Statecraft Unit Test Plan 1.0
Project Statecraft

Introduction

When it comes to testing a digital board game, there are many things that must be tested before the actual project can be deployed. As this project is going to be built using the agile methodology and thus in increments, it will be tested by unit level as we go on. Below is a list of rules to be tested as they are highly relevant to what the user of our product will end up seeing. One of the things that must rigorously be tested is the game logic. Below is the list of features that will be tested in a sandbox environment to see if they are working as intended and by the rules of the game where they pass or fail depending on whether they are living up to those standards.

Game Logic (H)

Spring four-phase turn

1. Diplomatic phase
2. Order Writing phase
3. Order Resolution phase
4. Retreat and Disbanding phase

Fall five-phase turn

1. Diplomatic phase
2. Order Writing phase
3. Order Resolution phase
4. Retreat and Disbanding phase
5. Gaining and Losing Units phase

1. Diplomatic phase

During this phase, players meet to discuss their plans for upcoming Turns. In order to test thing in a more digital environment, we would need to be able to be able to create chat rooms where players can discuss amongst themselves strategies and what not. Several testing involving this would be ensuring only players that can see the conversation are those in the chat room itself, and we would need to ensure that messages are being sent correctly.

2. Order Writing phase

During this phase, players begin issuing orders to their units. In terms of game logic, this will require testing in terms of whether or not these orders or given during the correct season or not.

We also need to test logic for the types of orders given such as Hold, move, support, and convoy(More details on how these orders are tested in the section “types of orders”)

3. Order Resolution phase

This phase all players’ commands are to be revealed to all players and all conflicts must be resolved. Here all moves both successful and failure, standoffs, retreats, and disbandments. Testing will require that all all of those resolutions are working as intended(More information to how these resolutions should work will be described in types of orders). The board should accurately display an updated map based on the resolution of all phases.

4. Retreat and Disbanding phase

Here we would expect that units who are defeated will be forced to retreat to a nearby province they could ordinarily move towards. Retreats should be written down and displayed immediately. Retreats should only work in areas that are, not occupied,not a province from where the attacker came, and not a province left vacant during a standoff that same turn. In the event a unit cannot retreat, they are disbanded and removed from the board

5. Gaining and losing units phase(after fall turn)

Players check to see how many supply centers they control. A country controls a supply center when one of its units occupies that supply-center province after a Fall turn has been played and completed.

Once a country gains control of a supply center, it can leave the center vacant and still keep control of it, as long as that center isn’t occupied by another country at the close of a Fall turn. A unit that moves into a supply center during a Spring turn and moves out of it during the Fall of the same year doesn’t affect the ownership of the supply center. In short, a country retains control of a supply center as long as, at the end of each Fall turn (including retreats), the supply center is either vacant or is occupied by one of its own units.

If a country has fewer supply centers than units, it must disband the excess number of units (owner’s choice of which units)

If a country has more supply centers than units, it can place new units in each unoccupied supply center of its home country that it still controls. It can’t build units in supply centers outside its home country

Types of Orders

On each turn, each Great Power can order all, some, or none of its
units to do one of the following:

• Hold
A hold order is the default order given to troops which essentially tells them to stand in place and defend the territory they are currently in.
• Move
Depending on the unit there are several rules. Armies can't be ordered to move into a water area. No two units can occupy the same province at the same time, an Army that is ordered to move to an adjacent province can end up not moving at all. A fleet however can move into water areas but not land. These features will be tested by trying to make both legal and illegal movements with results being predicted accurately.

Now there are types of movements that are restricted such as any location on the game board that isn't named can't be occupied. These will be tested to make sure that the restrictions are working.

Many times moving a unit will involve a standoff. Our game shall be tested that the following rules are followed;

- Units of equal strength trying to occupy the same province cause all those units to remain in their original provinces
- A standoff doesn't dislodge a unit already in the province where the standoff took place.
- One unit not moving can stop a unit or series of units from moving.
- Units can't trade places without the use of a convoy
- Three or more units can rotate provinces during a turn
- provided none directly trade places.

• Support
A unit gives up its chance to move on a turn in order to support another unit's order. The province that a unit is providing support to must be one that the supporting unit could have legally moved
to during that turn. The pass/ fail criteria is if our program can follow the rules listed above and below:

- A unit not ordered to move can be supported by a support order that only mentions its province.
- A unit ordered to move can only be supported by a support order that matches the move the unit is trying to make.
- A dislodged unit, even with support, has no effect on the province that dislodged it.
- A dislodged unit can still cause a standoff in a province different from the one that dislodged it.

- Support is cut if the unit giving support is attacked from any province except the one where support is being given.
- Support is cut if the unit giving support is dislodged
- A unit being dislodged by one province can still cut support in another province.

• Convoy
A Fleet in a water province (not a coastal province) can convoy an Army from any coastal province adjacent to that water province to any other coastal province adjacent to that water province. Only Armies can be convoyed. “Support” can’t be transported from one Army via a convoy to another unit If Fleets occupy adjacent water provinces, an Army can be convoyed through all these water provinces on one turn, landing in a coastal province adjacent to the final Fleet in the chain. With convoys some tricky situations arises such as ;

- Dislodgement of a fleet in a convoy causes the convoy to fail
- A convoy that causes the convoyed Army to standoff at its destination results in that army remaining in its original province

Rare Cases:

- A country can’t dislodge or support the dislodgment of one of its own units, even if that dislodgement is unexpected
- An attack by a country on one of its own units doesn’t cut support
- Two units can exchange places if either or both are convoyed
- An Army convoyed using alternate convoy orders reaches its destination as long as at least one convoy route remains open.
- A convoyed Army doesn’t cut the support of a unit supporting an attack against one of the Fleets necessary for the Army to convoy.
- An Army with at least one successful convoy route will cut the support given by a unit in the destination province that is trying to support an attack on a Fleet in an alternate route of that convoy

As seen above, that is a rough list of game logic items that will be tested for accuracy and correctness. Each of those rules will be tested in a sandbox environment where we will have a board state that is created specifically to test that situation and see if they give us the expected outcome. If they do not, we will go back to the code where this is tested and potentially get the software working as intended.

Lobby joining and creation (M)

Before a player can play, we need to make sure they are logged in and see if they want to create or join a lobby.

1.1 Log in page
A page to log in to the game, the player will be able to enter his username.
1.2 Create a new lobby
A page to create and edit the options of a new lobby.
1.3 List of lobbies
A page listing all the current lobbies, and having the possibility to join them

In order to test this feature, we will create ourselves a user and check to make sure the username is being displayed correctly, we can create a lobby and edit the settings, and finally can we list current active lobbies with the ability to join them.

Server/Client Networking (M)

As the game is going to have potentially multiple people playing this game, we will need to establish that our server and clients are working as intended. The user will be accessing the game through the client and all of our clients will send the information to the server. The server will be in charge of keeping track of all the players and their actions and how they follow the game logic rules. As such there will be several items worth noting to be tested when it comes to the networking aspect of our project.

1. Authenticate players:

Here we will test that each player is given a unique token that will authenticate them. The server will receive a notification that a player has connected and the server will send a message back to the client that they are connected. Pass/fail criteria will be if the notifications for connecting are properly working which can be tested by just creating a user who will try connecting.

2. Chat:

As mentioned earlier in our document, the game will include a chat feature. The server will be in charge handling the chat and sending a message to the correct user. In order to test this, one user looking to send a message to another user, the message will first be given to the server which will then send the message to the user that is suppose to receive. This function will work similarly with a group chat as 1 person sends the message and the server ensures the right users get the message. Pass/fail criteria for testing this is just making sure the messages are being sent to the right people.

3. Multiple games:
The server should be able to handle more than 1 game at the same time as such we will test if our server can handle concurrent games as such if it can't then the test will be considered a fail.

4. Rounds Engine:
As the server is the one keeping track of the actions and logic, it is also the one that determines when a round is finished. A round is finished when all players have submitted their moves as such when the server is notified when all players submit their moves, we will proceed to the battle phase.

5. Battle Engine:
Continuing from the Round stage, this part of the server is in charge of determining the winners and losers of the round. This is done by following the game logic that was explained earlier in this document. As stated such, the pass/fail criteria for this test is ensuring that the logic is followed exactly and the winners and losers of the battle are decided correctly. In such a case, the losing side will have units either retreat, or destroyed properly such that they don't take up memory and are no longer displayed on the board.

6. Lobby: When a player attempts to join from the client side, the server will be responsible for letting players in assuming there is enough room to do so. The test for this feature will be being able to join available rooms.

Build Testing (L)

As this project is going to be built using the electron framework in order to create things like webpages, we will need to test certain features this will bring to the product. One would be building the server such that we can handle multiple games as stated earlier.

Deployment:

As this game will be made for the public, this game should be able to be downloaded for the public to use. Pass criteria is if the downloaded version of the client is the same for everyone.

Features Not To Be Tested

A list of features not to be tested as they are entirely low risk.

- A user can have a blank username.
- Create the Executables as this feature is can it run of different operating systems. As our project is going to be run on a webpage, it can be reasonably assumed if they are on a computer running an operating system with internet access, they can use this product

Approach

- As this is a project that will be built on a web page using the Electron Framework, ideally only software components will be tested as the project is meant to be versatile in terms of what hardware can run it.
- Regression Testing will be done in the event an update is made to a component, then a test will be conducted to see if it is working with other components and if it does not, worst case scenario is we roll back the component to an earlier build.
- As for testing features, some will be grouped together by functionality for example all the test cases relating to movement will be done together when that specific functionality is considered ready to be tested.

Item Pass/Fail Criteria

While in most of our cases, the pass/fail criteria will be whether or not it follows a set of rules and if it doesn't we would need to go back and fix it. Some things that might be a bit harder to tell.

- In cases where our problem is considered "game-breaking" those might need to be addressed right away as those items are going to be considered a fail.
- In other cases such as maybe some text might be cut off, those might be left alone in the event of time constraints and thus pass as they don't typically affect the overall project nor is it something that would be considered a major issue.

Suspension Criteria and Resumption Requirements

- Testing will be suspended and only resumed when we run into an issue regarding the component being tested such that it doesn't pass our criteria for pass/fail. In this case, we will need to go back and attempt to fix that problem before we can continue on with testing.

- As stated in the previous section of pass/fail criteria, items that are game-breaking will cause testing to stop until those items are fixed whereas non game-breaking defects will be evaluated and determine how severe they are and whether or not to stop testing all together.

Test Deliverables

- Test plan
- Test case specifications
- Test logs
- Test Incident Reports
- Test Summary reports

Environmental Needs

- Computer with the electron framework installed
- Potentially internet access

Responsibilities

Ideally the person responsible for testing a specific component will be the person who helped created that component as that person will have more knowledge on how that specific component will work.

Staffing and Training Needs

- Ideally, all the members of the group will have knowledge on the relevant software and programming language used to develop this product.
- It is not enough to know the tools needed to program the product, they also need to know how the product is designed to work. In this case they will need to know the game rules and how it is played in order to properly develop and test to make sure it is working as intended.

Schedule

As the actual development that begins after design will last roughly 2 months in total, ideally it should be planned for the implementation phase to be finished leaving roughly two weeks for testing when it comes to code integration, testing the final product as a whole so that issues can be resolved in a reasonable time frame.

- Generally, in terms of unit testing, it will be done as soon as the component is completed in an isolated environment provided there is still some time before a deadline approaches.
- In the event someone is having issues that may potentially cause delays, it will be advised that they move to another tasked while another person takes a look at the original task that is having problems therefore even if one task is experiencing delays, some part of the project can still be developed in order to minimize the overall delays in the product as a whole.
- In the event that all the features are unable to be developed by the end of the deadline, as a group it should be decided which features are not necessarily needed but just nice to have should be cut, or if we need to keep them and reduce and reduce the amount of time allocated to testing the product.

Risks and Contingencies

- Changes to the original requirements or designs means if something is already developed, it would need to be reexamined and if change is needed, than we have to test these changes to ensure it is still working
- Late delivery of the software, hardware or tools as any delay may cause the rest of the development to be pushed back and thus other phases in software development may either need to be reduced in order to meet the deadlines or an extension in the deadline.
- Delays in training on the application and/or tools. As everyone will need to know the relevant software and programming languages associated with development, any delays in training will mean the development process may take longer as members will need to be caught up to speed.

While delays in development are unfortunate, in the event it happens, other areas may need to be shortened for time such as a reduction of scope may be needed in order to allow adequate time for testing and deployment by the deadline.

Approvals

This will be specified when all parties are completely satisfied with the way the component works thus we will classify it as being product ready and allowed to be in the final version of the product thus allowing us to move on to work on other areas of the project. As this is a team effort, we will need a general consensus as to whether or not a component is working the way we intended.