrating that the API will not allow the use of less than 3 parameters for this method.

Successful

# Test 133 – PUT Event (No Parameters, Erroneous)

*Description*: The PUT Event method is a way for the API to dynamically update the last online timer as well as facilitate the loading of events, two essential features for the running of Tauresium. Due to this method being a secure one, an API key is required as the only parameter for this request. This first test will attempt to use the Event method with providing an API key.

*Result*: The API responded with the generic “401 Bad API Key supplied” error, demonstrating that the inclusion of a key for this method is a requirement.

Successful

# Test 134 – PUT Event (API Key, Acceptance)

*Description*: This test will attempt to use the event method to update a players in game time. For the purposes of this test, the player in question has no event stacks and their last recorded online time was at 16:46:30. After this test, it is expected that this time will have updated, as well as by extension the event stacks, but no event should be loaded as a result.

*Result*: The webserver sent a “200 Updated last online time. No event could be loaded”. Checking the database proved that both the timer and event stacks had been updated, whereas no new active event had been loaded, confirming the response was legitimate.

Successful

# Test 135 – PUT Event (API Key, Acceptance 2)

*Description*: This test will build off the last event, by providing the player with an event stack, which should permit the API to load a new event after the PUT request is made. This test will use the same parameters as before but this one difference in situation should permit an event to be loaded.

*Result*: The API responded with “200 Updated last online time. Loaded new event” by both updating the event time and stacks before loading a new event into memory, allowing other requests like the GET Event and POST Event to be used.

Successful

# Test 136 – PUT Event (API Key, Edge Case)

*Description*: For the purposes of this test, the events stack of a player was set to 0.999999, the closest possible to 1 event stack. When the PUT event is received, the event stack should increment to 1, and hence a new event should be loaded into memory, despite there being no events stacks available when the API originally received the request.

*Result*: The API sent the response “200 Updated last online time. Loaded new event”, demonstrating that the loading occurs after the updates to time and event stacks occur. Checks to the database also demonstrated that this action resulted in the database loading an event for the player and said players event stacks being reset, with any additional event stacks that exceeded the 1 being all that remained.

Successful

# Test 137 – PUT Event (API Key, Erroneous)

*Description*: Due to the circumstances around the method, the only possible erroneous input for the event method is to supply an invalid API Key, as the API will never deny a time update for a user and the method only accepts a single parameter. Hence this test will examine exactly this – the use of an invalid API key in the PUT event method.

*Result*: As expected, the API returned a response of “401 Bad API Key supplied”, demonstrating that the API does check that the key specified belongs to a specific player.

Successful

# Test 138 – DELETE Event (No Parameters, Erroneous)

*Description*: The DELETE Event method exists as a method only accessible by the API, as it was discovered to provide no tangible benefit to a user during development, however due to it not being removed from the system, it is still important to test its usage to ensure it cannot be used for exploits. Like its PUT counterpart, the DELTE Event only requires an API Key as its provided parameter. This test will attempt to send a DELETE request without this parameter included.

*Result*: The response received was a “401 Bad API Key supplied” error, and the request made no impacts on the database, showing that the API had made no alterations to the database in any way as a result of this request.

Successful

# Test 139 - DELETE Event (API Key, Acceptance)

*Description*: In this test, I will attempt to use the event method as intended, using a valid API key with a player who has an event loaded to skip the active event, without refunding any event stacks or providing any points to the player. This test will use the following input:

* API Key: N5DAXWGKLQEGXMMC

*Result*: The API responded with a “200 Skipped event if applicable” response, and successfully removed the active event of the player from the database without incrementing any other statistics or refunding any event stacks.

Successful

# Test 140 – DELETE Event (API Key, Erroneous)

*Description*: Following the previous test, the response from the API seemed to indicate it will provide a 200 code even when there is no event to remove. While this is not a serious error it is worth confirming if that is true, therefore this test will use the same inputs as the last test to discover what response is received when a player removes an active event when there is none available.

*Result*: It appears that the response code does refer to both a successful response and a failure response, revealing that the API will not submit a valid failure response when it cannot delete an event from an account. This problem is minor but could be worth changing.

Low Priority API Issue

# Test 141 – DELETE Event (API Key, Erroneous 2)

*Description*: This test will examine if the use of an invalid API Key will provide any changes to the database in response to a DELETE request being made. Ideally, the response will be the same as if no parameters were provided, sending a response that acknowledges the lack of a user tied to the supplied API Key.

*Result*: As anticipated the response received was a “401 Bad API Key supplied” response, and no tangible changes to the database were discovered as a result of this. This means that the API is correctly protecting the use of the command behind the API Key, only allowing users with a valid key to complete a request.

Successful

# Conclusion

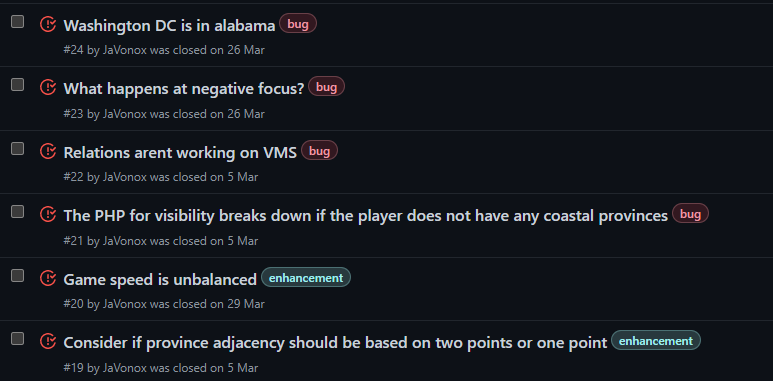
While the above tests may not demonstrate any changes as a response to tests with erroneous responses, it should be noted that many of the successful tests recorded here were things that had issues during development that were changed as a result of debugging. Throughout the course of this project, I held a private GitHub to store all the files for the web server, which I also used to record any issues that required fixing. As of the finishing of this document, there were 43 issues recorded on the GitHub repository with 42 of them closed. These issues primarily included bugs but also had some gameplay enhancement features as well as documentation requirements among them, and also only included problems I could not solve within a single session of development. As previously mentioned, 42 of the 43 bugs were fixed by the end of the projects lifespan, with the last bug being too obscure and hard to reproduce to test in any reasonable timeframe.

Figure 8 An excerpt of the GitHub issues log detailing many of the issues i came into contact with throughout development.