H446 Programming Project Report NEA

Name - Abigail Hinchliffe
Project Title - Control
Centre Number: 16327
Candidate Number: 9384
Date project started – 08/06/2022
Date project finished – 04/05/2023

Analysis: 4
The problem 4
Computational Methods used: 4
Stakeholders 5
Stakeholder interviews 6
Appropriate features (problem research) 9
Limitations 13
Requirements 14
Success Criteria 15
Design 17
Algorithms 17
Subroutines & Diagrams 18
User Interface Design 18
Usability features 24
Database Design 27
Validation 28
AI design 29
Game Mechanics 33
Key Variables, Data structures & classes 39
Test Data & methods 41
Iterative Development 46
Evidence of Development & solution 46
Test 1: Setting up my Code – Cursor feedback and GUI 46
Main menu design 47
Test 2 - The GUI can be closed with a 'quit' option & Test 3 - The GUI display buttons are shown upon execution, and they interlink with one another 47
Stage 1 52
Test 4 – Timer Mechanics Prototype version 1 52
Test 5 - The map is able to load, with all the buttons in the correct locations 53
Test 6 - The buttons are coloured accordingly upon distribution 55
Test 7 – The troops are distributed accordingly from setup 58
Test 8 - Timer prototype version 2 & 3 61
Test 9: The timer, upon reaching 0, goes into the next phase's timer 64
Test 10: When a new phase is executed, the new rules are met with the previous ones being overruled (Deploy) 65
Review of Stage 1 80
Stage 2 81

Test 11 & 12: Attack phase & Dice Mechanics 81
Test 13: Transfer Mechanic & Fortify phase (Failed prototype) 91
Test 14: Transfer Mechanic & Fortify phase 93
Test 15: The game can be won upon elimination of opposing territories 97
Test 16: Settings – Colourblind settings & rules 99
Review of stage 2 100
Stage 3 – AI development 102
Test 17: Statistical probability 102
Test 18: Adding more territories and ai deploy 104
Test 19: AI attack & AI attack validation 106
Review of Stage 3 109
Evaluation 111
Testing to inform evaluation 111
Evaluation of Solution 117
Further development 119
Success criteria for usability 119
Limitations of program 121
Potential for program improvement & Avoiding limitation 122
The finished code 123
NEA main project.py 123
Classes.py 150

Analysis:

The problem

One of the many issues with classic world domination board games such as RISK is the large portion of time required to sit down and play it, which can take hours, and in most extreme cases, days to play out. This project aims to produce a digitalised version of territory driven conquering games and can be suited to a range of people spanning from those with social limitations to those with lower amounts of free time. This project requires only one human user, meaning that additional players are not required to play the game to its fullest capacity.

The features I hope to add to this project will have many benefits, including reducing processing requirements through abstracting details like moving all the physical pieces into one location visually by representing troops with numbers; to performing all calculations via the computer, which eliminates human error and saves time in carrying out battles. One such example of this is the dice function, as an automated dice rolling system will save a user the effort of manually carrying out multiple attack attempts at once, which will allow the game to flow quicker.

Due to the amount of physical space required to play a traditional board game, it can prohibit the use of an area for a prolonged period of time if left uncompleted. The ability to save game data on a digital version will be useful, and will allow a user to continue the previous game at any given time, without having any side effects on day-to-day life and the overall position of the pieces, as no pieces can be knocked and affect the overall standing of the game.

In addition to this, a computational solution would reduce calculation time, as all math-based functions such as probability & troop bonuses are run by the computer, which ensures that the player gets a more efficient experience.

To describe this project simply, it is a game where the player must occupy all/most territories by defeating the enemy player in a series of conflicts driven by different phases in the game: Deploy, Attack & Fortify.

Computational Methods used:

In the making of this project, details have been abstracted to create a simpler model that fulfils its purposes and does not perform more than is required or is necessary to reduce the requirements needed to successfully run this program. A list of details which were identified to be optional and therefore excluded from this solution include:

- Piece Animations
- Individual dice rolls & dice roll animations
- Physical representations of the pieces on a board (this can be managed by numbers alone)
- The option to zoom in/out the map
- Water animations for the map boundary/ocean

Other issues that have been identified in creating a structure for the game include (and some possible solutions):

- Having clickable abstract shapes:
    - An invisible ‘hitbox’ is created around the shape, which a player may click to act as a button when selecting a territory.
    - A colour-based system, i.e., red = enemy, blue = player so if the player cannot place on red marked locations during deploy phase (however, this poses issues as when the territory changes ownership it will have to change the entire ruleset of that location immediately)
    - The shape itself is recognised upon clicking, but this can only be simple shapes with various co-ordinates.
    - A small button that brings up a pop-up window dependant on the ongoing phase
- Following set phases with unique rule sets
    - Each phase will call upon the next once completed
    - Certain functions will be prohibited for different periods of time

- Showing a pop-up on an attack button with statistics ie win probability.
    - When the attack button is hovered over, statistics on how probable it is for a player to win a battle with the current troops on either side, based off the likelihood of one or two dice rolling higher than the defending dice.

Factors that are expected to influence others:

- How many troops attacking vs defending will influence the win probability of the dice and therefore the attack
- The troop bonus is expected to influence the decision making of the ai, as one location could quickly become heavily fortified within several moves depending on how many are to be added. A higher troop bonus also decreases the likelihood of a player losing control over a continent.
- Card hand-ins are expected to influence the volume of troops on some locations, especially if pre-owned upon usage

Factors which must be used concurrently:

- Timer and player moves (timer must still move down regardless of a player's actions unless paused)
- Checking ownership of the territory & phase rules – i.e. what can be attacked but is now conquered cannot be attacked again unless if by the opposing player
- Saving game data whilst still loading the objects in the current game
- AI calculations involving comparing multiple locations
- High attack VS high defence troops requires the simultaneous comparisons of high rolling dice

Stakeholders

The stakeholders for this project will be users of a computer, most of the people represented in this demographic are casual gamers or users of a specific demographic. Under the assumption that each user is able to use a keyboard and mouse, users who do not intend to play the game itself will either be providing feedback on usability functions or trying to find ways to intentionally disrupt the game.

A suitable topic to research for this project is to use stakeholders to analyse the average time taken to play a standard game, this would help to improve usability features suited to a person's preference, as a digital version to play as opposed to a usually long-winded board game may benefit those with shorter attention spans or those with difficulty concentrating. By examining the average concentration time of these users, it may be useful to implement a fast-paced game feature to cater to these players, this could be done by adjusting the timer on each phase of the game to a player's needs, such as reducing the turn counter from 60 seconds to 40 seconds, so that they may enjoy a quicker game. In general, collecting information on this would help to add usability features.

Another group of stakeholders involved are those which have different ranges of colour vision; this is to help choose a new colour palette for users of various forms of colour-blindness, so that the visual representation of the game is less strenuous on the user. This will also allow better distinction between the different sides, as ally and enemy sides have contrasting colours, and this will make the game more versatile to its audience.

Other stakeholders which I will be reviewing is a mix of teenagers, aged from around 13-18, this will be done to allow me to examine what strategies they use – if any; this would be useful in adapting the AI's decisions based on the average/ most common playstyle of the user. In doing this, I can also produce data on the average time taken to play a standard game of my project, and this can be used to decide on how long the timer needs to be at a default. This may also help to produce graphs on the patterns of playstyle; as these may show if a user is predictable. After reviewing the stakeholders, they will complete a questionnaire so that I can gather feedback.

Additional stakeholders may stretch to individuals who will find ways to exploit and abuse the game, as their feedback in finding any loopholes in the game rules or their feedback on their method of attack will help with

testing; although no individuals will participate in the actual development of the project, data gathered in both verbal and documented feedback may help to patch the project.

Stakeholder interviews

When interviewing a group of stakeholders, the questions included in the surveys for the separate groups included:

The responses to this survey are recorded, but are anonymous, to protect your privacy in collecting this feedback.

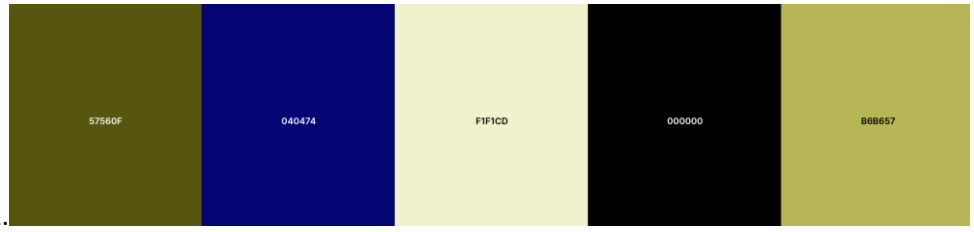
Colourblindness

Q: Type of colourblindness?

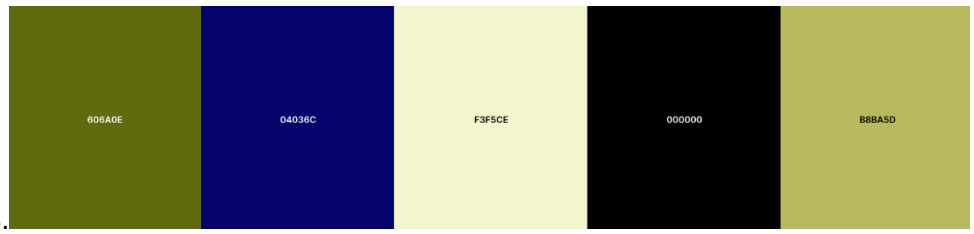
Responses: *Tritanopia, Deuteranopia, Protanopia*

Q: Compare the following palettes to the unchanged version. Which one, 1-8 do you find most contrasting to you?

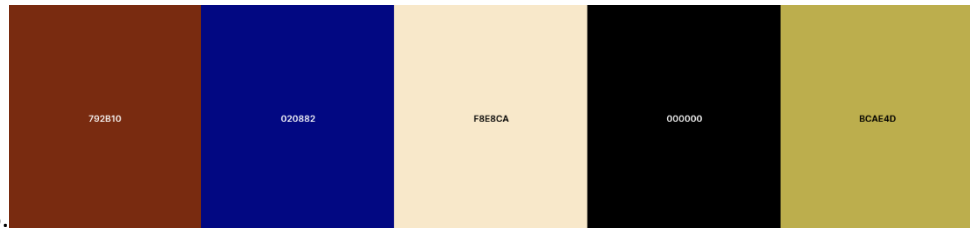
Example

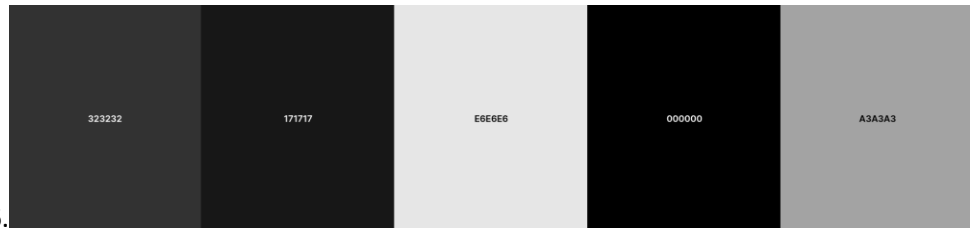
1. 

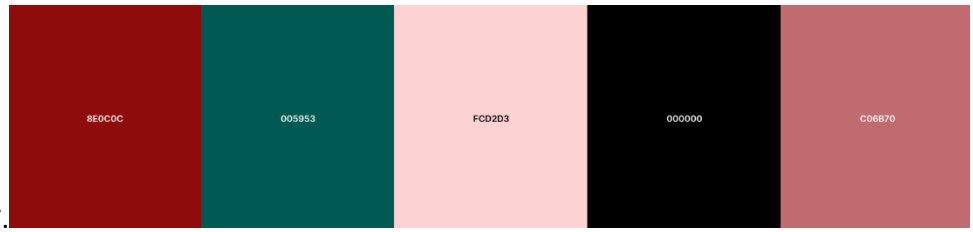
2. 

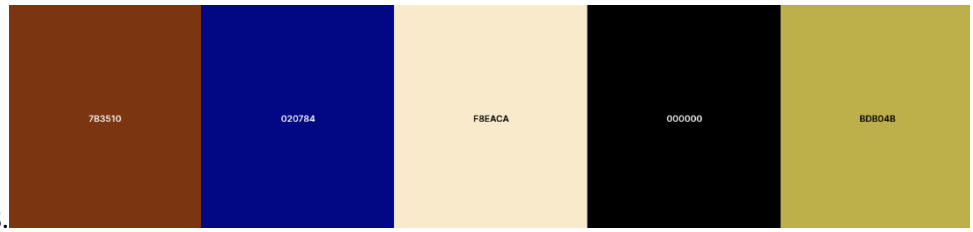
3. 

4. 

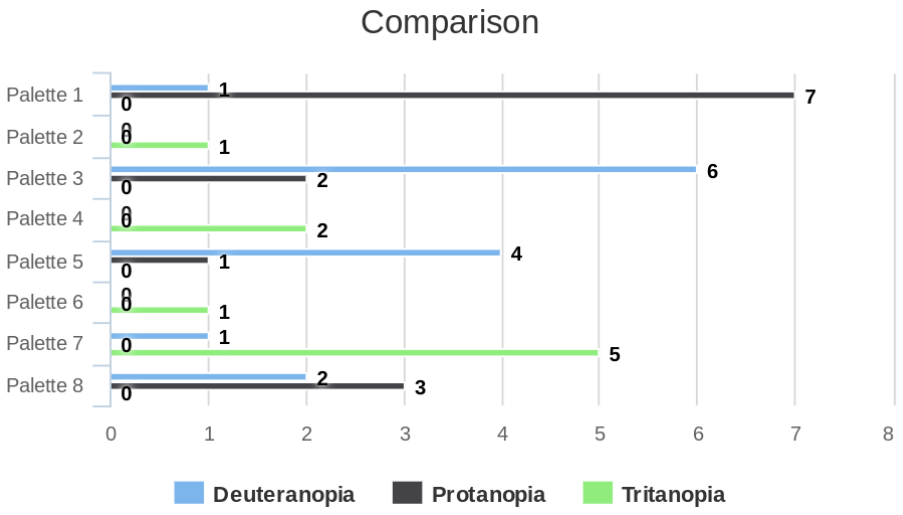
5. 

6. 

7. 

8. 

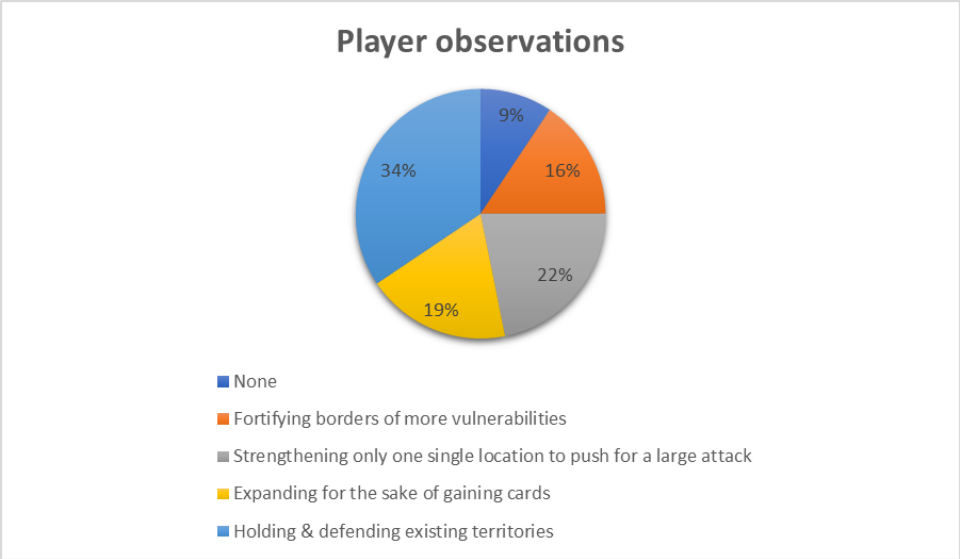
Results from each category:



This bar chart shows that the colour palette most suited to protanopia users was palette 1, with palette 8 being a contender. Results which scored high in Tritanopia often did not fare so well with other results, this could be due to the distinct difference between this variant. High scoring Tritanopia was found to be palette 7 by quite a significant margin, whereas protanopia scored mostly on palettes 3 and 5, therefore it would seem wise to assume that using the palette with the highest score for that group would be beneficial.

Game players

After reviewing a group of 17 students, it has been found that there are specific patterns in the way people choose to play strategic games, this was carried out following an assessment of observing how various people played RISK. The results in strategy tactics observed is as follows:



From this data, it can be gathered that nearly 1/3rd of people chooses to strengthen and improve their hold on their existing territories before pushing forward and attacking, so this data could be used to get the AI to try to counter this attack, for example if all the players' deployable troops went into one location, if the probability of taking it before it builds up is greater than 50%, then it should try to wipe out the enemy before it has fully strengthened for attack if applicable.

This could also be used to calculate if it is worth trying to wipe out the opponent on a territory if its surroundings are also being held, as a split-up army of equal sizes would be more of a risk to try to destroy in one go and targeting one could sabotage the AI's chances of trying to prevent a large army in a small area from forming.

Similarly, it appears that 50% of the people reviewed preferred to strengthen specific locations as opposed to strengthening territories with large numbers of enemy borders/ weak points, this could be used to the advantage of the AI as this may be used to attempt to blockade the player from entering a different territory altogether.

In researching menu design, I sent out the email:

Hi all, attached is some diagrams of what my user interface should look like, as well as the menu layouts. To some degree, I have tried to make the buttons large enough to be easily read, however, what features, or changes would you suggest to make the program more universal?

These are my most useful responses -

Response A:

'The buttons appear large enough to be easily read, however, it may be useful to have an option to change text sizes if this is such an issue? Personally, I have no problem with the layout of the menu as the order of settings and options seems justifiable although you may want to consider having a more neutral background colour.'

Response B:

'The layout is very simplistic – this is good for putting a direct focus on the game, but the buttons don't need to be as central as they are, and you could probably do with having them a little more spaced out, also, would be even better if you would add a 'back' button into the top-right or top-left corner so that it is out of the way'

Response C:

'If you are planning to add the login feature as you previously mentioned in an earlier survey, wouldn't it be beneficial to add small extra features, for example, showing the player's username, perhaps at the top or a corner. Also the text size is fine, but the colours of the buttons themselves are in places a bit difficult for my dyslexia to interpret, perhaps a highlighter for the text or something like that on a white backdrop may be better.'

Response D:

'How are you planning to be able to exit the menus if there is no back button? More neutral colours may be a safer way to go so that there isn't a sharp contrast in colouring, and interchangeable text sizes would be good'

Appropriate features (problem research)

Existing Solution1 - Axis & Allies 1942 Online

This solution has a diverse set of rules compared to my proposed plan; the game follows a sequence of stages with 5 different factions (Which makes up 2 different teams). These stages may prove useful to help set out a structure of rules for my game to follow. Although this is an overly complex version of a solution, it has 2 overall teams made up of 5 factions. My project will only have 2 players. Another detail from this example that differs from my project is multiple troop types; despite its uses in this game, it would be too difficult and confusing to follow if I added multiple combat options alongside fighting on territories. This would require me to add more spaces, making it infeasible to code into the game at my level; the development of unnecessarily complex mechanics would defeat its main function: to provide as a simple mechanic-based strategy game that would reduce time taken to play.

A smaller, simpler map than the one shown above would also significantly reduce processing requirements, and reduce the time taken to load up a new game. A game that played on a smaller scale would decrease time taken and would require less variables; although the classes of each location would still need to follow the same degree of rules to function.

The layout of this solution is an interesting idea; as it limits the amount one player can do in a turn – so that a player cannot abuse their turn and make it infinite; the stages will also make it more feasible to make both the AI and locations follow a specific set of rules; this will also allow me to change the parameters at different stages.

An idea to collect from this is maybe an option to have a singular device mode; where two players or more players (could be a viable option if no AI available, however will require additional settings) take turns in using the same device to play against each other.

Parts that I can apply to my solution

Stage/ turn based rules to follow – A timer would offset at the beginning of a round, and only certain moves will be available for that time frame until the time reaches 0. Upon reaching this, the next phase will begin, adding one to the turn counter and changing the set of rules laid out. When the turn counter reaches 3, the GUI will no longer be able to be interacted by that user as the turn switches player, resetting the turn counter to 0.

Interacting with the area will also cause it to be highlighted in that player's colours – this is to enable a distinction between two player's territories. This could be done by button or by filling a transparent image position on a specific co-ordinate with a colour that is related to the player that owns the location.

An end phase button is also useful for my solution, as it would enable a game to be played at a quicker pace, if a player wants to end their turn at an earlier stage, then this can be done by pressing a button that resets the timer to 0, ultimately triggering the next phase.

Existing Solution 2 - Conquest of the empire

Despite this solution having no existing online version, there are some concepts from this board game that I would like to note down.

This solution, similar to others such as diplomacy & age of conquest, is a world conquering game with variations that deviate from simpler games. The game follows a turn-based structure, as seen in most of the examples of existing solutions; players must conquer other provinces with their legions, for which they exact a tribute at the end of their turn. These tributes are used to pay for military expenditures. If a player wishes to add a city to a province alongside its troops, this provides additional tribute and fortifications, which provides a combat advantage in battle. Players may only collect 'tribute' when still in control of their home province. If a player has captured another player's home province or Caesar, a bonus is awarded.

In relation to my project, the idea of awarding a player for defeating an opponent may be useful, for example, if a player defeats another player against their strongest army, the opponent must forfeit any cards which they own that have listed locations owned by the opponent – this could prove to be a gamble which players must decide to take – as this could range from no benefits gained(and thus, only weakening their army with little to no gain) to an advantage. This may also ensure that if a player shoulders a risk, they may be rewarded.

Another feature of this solution that is unique is that roads can be built between cities of neighbouring provinces, which allows troop movement to any part of the road network in one singular turn. This is also an additional risky option, as the opponent player will then be able to take multiple locations at once if the player refuses to destroy a city to sever the road network; preventing the enemy from gaining further 'tribute' which increases their power.

In my solution, in the movement phase, I may limit the movement of troops to only be able to transfer to different territories that are physically connected, this way, the armies are all essentially linked in at least one way; making it easier for players to provide additional support on a front that is at risk of attack.

Parts that I can apply to my solution

Exacting a reward for the player depending on the volume of conquered territories per turn for example continental bonuses or additional reinforcements would be useful, for example, every owned territory adds a value of 1 to a counter variable such as 'owned_territory_counter', then the set of parameters could be checked inside a main loop so if this counter reaches a certain number, it adds troops for the next commencing round (for that player).

Alongside this, distributing a card to a player when they have taken a new territory on their turn will help run a card mechanic, if enough cards accumulate for the player, it can then be used to be traded in to help add to the available troop counter at the start of the turn, likewise, if the value of the card happens to be a territory that the player occupies, then an automatic addition of troops will be used.

The fortify phase can also be adjusted, so that the troop transfers can only be applicable to a series of territories that are interconnected, as transfers between remote locations should not be viable, this would ensure that the player must also place troops and invade strategically, as a location that is cut off from supplies from other territories would therefore have a vulnerability for the enemy player to exploit.

Existing Solution 3 - Age of conquest online – optimised for android

Like many other solutions, the requirements that need to be met in order to win the game are simple – the player that reaches 100 points wins the game. The game itself is far too large of a scale and complex to attempt to replicate, with far more than 10 players fighting one another in real-time, however there are a few features that I would like to incorporate; although I will not try to replicate the mechanics for continuity and difficulty reasons, the overall game actions are clearly displayed on screen, and the design for the user interface appears practical.

The interface for this game is unique; the idea of buttons instead of square panels may prove useful in fitting all user accessible options into one space. Taking inspiration from this, pop-ups may be necessary for optimisation purposes, as the tabs will hide and reveal themselves as an overlay when clicked. Another feature I like about this is that the events are shown clearly on what is currently taking place; for example, a sword image is located on a territory currently being attacked, and this icon allows the player to follow the game easier. Although the map of age of conquest is large; the ability to zoom in on a location and see the mini map in the top left corner allows players to play at a map size more compatible to their playstyle; some users may find this easier to follow, as a zoomed in map may mean that they are taking in the details at a smaller scale; whereas some users enjoy looking at the fuller picture.

Despite the uses in being able to zoom into a map, its practicality is limited, and will probably not have much use in the project, unless prompted by stakeholders, although, the idea of icons marking active conflict like attack and defence may be useful in tracking the moves of another player, and the small icons marking the edge of the screen may be a useful option in providing additional features.

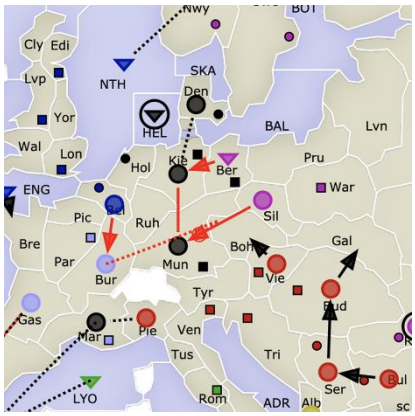
Parts that I can apply to my solution

The GUI display shows buttons on the screen, which could be used to enable certain actions/ activities for the player. As previously mentioned, this could include an 'end game' option, with a notifications bar to follow the gameplay (useful during testing and also for players), a button to verify location selection, timer display, trade-in, save data and return to the main menu.

A pop-up also appears upon location selection, and I could implement this to be able to record and display statistics and options to the user, such as win statistics.

Existing solution 4 - Diplomacy online

This solution is unique in its main objectives and focuses (with some locations being unplayable, such as Switzerland) - the aim of the game is to win possession and maintain control most of the most strategically advantageous cities and provinces marked as 'supply centres' on the map. These locations are beneficial as they provide the ability to produce more units; unlike other domination games, the troops are not given automatically at the beginning of every round, and instead must be gained by careful placement and priority. The number of supply centres a player controls determines the overall number of armies and fleets a player obtains and has on the board; once players lose control of these centres, they may raise or disband units correspondingly.



The game follows rounds of negotiations, where each player can issue an attack and support orders, which are later carried out in the movement phase; the provinces are then taken or maintained based off how many attack/ support orders are issued to that location, if the number of supporting orders is greater on the attacking province than those issued to the defending district, the location is controlled by the attacking force.

The lack of dice & the focus on player alliances and betrayals in its negotiations rounds reduces the consequences of random effects from additional factors such as wildcards and strength, making the game more heavily based on mutually beneficial strategies between players.

Factors from this solution that I would like to take note from, is the hierarchy of separate locations – all holding separate significance from one another in diverse ways that benefit the player. Ways I could incorporate areas of significance could be to provide an additional troop bonus on areas with a disadvantageous location – where the possibility of invasion from multiple borders is high.

A more simple way to include something akin to this may be to add a card to a player's collection each time they claim a new location, as the more they take, the more troops they may be able to gain as a bonus even if they suddenly lose those locations in the next round.

Parts that I can apply to my solution

The vulnerability hierarchy of a location could be applied to the AI's decision-making processes, a series of functions of comparisons could be triggered under various conditions, to ensure that a point with more points of attack is not fortified with a menial number of troops – as the attack will be rendered fruitless when the player attacks it again. This will be necessary to also help strengthen the defences of other locations, and provide a greater troop movement pathway.

The use of arrows for showing the attack pathways would also prove useful, as the player will be visually directed to the only viable places to attack, showing the distinction between the factional occupation between neighbouring territories.

Existing Solution 5 - Risk (2 player game version)

The game of RISK is the main inspiration for this project; it has many example mechanics that I could implement into my solution. Namely, it's two player rules involve a third, neutral, faction. This faction is not allowed to attack and

can only defend and increase its army. The faction gains half the troops the player receives at the start of the round, and these will be randomly deployed across the game.

Although the idea of having mission cards is a great idea for more than 3 players – as seen in the traditional 3-6 player game; it's addition to this project would be meaningless in the face of already occupying most of the territories. RISK cards, which I may loosely follow the idea for, is like an additional card mechanic that is proven to be random, (if carried out on computer so that it cannot be tampered with or biased towards the shuffling player). The cards themselves contain unknown benefits until combined with similar cards alike. This will help add a dynamic source of troops, as it may help struggling players gain significant troop bonuses once exchanged. Like RISK, these cards may be gained upon capture of another territory. A difference to my solution and RISK's 2 player board game version may be that the player starts with less of these cards (Typically it would be a three-way split), from which they may accumulate at random throughout the game. If the card, once exchanged, has a territory that the player owns, then a bonus should be added to the defence of that territory.

When the card trios are exchanged, the number of times this is done throughout the game by that player increases the number of troops they gain once those cards have been expended. This could be added by having a counter for each player that adds one to the total when each set has been exchanged. This could then just be checked against a list of pre-indexed values spanning from 1-10.

To attack, the dice rules would also be useful to follow, as this would make the mechanics more dynamic, and give attackers the advantage in pushing forward. If the attacker attacks with 4+ armies, they roll 3 dice. Likewise, if the defender has 3+ armies defending, they roll with 2 dice. In conclusion, RISK's 2 player game rules may be a good indicator for structuring the rules of the project, as well as some of the mechanics and gameplay, due to its more simple nature than other the other solutions, this makes it easier for players to follow and puts restrictions on making the project overly complicated (which would inadvertently become counterproductive for its intended purposes).

Parts that I can apply to my solution

A neutral faction will be useful for my two-player solution– the rules behind this entails the faction to not be able to attack surrounding provinces but it can defend itself. An additional variable for neutral available troops will be a counter that is half available troops, which can then be added equally to each neutral location.

A card mechanic system will allow a degree of randomisation to be present, which will help to mitigate the chances of the AI being so logical that it is rendered unbeatable by the average player. When the matching cards are added up, the value added accumulates over the course of the game depending on how many times this action is performed. A list of values defining this would be useful, so when a timer hits tis counter it exacts that amount onto the player (and it is not the same for each time).

The dice mechanic will be good for applying attack mechanics when working in conjunction. If the troops on one side is a specified amount, such as attack with a location with 2 troops, only one dice will be used, whereas if the enemy defender has 2 troops, they have 2 dice and therefore the probability of any of those dice rolling a higher number is higher, increasing the win probability in the favour of the defending force. Likewise, this caps at 3 dice for attacker and 2 dice for defender, so that the attacker can always (to some extent) have an advantage over the defender, which presses forward the importance of strengthening existing reinforcements.

Limitations

Considering the above solutions, I have produced a table on details that I will not be able to carry out, and why.

Feature	**Why is this not viable?**	**Would this have added anything to the program?**	**Are there any conditions in which this may be considered?**

Multiple teams	More than two players means that I would have had to either add a multiplayer function or create an additional AI player. This AI would end up competing too heavily against the other AI, making it near impossible for the player to effectively win.	This would have exponentially increased the time that it takes to make the project if I was to make it an equally fair opportunity for the player to win. Overall, the addition of another player is ultimately	If a pass-the-device mode was added, as it would require an iteration of the same rules, just repeated; with the only variations being selecting how many players and the overall map distribution.
Points of interest	To introduce points of interest will mean that I will have to create a separate set of rules for locations to follow	This would add unnecessarily complex levels to the game; it will be performing more than its primary function, which may become less user-friendly	If I wanted to make the program more dynamic and unique than current existing solutions.
Mission Cards	Only two players means that the map will be evenly split, so the point of mission cards is futile if a player starts with most of their target continent	This would have been better for games with more than 2 players, as the game stands, this would have been a waste of time	More than 2 players required, or the addition of a neutral team.
Risk Reward System	This would include a lot of information to include into my solution	This would have made the game more dynamic in the case of some players, as it rewards risky gameplay. However, this would have made people more unpredictable	If playerxtroop =< playerytroop and wonbattle==True: Where playerycards = playerxterritorymatch change to playerxcards But this is too complex to fulfill the core objectives of the game.

Requirements

In order to run my program, the software requirements for the user will require an application or OS that allows the running and compiling of python programmes, for example Windows XP or any versions above, as my game will require python & its various modules such as pygame & SQL. Connector. Examples of software that allows the

running of python include Python IDLE, Visual Studio Code or Notepad++. Additional extensions may be required to run additional modules.

Sufficient hardware requirements to run my game will be a 1.6 GHz processor - as the game is a fairly lightweight program this should be more than enough. The minimum requirement of memory is 1-2GB of RAM (Random Access Memory), and even then, this should be more than enough. Due to the contents of the program, the requirements of the graphics are incredibly minimal so any graphics cards compatible with DirectX 9 would be suitable. To interact with the program, the user is required to connect a keyboard and mouse, as the proposed solution will not be applicable to android or IOS users. Most of the controls will be based off interactions with different buttons/icons in a Graphical User Interface.

Success Criteria

1. **An on-screen clock should be displayed on screen**

I would like the solution to have an on-screen clock that counts down from 60 to 0 per round and 30 to 0; with 1 second increments and once the time hits 0, the round ends and the player can no longer interact with the GUI until the next turn is reached. The total time per turn must be at least 90 seconds; (from 60 to 0 repeated 2 times) clock is reset once it reaches 0, setting up for the next phase. This could be achieved by using the pygame timer method with get_ticks() and further subtraction, represented by text in the top-middle of the screen in size 12 Calibri font. A timer puts pressure on the player to finish their move, and it also lets them know how much time is left of their turn. It also speeds up the time of the game – to prevent the game from lasting for days – to make the game worthwhile to code. I am aiming to complete this inside of 3-4 hours, to make sure that the function is completed & working under different conditions and test to see if it is enough time for a player to decide their move.

2. **I would like the game to follow a turn-based structure**

Only one player can perform their move at a time. A display could be added to the clock, where the current phase is highlighted to inform the player what phase is currently running so that they know what rules are being followed. For example, if the round is shown to be fortify, a player will know that they must deploy troops, but they cannot attack another player just yet. A phasing system would ensure that the player can only follow a specific set of rules in a given time. For example, if 'timerfortify' clock reaches 0, set to False, while 'timerfortify' = False, Set 'timerattack' to True. Making the game turn based ensures that the AI cannot make moves too fast for the player to react or comprehend. Giving the game a turn-based structure would make the game more controlled. This is expected to take at least a full day to implement and should be done at similar times to the clock countdown function.

3. **The AI should follow all pre-set rules of the game.**

The speed at which it makes/ ends its moves must be within a certain set of parameters, yet it cannot be too many/few for the player to comprehend or react to. All standard rules like available routes, piece starting, time limit etc are limiting factors in what the AI can do. Time. Sleep() can be used in between each of its moves so that it cannot make as many moves as possible in the available time. This makes the game fairer for the player and ensures that the AI player is incapable of cheating the rules of the game. It also provides a layer of simplicity so that the AI is beatable. This may take a while to integrate into the game, as the AI must act within accordance to the game rules, and a lot of testing will be required to fully meet these standards.

4. **Territories must be divided equally, randomly and fairly.**

Each territory when set to true would signify it is owned by player 1, if false it is player 2. When set true the border can be made to change colour to signify this change, so that a player is aware of all territories they own. Importing the 'random' module and applying it to the list of countries, this could be done by iterating a set number of times, each time a new value not matching the new list is added, 1 is added to a counter variable until the counter has reached half the number of territories available in the game. Adding this would ensure that the game would be able to distribute different territories so that each game would have separate starting positions. This adds a dynamic layer to the project and helps ensure that the AI player has more parameters to consider. This is expected to be one of the first things to add in the game and is expected to take a while to fully implement, as it will need to be random to an extent. Ie. If a major continent is set true for a player, then an unfair advantage (by chance) has been given, so it would have to reset the list and repick the territories.

5. **I would like to add a button that can skip phases and end turns prematurely.**

Apart from the deploy phase, if a player has finished their attack or fortifying before the countdown reached 0, then they should be allowed to finish their turn. A button on the bottom right of the screen can set a condition to true when clicked, this can stop a function of the phase mid execution. This would be suitable for players with time restriction, as it ensures that they do not have to wait the full time if they finish their move early; equally if the AI has finished all its moves. This should be a quick addition, as its sole function is to appear during certain phases and to halt current phases to continue onto the next + set clock to null.

6. **It would be suitable to add a notifications box.**

For players who cannot follow the movements of the enemy player or the game. Ie. How many troops they have been granted at the start of their turn, it will allow them to read what is happening. This could also be added to the pause menu and would be suitable for usability, as features such as font size or box colour can be suited to the user. A text box should be present whilst the game is being played, this could be done as a button that can pop-up the box to not interfere too much with the interface. This helps to keep track of the course of the game and acts as a multi-purpose tracker. This allows for usability options that can be tweaked to suit the user. This will have to be added to throughout development, as I am able to integrate more features into the game. This is one of the final things expected to be completed.

7. **The game should be winnable against the AI.**

Each move by the AI should be controlled to prevent too many turns at once, each turn cannot be the best possible move, as it would become impossible for the average player to win. From each move, a timer between decisions i.e., 5 seconds is good, with the AI still being on a timer. The decision it makes can be ranked in an open list, and the #2-8th best decision ranked should be the one picked. This allows player 1 to be able to win the game against the AI – allowing the game to be finalized as functioning and playable. This can only be achieved from testing the AI throughout development. It is not expected to be able to fully perfect the logic it takes, but a basic design that is beatable is the aim.

8. **The neutral force should be powerful, but not unbeatable**.

For each of a player's turn, the neutral force gains half the amount of reinforcement that the player gains. These will only serve as a defence of the neutral territories. This could be achieved by declaring two variables such as 'neut_troop' and 'current_addition' so that the neutral troops added with be half that of whichever troops are being

added, irrelevant to whatever team is adding it. This will allow the player to still have an advantage over the neutral territories, making them winnable, yet, also causes the player to think about when to attack. This will require test games to record how powerful the neutral territory grows over the span of the game and this test will help gauge to what extent large attacks affects the probability of winning a siege.

9. **Saving the game should be available from logged in users**.

When a user logs in, the game that they are playing will autosave onto a database, so that when they exit, their save game code transfers across Using 'MySQL. Connector' module in python will allow the program to read/write latest information onto a SQL database, as well as a gameID which will check to be updated every round, replaced and logged. This solves the problem of closing it and losing all progress, which is useful as a user can take a break or switch tasks. This will only be viable if there is an appropriate amount of time available. If not, then this will most likely not be possible to meet; this will also require multiple testing against SQL injections, database logging & drop table as well as input validation.

10. **A simple menu should be used to link together features of my project**

To be able to access and make use of my project and its features, a menu GUI will be used. The idea of the menu being simpler will try to prevent the formatting from becoming overcomplicated and will just put emphasis on the actual quality of game itself. The purpose of this menu is so that everything can be in one place, and additional features will therefore not require any extensions.

11. **The game can be quit at any given time, with the use of a 'Quit' button**

To ensure a user does not find themselves stuck with a window that will not close, a 'quit' or 'back' button will be available on every screen, including the main game screen. The placement of the button will be near the top of the screen, but out of the way whilst remaining clearly visible to the user.

12. **A card system should be implemented into the game**

To add more troops to an ally, the use of territory cards should be used to add a bonus to the occupied territories, as well as matching pairs. The cards can cycle through at random and have two values of interest: one which will be either Soldier, cannon or cavalry, as well as the actual territory. This will be validated against one another, with the user only being able to trade in one type at a time

13. **Validation should be used for the login system**

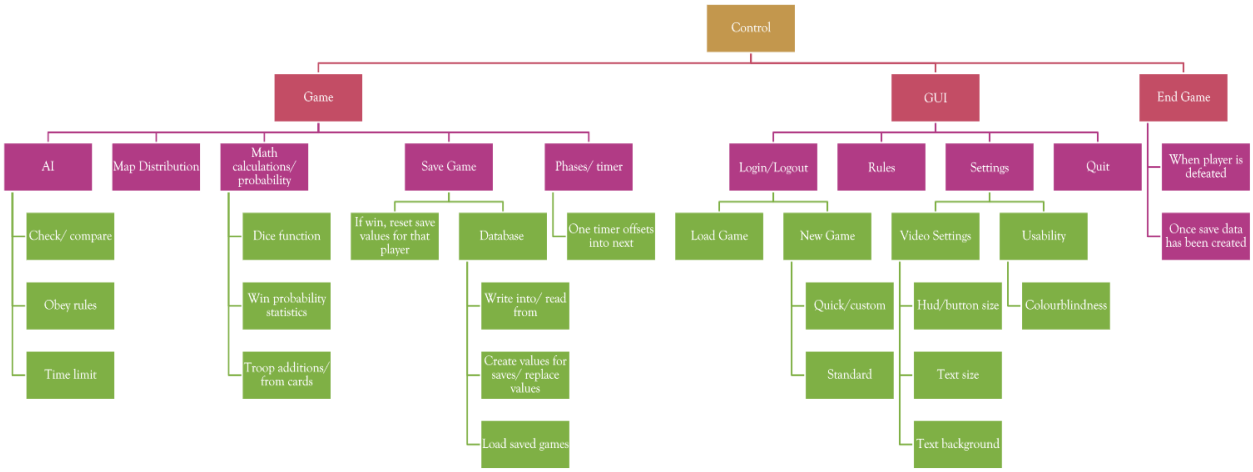
The use of validation to check for appropriate usernames as well as various data types will allow me to plan against potential misuse, this can also include attempts to maliciously delete locations in the database or gain any form of access to the database entirely. For example, FROM Users DROP table or SELECT * FROM Table.

14. **The dice function & probability statistics should be in accordance with one another**

To run the statistics, this can be done with a pop-up on each location, which compares the amount of dice, as well as the probability of the dice winning, whilst taking into account the number of rollable dice, as well as the chance of a higher score being played. Addition features such as the 1v1 mechanism must ensure that a location breaches the base game rules of a minimum of one player on a territory at a time.

Design

Algorithms



The diagram above shows I have split my plan into three components; that of the actual game design, GUI design & the procedures to end the game at any given time.

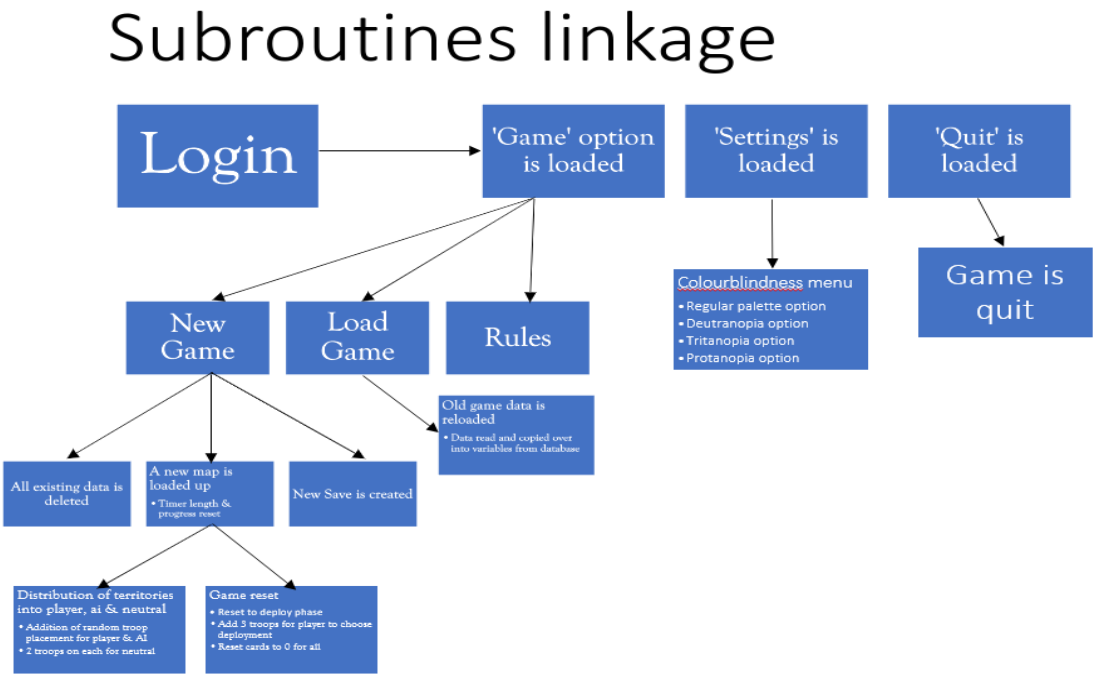
Inside the Game section, the AI function has been added, however this will have its own section specified in the design process, so that the decision-making process/ game rules to be followed can be easily outlined. This will also benefit in easily outlining various success criteria.

Each section of the diagram has a plan, to ensure that there are minimal issues in continuity & clarity. The login/logout section is further outlined in the Database design section, as it directly cross references user data/ game saves.

The actual data types and details of the game itself will be referenced throughout this section, however, the GUI & database section are being kept mostly separate.

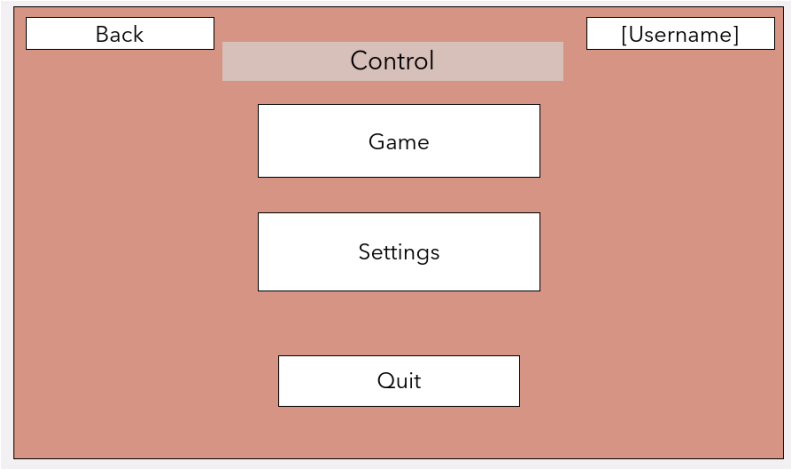
Subroutines & Diagrams

User Interface Design



Menu

Success criteria 10: **'A simple menu should be used to link together features of my project'**

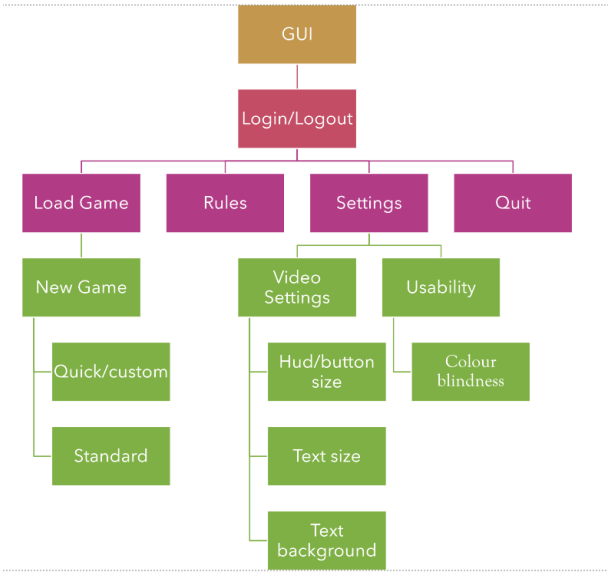


The image above details the first menu that appears to the user after logging in (will be shown in database design), where the three main buttons that appear to the user below the main title is: 'Game', 'Settings' and 'Quit'. The actual design of it is simple, as I am putting more emphasis on the gameplay itself.

The top right will detail the user currently logged on, whereas the 'back' button allows the user to back out onto the login screen.

The purpose of the Graphical User Interface is to provide a menu that is able to link together all functions of my proposed solution, and the planned structure for all options detailed in the main menu is displayed below, alongside an explanation for button placement & choice. The graphical user interface will also allow the user to interact with the main game screen and can be optimised for each user's individual experiences.

The structure diagram shows the list and order of features in which they will exist on the GUI.



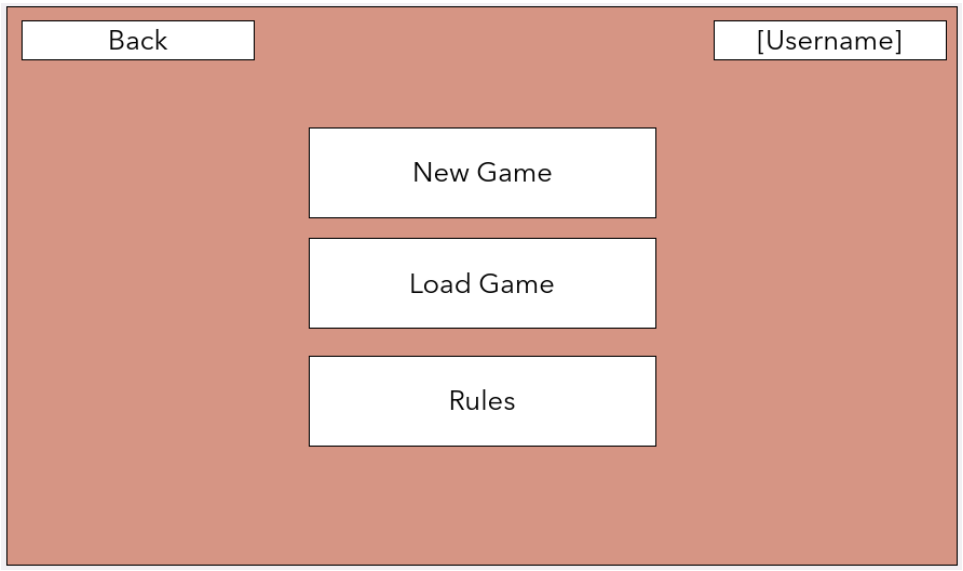
My decision to implement a login screen before being able to access the main menu is to allow the save function to be enabled throughout, as a game that is being played 'as a guest' will not have any value to assign it into, and therefore this would increase the complexity. This is referenced later in my database design section.

The menu relationships can be coded linearly, as each button will create a new page to go onto, and thus will have a one-to-one relationship with one another.

The next menu includes 4 button options:

- Load Game
- Rules
- Settings
- Quit

Load game (the two options) -



This option will allow the user to be able to access the main game, from which they can access one of two options: '*New Game'* and '*Load Game',* the decision behind this was to allow the user to overwrite the current save data by selecting 'new game' and to wipe the contents of the older game to begin anew. The following pseudocode is representative of the functions carried out when overwriting game data once the cursor has interacted with the 'new game' button. The actual overwriting of game data algorithm will be specified in the database design section.

All variables inside the overwriteGameData() are all interlinking with all my other sections, as the variable RunningGame is a function that checks if the physical game ie timer, phases etc is still running. The savedData function is used to check if there is a data entry for the user currently logged on, and if there is data to be found in the game code then it will bring up a pop-up on the GUI through a function called overwritenotice().

```
function checkRunGame():
//this function is used to check if there is currently a running game to trigger a pop-up notice
        if RunningGame != false then:
                overwriteGameData()
        else:
                gamemenu()
                set overwritenotice to false
        endif
endfunction
```

The 'load game' choice will take you directly to the current game played, however, the 'new game' data will re-generate the territories and save the data from that into the database. The 'new game' option would allow you to select the pace of the game based off how long the player wishes to spend in a singular game. The quick length games will shorten the timer down to a % of the original time per phase, however the deploy phase will always last an infinite amount of time, as this will decrease the complexity if a player chooses not to place any troops down, and ensure the player gets to select all their moves, however, the reduction of the other phase timers will ensure that each round is played in a smaller proportion of the duration.

Based off of stakeholder data, when asking **'would it be preferable to be able to edit the length of time of each round?'** , it was found that people preferred the idea of being able to adjust their game accordingly, both for personal preference reasons to add usability functionality, but also to add a flexibility option that cannot often be applied to the physical board game. The common responses were akin to: 'So that [the player] has more time to consider all their options and is not as inclined to feel rushed'. In light of this, the way this can be encoded is as follows, however, I have still decided to place boundaries on the length of time, due to processing and performance reasons:

```
function time_adjust():
        print ('what time would you adjust it to?')
        x = int(input ('.'))
        Timer = x
if x >= 10 then:
        print ('The time cannot be any shorter than 10 seconds')
        time_adjust()
elseif x <= 1000 then:
        print ('Too long')
        time_adjust()
        endif
endfunction
```

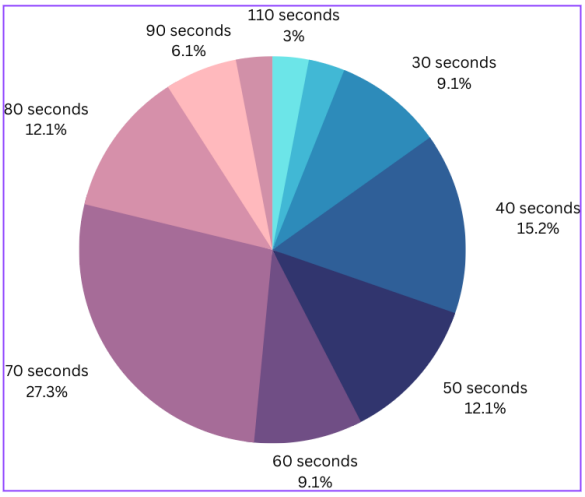
//the time will then adjust to the input of x, when under the condition of 10 it cannot be considered as it is too short, whereas anything above 1000 becomes excessive

```
if Custom_button_pos == click_pointer then:
        time_adjust()
Endif
```

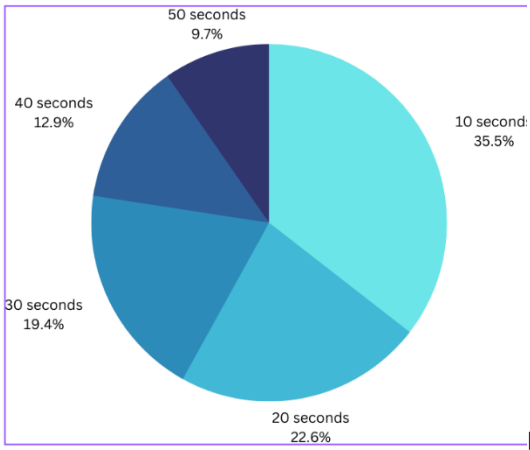
//when the time change button is selected by the cursor, it will trigger the function time adjust, which will alter the pre-existing, flexible code to the value entered by the user in 'x', essentially redefining the timer under the variable x.

This will swap out the values of the timer's increments down to the time specified, however, there is also limits to them amount of time for it to then become excessive or too low.

Based off of stakeholder assessments, it has been found that the average player requires at least 100 seconds to play each individual round for a standard game (unmodified), although the deploy phase took large varieties of time slices, most people were quick enough to finish it within the 10 second bound, however, there will be no timer needed for this phase. The results recorded have been put onto pie charts below that show the time taken for each player to finish their attack round:



With the majority of results being recorded in the 40 – 80 second range, it can be inferred that the sensible option is to make the timer setting for the attack phase 70 seconds for a standard game.



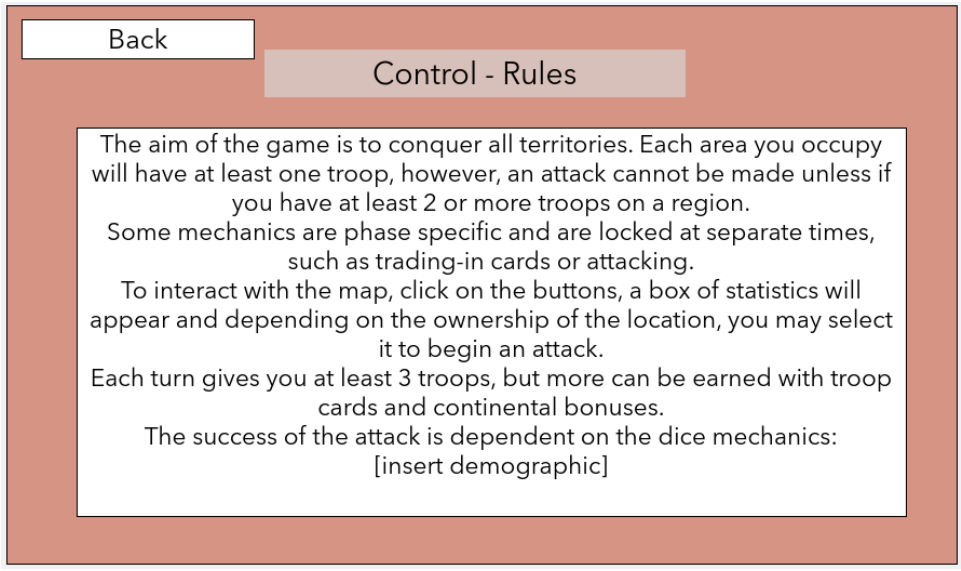
Likewise, a majority of stakeholders chose to fortify their defences in 30 seconds, however, since no fortify phase took longer than 50 seconds, it would be wise to cut this down to 30 to reduce time spent in the round, as only one move can be played in this phase

Rules -

This is one of the first things on the first menu, as it will be a straightforward way of finding the ways to navigate the game for those that are inexperienced with the game's rules.

This will explain the details of the controls, as well as outline the base rules & structure for the game, and its purpose is to explain some of the mechanics, such as the dice function for attack & defence, as well as explaining the card function and how to interact with the game overall.

An example of what this may look like:



Settings -

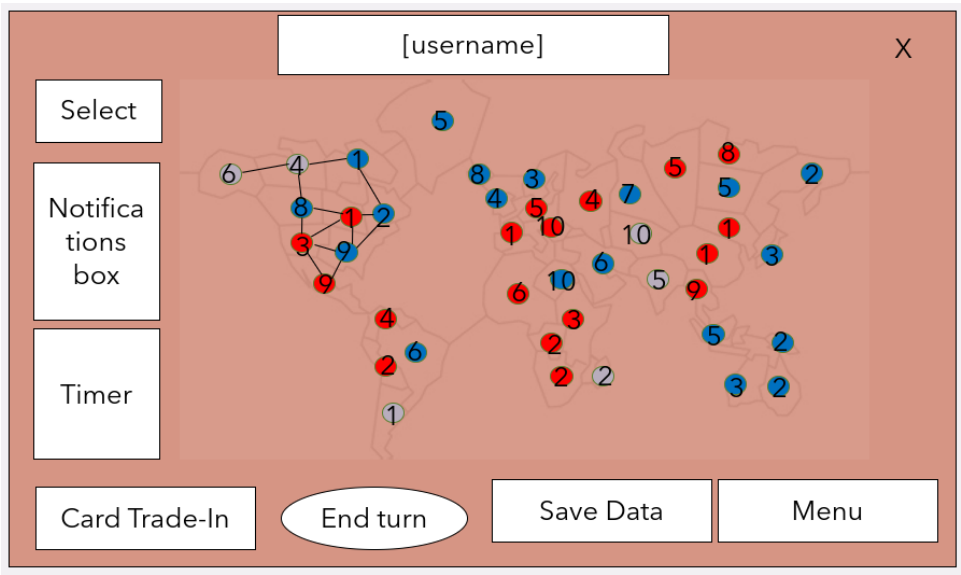
The settings are to be able to access changes to the GUI for the game, the purpose of this is to add a level of flexibility for the user in how they wish the game to be presented. The usability features are outlined below.

Quit -

Success Criteria 11: The game can be quit at any given time, with the use of a 'Quit' button.

A quit button serves the purpose of being able to exit or close the game down at any given time, whilst also halting all assets. The placement of this button will be on or easily accessible to every GUI screen entailed in this project.

<u>Main game screen</u>



Success Criteria 6: It would be suitable to add a notifications box in the top right.

The decision behind this, was so that a user is informed of any changes, such as how many troops they received, the turn or phase currently entered or what territories have recently been conquered.

A few notable features I would like to implement on the main screen and a brief explanation as to why is:

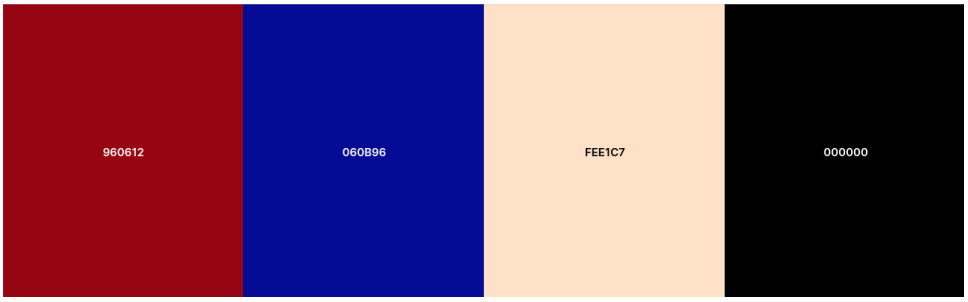
1. Username - Displaying the username will allow the user to know who is currently logged in, this is useful to keep track of which account is currently making the game progress.
2. Select– To be able to verify the choice, so that the player can confirm the choices that have been made, such as increasing the troop count of a location, seeing & selecting the attack pathways, or just checking out the win probability of that location, and if they decide against an attack, the alert will have a 'back' & 'confirm changes' option, so that any changes made may/ may not save unless specified by the user.
3. Timer – so the player can keep track of the time remaining on that round and make decisions accordingly.
4. Card Trade in – so the player has the option to increase their troop count in every deploy phase, will be disabled when in attack and fortify phases.
5. End turn – to be able to advance to the next round, however, this option will not be available during the deploy phase, as that phase won't end until specific requirements are met, for example, all troops placed down.
6. Save data – to be able to save a new value into the database so that the game can be loaded up to that point
7. Menu – so the player can access the main menu and all its features
8. Notifications box – to be informed of the games changes as the phases and turns make progression. (It is also easier to follow for testing this way)

These will help to add a level of interactivity with other mechanics and show essential information for the user in order to be able to play the game. The algorithms for these features are under their respective success criteria in other sections of Design.

Usability features

Usability features that have been considered for my project consist of Variable game lengths, colour blindness and the ability to change text size. The purpose of this is to allow users to access and change key parts of my program, so that it may be better suited to them. An example of this may be a larger text size will allow users with reading or vision difficulties to be able to easier utilise the program.

As mentioned in my stakeholder interviews in Analysis, here are the concluding colour palettes for the three most prominent forms of colour-blindness, along with my current simplistic colour palette, few colours are used as a part of this project, so that it can perform its main function of highlighting allied and enemy players:



Protanopia:

Protanopia is the insensitivity to red light, resulting in the confusion of greens, reds, and yellows.

Deuteranopia:

Deuteranopia is the insensitivity to green light, causing confusion of greens, reds, and yellows.

Tritanopia:

Although Tritanopia is not as common, it is the insensitivity to blue light, resulting in the confusion of greens and blues.

The colour palette is also purposefully not inclusive with green and yellows, in order to avoid confusion between these colours. The red is to be used for the colouration of the AI player's territories, meanwhile blue is for the player's, and the bisque colour is for the neutral forces.

The way to go about this change is if the default colour palette is set for each individual button, with the black being used for details such as the map being switched out with the new values, based off which option is selected. Below is some pseudocode used to explain how some of these changes will be made after a button is pressed in the settings, with the tritanopia colour palette being used as an example:

RED = (150,6,18)

BLUE = (6,11,150)

BISQUE = (154,255,199)

BLACK = (0,0,0)

Allegiance = countryowner

colour = ownercolour

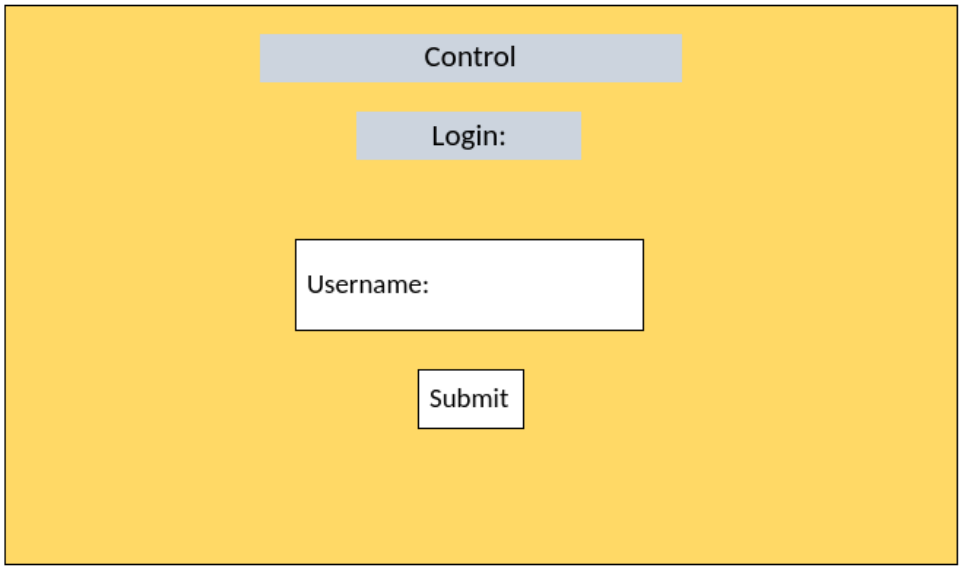
```
ownercolour = BLUE
enemycolour = RED

CLASS ButtonIdentifier
        PRIVATE allegiance
        PRIVATE colour
        PUBLIC procedure new (country)
        country = givenName
        endprocedure
endclass

CLASS Trit_button inherits ButtonIdentifier
        PUBLIC procedure new (colour)
        PRIVATE allegiance
        endprocedure
endclass

if Trit_button == pygame.mouse.get_pressed() then:
        Tritanopia()
endif

function Tritanopia():
        Trit = True
        while Trit != False:
        RED = (142, 12, 12)
        BLUE = (6, 89, 83)
        BISQUE = (252, 210, 211)
        BLACK = (30,19,19)
        endwhile
endfunction
```

If the button is pressed, it is set to trigger this function, meaning that its only method of activation is through this. The tritanopia function is used to rewrite all the predefined colours so that anything using the same variables as RED,

BLUE etc is recoded into a different palette. A while loop ensures that this change remains after the function has been called, until the boolean value has been set false.

The function can be reapplied to the other types of colourblindness, and can also be used to revert back to the default settings style. The classes for the buttons are useful as they are my main representation of colours and user interaction, typically the normal class for this would be classic_button, but its resetting to a different button altogether through inheritance allows it to behave differently to the default option.

Database Design



Save game data & the database

Success Criteria 9: Saving the game should be available from logged in users.

The database design involves the login & logout of users, involving input validation and save game data for the player. The python library 'MySQL' will be used to create a link between the python programme and the database.

The game save system will act similar to a stack in memory, in the respect that it will transfer all the contents of variables into a list, however, this will create a code, with each continent owned under the player1owned list. Acronyms will be used as codes for locations, so that the data representation ends up like a ratio of troops: location, for example, if there are 12 troops on Iceland, it would be represented as '12:IC' could then be combined into a list, where the values will be retrieved. These values would be entered under the entry of the username.

The user verification system will involve the checking of the username against this database. I have purposefully chosen to exclude passwords & password input, as this would introduce concepts that are unnecessary to the development of this program.

Where ':' is used to reversibly concatenate the data string between reading from and writing to.

An example of the entries in this database is:

Username	PlayerSaveCode	EnemySaveCode	NeutralSaveCode	Phase & Clock Timer
exemplar	'Troops:location','troops:location'	'Troops:location','troops:location	'Troops:location','troops:location	Phasecode:second
user01	1:UE,3:NA,12:IC	2:GU,6:AU	2:ARG,4:GB,11:BR	FOR:27

For the translation of each code, a function will be required to convert the acronyms into the actual classes, this could be done in a list. The representation of the mysql.connector library is very limited in pseudocode, but it will act

in a similar way to reading from and writing to file, but instead with search queries such as WHERE [entry] = {x}, with variables such as 'mycursor' or INSERT INTO, .executemany etc. An example of the concept of this (pseudocode):

```
//reading from
database = openRead("db.txt")
while NOT myFile.endofFile():
//finishes at a specified line
        print(myFile.readline())
endwhile
database.close()


//writing to
db = openWrite("db.txt")
db.writeLine("User01")
db.close()
```

An example of the sorts of SQL required to be able to take the fields and locate them in a query will consist of:

```
SELECT playersavecode
FROM logindb
WHERE username = [currentusername]
```

Validation

In ensuring that the program is as robust as possible, all user inputs are required to have at least some form of validation to ensure the correct data types are inputted. The following are all ways of validating user input, as well as this, an example of coded validation will be used in controlling the login system inputs.

Buttons

These ensure that the user cannot interact with anything other than buttons for specific options: it is either selected or not selected. Buttons are used to interact with a GUI, making them useful for menus as the data type cannot be changed, if they are not connected to anything they will simply not lead the user elsewhere, ensuring that they can be an easily controllable link between features of my program.

Text boxes

This is mainly used for the initial login, and is useful for validation of characters, as I am able to control the length and type of the characters used. An example of validation for a username could be the restriction to 10 characters, and the limitations of using no numbers or special characters, but also may require a minimum character count of 14.

Success Criteria 13: Validation should be used for the login system

Data input	Validation type	?
Name	Normal	
[NULL]	Invalid	Needs a value
abcdefghijklmno	Boundary	14 characters only
A%$3s	Erroneous	A-Z only

The table above outlines the accepted data types for the username. To ensure a maximum character count, .len() will be used to ensure that the user input meets this standard, whereas a .txt file of all accepted values (integers and alphabetic values) will be used to ensure that special characters cannot be used.

AI design

When assessing the actual design for the AI player, the research instilled some base objectives for the ai to be able to conclude, however, the limitations of this research was that it was a fully coded ai beforehand, and thus already had a grasp of the base game rules.

Stakeholder responses relating to my proposed plan to implement an ai varied. When asked ***'Considering the AI's purposes as a substitute for an actual player, what potential drawbacks should be considered and how can we justify preventative measures?'*** these responses were the most useful:

Response A: The AI should be constricted to the limits of a timer, and each decision should take up a fair amount of time so then you can argue it behaves somewhat more like a human in terms of 'cognitive' thought. It should also consider doing this one at a time, so that it cannot attack multiple locations concurrently.

Response B: You should make preparations for the possibility of the algorithm to attempt to make multiple calculations at once, if it comes to more than one decision, it must do one OR the other: not both unless if there is sufficient time to do so.

Response C: Don't make it impossible to beat – the risk of too many calculations will make it increasingly difficult for the average player and can take the positivity out of the program.

Response D: An element of randomness would help ensure the player can salvage any unwise moves; if this program is for the average player, then it should be considered that not everyone makes the best possible moves during their turn.

Response E: Attempt to shorten the timer, as the calculations processed by the AI will be in quick succession, and even with time restrictions, it is still capable of running thousands of calculations inside a second, which is an advantage that the player does not have.

Response F: It must consider the logical order/ rules of the game, and thus must deploy what it can before moving on to attacking and reinforcing its areas. By structuring its decisions, it ensures that it can act logically, and does not have any unfair advantages over the player.

Ai player

Success Criteria 3: The AI should follow all pre-set rules of the game.

Prerequisites for the AI to follow (base rules):

- Cannot attack a territory that is its own
- Cannot fortify or deploy on enemy territory
- Cannot make decisions in quick succession – there must be a cooldown before moves
- Can only add available troops
- Must use the correct connected attack pathways
- Cannot fortify an isolated territory

This could be checked with recursive functions, applied to all territories that meet the requirements of being owned by the ai. The implementation of this can be through a function that can be generally applied to every territory that meets the initial requirements of this. Nested functions will be necessary to ensure that the AI follows the timers set out for it.

Breakdown of phases & rule criteria it must meet

Deploy

- Available troops -> ensures that it cannot go below this counter, so will only place down troops it actually has available until aiavailabletroops = 0
- Location ownership -> can only be transferred from aiavailabletroop counter and added to ai territory troop attribute(s).

```
array AIowned [a]

AIowned [0] = Egypt

AIowned [1] = Congo

AIowned [2] = N_Europe

AIowned [3] = Peru

//array defining what is owned by ai


ainewtroop = 3+continentalbonus

aideployrandomiser = print(random.choice(AIowned))

a = AIowned.count

//a is how many vals counted in array AIowned


function aideploy():

        while ainewtroop!=0:

        aideployrandomiser.troopval +=1

        ainewtroop -=1

        endwhile

        aiattack()

endfunction

//concatenation of integers used to calculate ainewtroop value
```

Attack

- Attacked locations must be either neutral or the player's
- The attack is only worthwhile if the win probability is at least 60% or greater
- When attacking, the dice values must be repeatedly compared, with the side with the highest value removing 1 from the attackingtroopvalue of at least one side
- The dice vs dice will be repeated until one side reaches a counter of 0
- If the results of the dice reaches an equal value, no troops will be removed, and instead a reroll will be issued
- If an attack is won, the colouration of a button will be characterised by a change in class
- The timer must go down by 1 every second whilst this phase is running until it reaches 0 and activates the next phase.
- Attacks must come from owned territories

AIcountryenemyrandomiser = print(random.choice(AIenemyneighbours))//reusing a similar list to the example for fortify, so it scans through all neighbouring attributes for the territories that meet the requirements as not belonging to ai player

//Example algorithm:

dice = [1,2,3,4,5,6]

function AIAttack():

AIclock() //this is similar to the regular clock mechanic

While AIclock >0:

if neighbouring_loc == p1_alignment then: //if enemy is next to picked

if AItroop>enemytroop then://if ai has more troops than enemy

if win_prob <=60% then:

while AItroop!=0 OR enemytroop!=0:

function AIattackdice():

AIdice = dice.random

enemydice = dice.random

if AIdice == enemydice then:

time.sleep(1)

AIattackdice()

elseif:

AIdice>enemydice

enemytroop -=1

time.sleep(1)

else:

AItroop -=1

time.sleep(1)

```
                    endif
                    time.sleep(1)
                endfunction
                AIattackdice()
            endwhile
            if Aitroop > p1troop then:
                set_button_AI()
                //function that sets the territory to the ai's colours
            endif
        Elseif:
            time.sleep(1)
            AIattack()
        endif
    endif
    time.sleep(1)
AIattack()
endif
endwhile
endfunction
```

The regular time.sleep's are included to place a limitation on the amount of moves the AI player can use. Meanwhile the aiclock function will count down from 70. The timings will be changed throughout development to check if it is too long/ should be amended.

A nested function is used to ensure that the AIdice is contained within the aiattack function, so it cannot be activated at any other time other than whilst this is active.

Fortify

Factors affecting fortify:

- Must be interconnected to allied territory
- Is the transfer of troops
- Cannot leave less than 1 troop on a location
- Must abide to the rules of the timer

The fortify phase will be difficult to restrict to a singular pseudocode algorithm, so therefore more about this phase will be detailed later in the game mechanics section.

Success Criteria 7: The game should be winnable against the AI.

If the AI was to take every logical move possible ,it would become unbeatable, therefore, an element of randomness would be needed for logical operations, for example, if it found a choice between attacking location A, B & C, then it

would have to select from an array, a random number, this could then be used to make a decision – so if the AI chose an illogical move, the player is able to recover from it and beat that force.

Each recursive function can be limited by a duration with the addition of time.sleep(x). So that the ai cannot make more than one moves simultaneously against the player.

Neutral force

Success Criteria 8: The neutral force should be powerful, but not unbeatable.

To split the board further, a neutral force is required to be able to put barriers on a player quickly dominating continents, and tapping into the troop bonus at the start of the game. The following rules are outlined:

-Upon map generation, each player gains 40 armies, with 28 armies on the neutral force's side (always 2 on each of those territories so 14 in total)

- For two player games there are no mission cards available to play, so the objective of the game is to beat the enemy player – however the neutral force is not included in this.

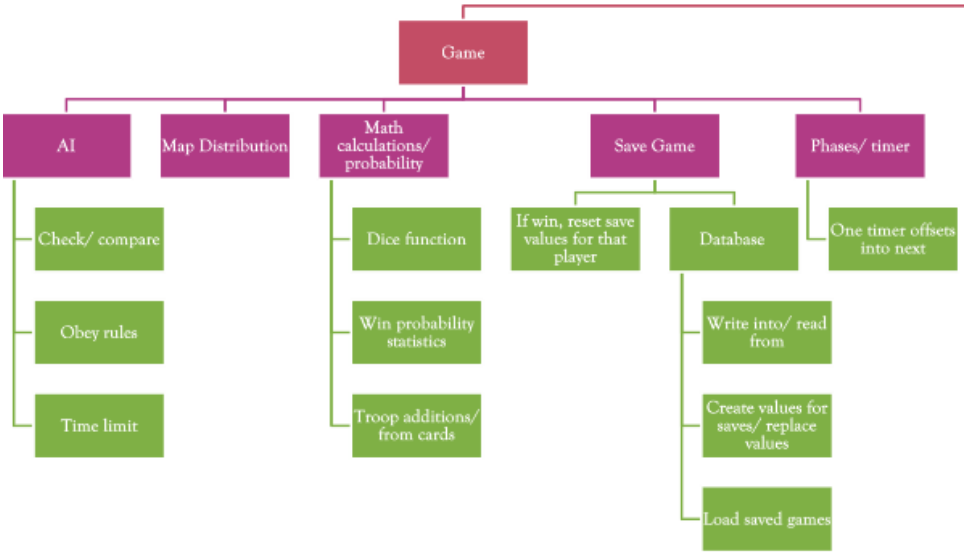
- When each player gains their troop bonus, the neutral force gains half of those reinforcements

neutraltroops = aiavailabletroops+p1availabletroops

neutraltroops = neutraltroops/2

- They cannot attack, but can defend their territories, so therefore don't take any 'turns' - they are limited to defending and deploying.

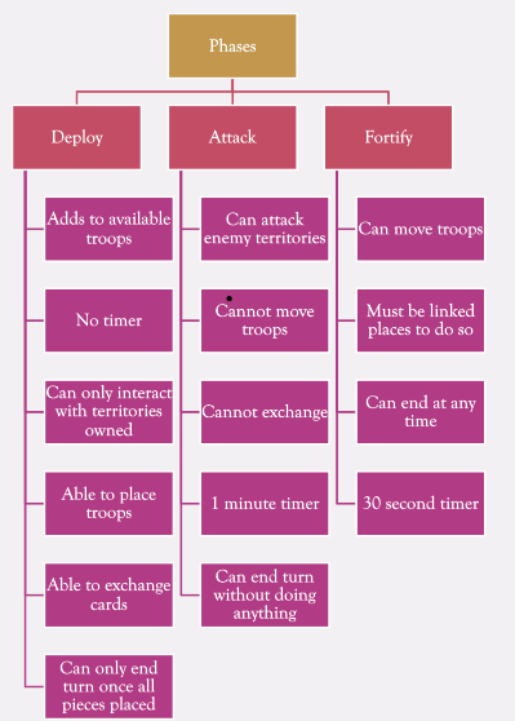
Game Mechanics



Phase mechanics

Success criteria 2: I would like the game to follow a turn-based structure

As a result of this, a phase structure to the game would be sensible, as it would allow the transition between different rules and functions without it becoming too overwhelming for the user. The phases have been separated into 3 values:

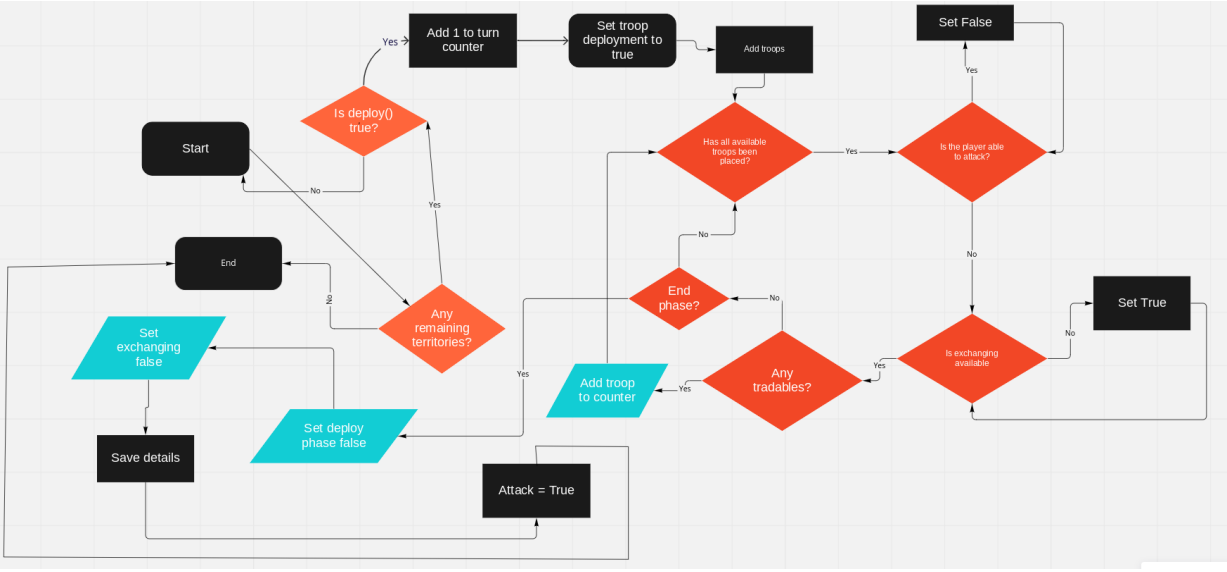


Deploy

This is the first phase, and therefore must follow a set of parameters before the player can begin their moves. A system that is able to check if certain features have been disabled or enabled will be the first step in this 3 part-system. At the start of the turn, the player is granted with at least 3 new troops, however, this troop bonus must be flexible enough to be able to change if a bonus is appended. The first step in this model is to first check if the game has been won already, so that it doesn't change turns to a player that as no remaining territories, and therefore cannot play their move.

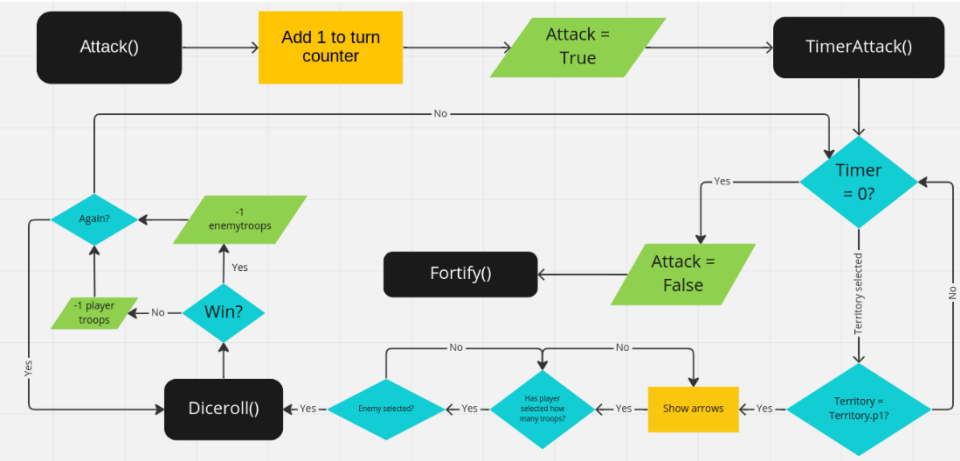
Unique features of this phase must also be allowed to execute, for example, if the player has any cards that they wish to trade in. When doing so, I must also ensure that the button is visible, and brings up a new menu and set of functions. The function deploy() will act as both an initiator of events, as well as hold a variable with the Boolean value of True/False.

In planning the code for these phases, recursive functions will be used to ensure all requirements have been met for that phase.



This model ensures that the phase has been enabled before being enacted, a while true() loop may be useful in checking the progression through the phases and ensuring that checks are following one function at a time to prevent the user from re-loading the program.

Attack



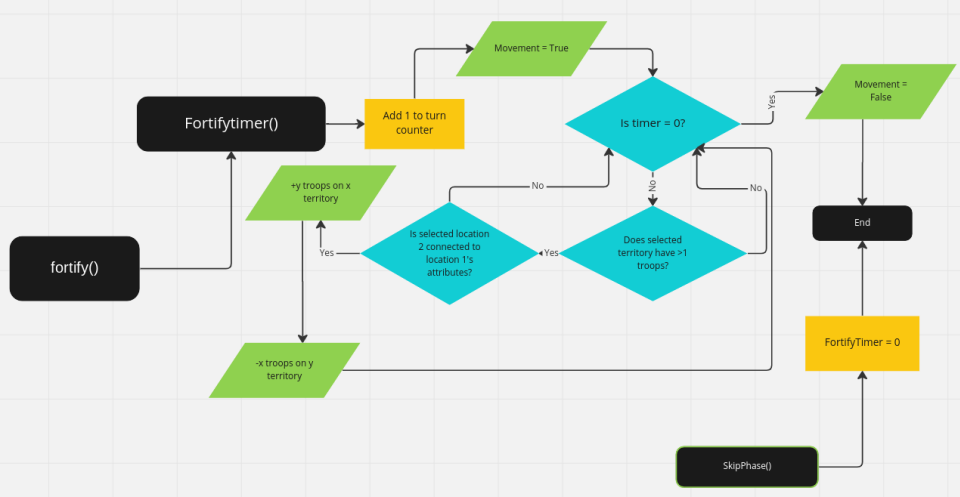
The attack phase is not as versatile as the deploy phase, as the only thing you can do is attack another player, therefore, as a result, the boolean value ‘attack’ is set to True. However, a difference with this is the sets of functions that are called throughout; the dice roll function will be implemented as a nested function, which can be called upon as a result of the attack phase being activated.

Another difference to the deploy phase is the usage of a timer, if multiple outputs result back to the condition of the timer reaching 0, then I can begin the next phase of the program. The attacks will involve the dice roll mechanic, which will feature the typical rules of dice rolling in many board games. They are outlined here:

- One attacker vs 1 defender: 1 dice each
- One attacker vs 2 defenders: 1v2
- Two attackers vs 2 defenders: 2v2
- Three attackers vs 2 defenders: 3v2
- Three attackers vs 3 defenders: 3v2
- The attacker’s dice caps at 3
- Defenders dice caps at 2

This system is used to give the attacker an advantage, as the best of two dice is pitted against one singular dice roll. Therefore the more troops you attack with, the higher the probability of winning or rolling a six is higher. This will then be iterated until the user decides to click away, or until the defender is defeated, or the attacker has no more troops to push forward with.

Fortify



Success Criteria 5: I would like to add a button that can skip phases and end turns prematurely.

Card mechanics

Success Criteria 12: A card system should be implemented into the game

As a part of the deploy phase, I would like to add cards to my game, these aim to add diversity to the deployment phase, and provide a strategic advantage over the neutral territory. I think the way that this could be done is through creating a class called Queue, which has a specific capacity, and acts like a grid, where each card in the position is x+10 (as an example), then these images, when clicked can be added to a new slot which can then be for trading them in. This would also require validation on the card types, thus why a card class may also need to be created, as a superclass variation of the queue slots.

CLASS Cards

PRIVATE Title

PRIVATE image

PRIVATE type

PUBLIC procedure new (draw)

Blit(image, (co-ordinates = x,y))

endprocedure

PUBLIC procedure new (input)

If mouse_position == cards

Return collide = True

Endprocedure

endclass

CLASS Queue

PRIVATE no_of_slots

PRIVATE position

```
slots = []
        PUBLIC procedure new (add)
        If slots < no_of_slots:
        slots.append(x)
        endprocedure
        PUBLIC procedure new (remove)
        If x in slots:
        slots.remove(x)
        endprocedure
endclass
```

Setup mechanics

Success Criteria 4: Territories must be divided equally, randomly and fairly.

As part of the setting up of the game, a third of the values must be randomly picked from an array and assigned to a new list. The counter below is assigned 2 as an example. If the elements are appended onto a list, then this will allow the program to extend this list as more countries are gained. If a territory is lost, list.remove() will be used to ensure that any losses are amended.

```
counter = 2
ARRAY countries =
countries [0] ='argentina'
countries [1] ='brazil'
countries [2] ='chile'
countries [3] ='venezuela'
player1randomiser = print(random.choice(countries))
//shows a random choice for each country in list
player1territories = [player1randomiser]
//p1territories is a list of all random territories from p1randomiser

while counter != 0:
for i in countries:
if counter !=0 then:
        if i NOT in player1territories then:
        player1territories.append(i)
```

```
        counter -=1
        print(player1territories)
        print(counter)
        endif
endif
Endwhile
```

Clock mechanic

Success criteria 1: An on-screen clock should be displayed on screen

The justification for this is for players to be able to track the time left on each phase. The idea is that it is supposed to be near the bottom left of the screen, and count upwards in 1 second increments. An example of this is detailed below

```
end = false
function attacktimer():
        while NOT end:
clock.tick(1)
seconds += 1
win.blit(text, textRect)
if seconds = 70 then:
        seconds = 0
        fortifytimer()
endif
text = font.render("{}".format(seconds), true, (0,0,0), (255,255,255)
//the (x,x,x) format is for the font and background colouring of the timer, so that the timer and backgrounds are clearly visible.
for event in pygame.event.get():
if event.type == pygame.QUIT then:
        quit()
endif
        endwhile
endfunction
pygame.display.update()
```

The purpose of this is to create a while loop that checks if the timer has reached the 70 second mark. The text will be displayed as it gets updated with a following 'display.update', the timer will add 1 second to itself, and this will be shown in a rectangle and represented in the format where the {} is.

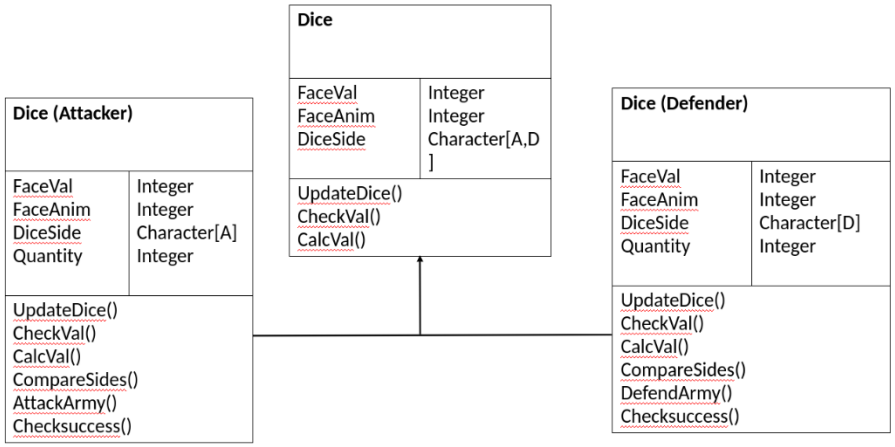
Once the end of the timer has been reached, the idea is for it to offset into the next phase timer.

Key Variables, Data structures & classes

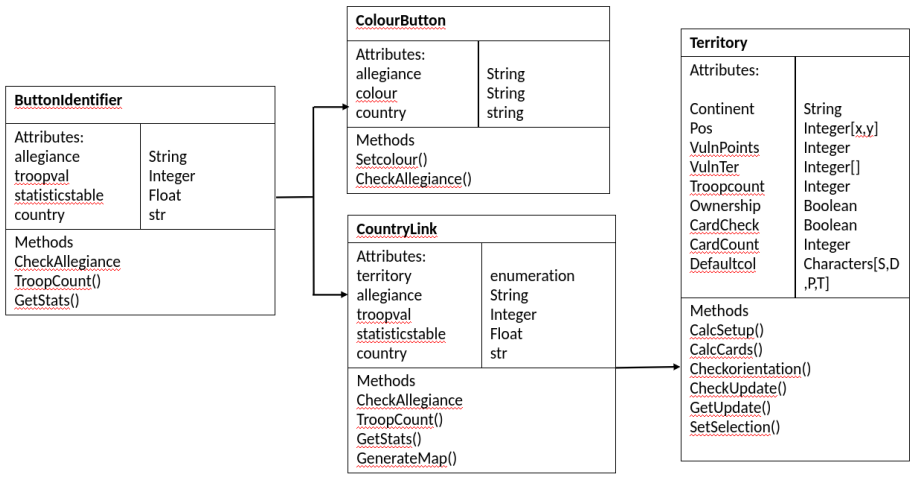
Name	Data Types	How it is used
Available_troops	Integer	A counter for how many troops a player can place down at the start of their deploy turn
Turn_counter	Integer	Counts how many turns have taken place, used in a calculation used to measure which turn it is when reloaded from the database.
P1_timer	Integer	The timer used to measure how long a player has in its respective phase
AI_timer	Integer	A separate timer used specifically for the AI's turn.
Acronym	Array	Is a list of acronyms for each territory, so that it is easier to store data inside the database
Country	Tuple	A tuple containing every location in the program for reference, will also help with the generation of territory distribution. This holds all territories featured in my program, from which other functions can take items from at random, especially useful for map distribution.
P1_territorylist	List	A list which has items that can be appended and removed throughout the game, used as a device to contain all player controlled territories
Dice_val	list	All values of the dice, 1-6
Attack	Boolean	Used to verify if the player has the ability to attack, when set false the feature is disabled
TradeIn	Boolean	Allows the user to trade in cards, boolean is used to disable this feature when set to false.
ButtonIdentifier	Class	A class that inherits & links together features such as alliance, colourblindness and physical representation. Also

		makes use of multiple inheritance.
StartPhases()	function	To activate the main phases, all phase functions are nested within this
Running()	function	The main loop for all values to run within the game

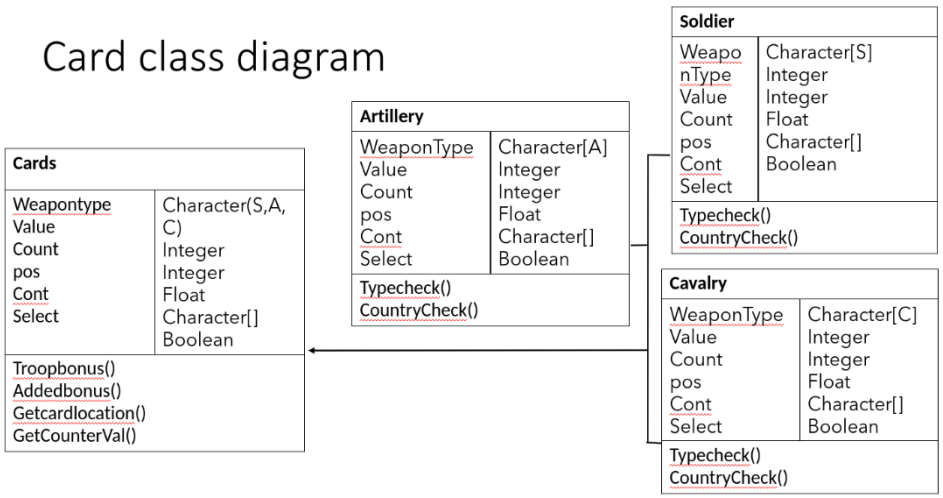
Dice Class Model (Player)



It was concluded that the dice will share the same mechanics to work from its identical parent class, and thus little attributes will be changed, however, it is important that the quantity of dice available is dependant on the role of the attacker or defender, as a defender's number of dice caps at two, whereas the attacker's caps at three. Equally, this will significantly affect probability statistics, and the type of dice is justifiable for making it a separate class, as lists such as face values will be shared.



I have decided to separate this diagram into 1 parent class and 2 sub-classes. The class buttonidentifier will be used to generalise the buttons on every location; the two subclasses coloubutton & countrylink makes use of multiple inheritance, as the colourbutton exists for the purpose of colourblind settings & visual representation, whereas the countrylink class exists to allow the attack phase to happen. From 'countrylink' exists the subclass 'territory', this will involve all the calculations, and can be used to interact with the AI alongside this, whereas the other attributes serve mostly the user.



The decision to make the card class have 3 inheritance groups is due to the individual values of the cards, as their attributes would be different to one another, and they will all share the same data types as the parent class.

Test Data & methods

As I develop my project, I will test each function as I create it to ensure that it works correctly for further development, hopefully ensuring that the program functions when completed, further testing will highlight important features that need improvements, as well as any bugs.

Test data will involve user inputs, as well as using a range of different data sets for example with changing arrays and variables.

After the development has been completed, my final test will consist of an attempt to misuse my program, for example, database entries & attempting to overload user inputs. Incorrect data types will also be attempted, and this will ensure that my validation methods are sufficient.

These tests will be logged for reference, and its results.

Once completed, this will be shown to the stakeholders to ensure that it meets their requirements, and if there is anything they would like to be added or amended.

Test Data for iterative development

Outlined below is a table of tests that are to be run during the iterative development process, I will make use of the agile methodology. By breaking my project tests into separate parts for usability, mechanics and login, it would make it easier to follow, for example, coding the colourblind parts would be unnecessary if the colours of the GUI have not yet been implemented, alongside this, the ability to be able to change colours dependant on ownership would have to be a prerequisite to the colourblind testing; otherwise it may not work as planned.

Test number	What am I testing/ looking for?	How will I do this?	Justification
1	If the mouse co-ordinates are able to be returned to the terminal when an empty GUI is created	Use a while loop to return the mouse pointer co-ordinates through pygame	To be able to figure out where to place the interactive features i.e. buttons, territories so that the features can be displayed without merging.

2	The GUI can be closed with a 'quit' option	While [button_name] and pygame.mouse.get_pressed(), pygame.QUIT()	So the GUI can be closed by the user.
3	The GUI display buttons are shown upon execution, and they interlink with one another	When [button] pressed, initiate function [x], kill button, load next display	This is needed so the menu can advance/ revert to other screens or settings
4	The map is able to load, with all the buttons in the correct locations	Use the mouse pointer feedback to gauge the placement locations for the buttons, once defined under a class, they will be assigned to these co-ordinates.	So that the user can have the base game elements to interact with upon the start of every game – these conditions do not change.
5	The user is able to interact with the buttons on the territories	Add an attribute to the class 'ButtonIdentifier' called 'Enabled', when set to true, a player can interact with it; when false it cannot be interacted with. To know if this has been interacted with, a print statement will be shown in the terminal at this stage	To allow user interaction that is also able to change based off conditions for example the phase.
6	The buttons are coloured accordingly upon distribution	An attribute colour will fill the empty buttons with the assigned colours, these can be rewritten at various points.	So the user can see a visual display of which territory is aligned with which player
7	The timer, upon reaching 0, goes into the next phase timer	The timer will count down with 1 second increments [timer = timer –1, time.sleep(1)] When timer not 0, will continue to go down, triggering next function	So that the phases can be timed, and creating a reliance upon the timer for certain actions to be made.
8	When a new phase is executed, the new rules are met and the previous one(s) overruled	When a phase is execute eg 'attack()', the previous variables such as enemybuttons are set to true (to enable interaction with these) and functions such as cardmechanic is set to false.	So that the base game rules can be followed, allowing the player to only interact with specific elements when appropriate.
9	Dice mechanics test	AttackDiceno = available_a_troop-1, if diceno == 0, close pop-up, elif diceno <= 4, set diceno to 3. Similar process with defence but defencediceno = defencetroops Then compare highest values of attackresult and	So that the dice mechanics can function in accordance with the game rules

		defenseresult, which results in the removal of troops on specified sides	
10	Statistics test (pop-ups)	Once the dice mechanic is finished, the hypothetical comparison between the two selected locations (one ally, one enemy) is printed on screen into a %, based off the probability of one side scoring higher than another	To ensure that the statistics function can be shown to the user
11	The game can be won by a player once either player 1 or aiplayer have no remaining territories.	When aiownedterritories == [NULL] or p1ownedterritories == [NULL]. Load next screen (P1 wins!) or (AI wins!), disable all game variables and after a button click, it reverts back to game menu (GameRunning? == False)	All assets cease to execute when the game has been won, essentially resetting the board and informing the user of the winner.

AI Testing:

Test number	**What am I testing/ looking for?**	**How will I do this?**	**Justification**
1	Does the AI have its own turn?	When p1turn = false AND pausemenu = False set aiturn = true.	So the turn based structure does not infinitely replay the user's turn.
2	Can the ai change the variables of its territories with a positive addition whilst still in the constraints of the available troops	For every troop placed down by the AI, remove 1 from the available counter until it reaches 0, thus ending and disabling that specific turn.	To ensure that it can interact with its own territories and follow the base game rules for the first phase
3	Can it make a decision based off of comparisons?	A procedure is used for comparison maths, then they are evaluated (essentially brute forced) for the best solution to attack, this will require many comparisons. Once evaluated, the best ones can be added to a list called decisions, from which a random one is executed.	So that the ai can be more dynamic in its choices, presenting a possible challenge to the user.
4	Is the ai limited by time restrictions?	A delay will be added involving the time module, so that every	To ensure that the program is fair for all

		time a process is executed, it will simulate a human response time, thus preventing decisions being made in quick succession at an inhuman rate.	players, and the player is able to win.
5	Can the ai only interact with its own territories once the attack timer hits 0 and resets into the fortify phase?	When timer == 0; p1 interact == False, and the only locations it can transfer troops must meet the criteria of at least 2 or more troops, leaving at least 1 behind at all times. The connected territories link must be used to decide which territories can have troops transferred between them.	This enforces the ai's ability to follow the game rules, respective of each phase initiated.

Login testing:

	What am I testing/ looking for?	**How will I do this?**	**Justification**
1	Does the login text box and submit button show on the screen when first run?	These buttons will appear on the gui by a function called on startup.	The login screen needs to be the first screen displayed for the user, so that they are able to access saved data.
2	SQL injection	Pass the string "__'__ into every textbox	To ensure that the system can handle this without error
3	Deleting all users	Ensure that any input containing the values 'DROP TABLE' or 'DROP' is not able to be used for	To ensure that all records are not deleted upon attempted misuse
4	Input validation	The inputted characters must be a string, with at least 2 numbers and a minimum length of 10 characters	To ensure that each user is able to have a unique identifier – making it more unlikely for repeat fields in a database
5	SQL connection dropout	A terminal test will be run, showing the feedback of the connection. If the	To ensure the user is connected to the sql database for save game data.

		connection is dropped and not equal to True, then it will run a set of parameters again to reconnect.	
6	The data from variables are copied to a field in the db	{SQL query: WHERE username = [currentusername], FROM [dbname], INSERT INTO [attribute], Value}	So that the data can be written into the database for later retrieval
7	The data from the attributes in the table under the current username are loaded into the variables	{SQL query: SELECT * FROM [dbname], WHERE username = USERNAME}, could be done by a series of retrievals so that each is loaded into the individual variables	So that the user can retrieve saved data

Testing for post development

Test number	**What am I testing/ looking for?**	**How will I do this?**	**Justification**
1	If the values of save game is returned to the database fields	OpenWrite (variable) INSERT INTO (field)	To ensure that the saved data is transferred to the database
2	All colourblind palettes appear interchangeably	Reset the colours under a set of variables from within a function	So that the users can change the colour options at any stage
3	If specific mechanics are linked only to their own respective phases	Check under a set of parameters before activating and deactivating rules	To ensure that the game follows its pre-determined set of rules
4	If the territories can interact with their neighbours during the attack phase	Check a set of variables to see if the button has been pressed;	To ensure that the rules of the game are followed accordingly so that the attack phase can be performed

Post-Development Test Checklist

The following test checklist will be used to check all functionality of my program has been met, this will be used to ensure all base functions are in working order, which will also help with project evaluation; details of the testing action align with my success criteria:

Testing action	***Does it work?***
Program menu's revert back to previous ones upon selection	
The program can be closed at any given time upon selection	

Colourblind settings load upon menu interaction	
Colourblind settings have the ability to revert back to default	
The timer will run whilst other processes are running	
All territories interlink and influence one another	
Player distribution is in effect upon game loading	
The player is able to interact with cards	
Cards can be collected throughout the game	
The save game feature is able to be written	
The save game feature can be read from/ loaded	
The player is able to login	
The correct dice is used in comparisons when attacking with variable numbers of troops	
The win statistics/ probability are clearly shown to the player	
The timers/ phases load into one another	
All rules of the phases are followed	
The game shows the winner	
The AI interacts with territories	
The neutral force is included in setup/ deployment	

When this list is completed, I will be sure that program functionality is met, so that it is able to be passed onto stakeholders for their final input.

Iterative Development

Evidence of Development & solution

Before development of my coded solution, the main graphics are created using a transparent image with a Creative Commons License. I am doing this so that I have a game visual to be able to use around development, as a game without a map will mean that I am not able to code in the button locations.

Test 1: Setting up my Code – Cursor feedback and GUI

`` #imported modules import random import pygame import time import os import sys pygame.init() #variable declaration (GUI display) SCREEN_WIDTH = 1000 SCREEN_HEIGHT = int(SCREEN_WIDTH * 0.8) ```	The imported modules used for the setup are: *Random* – a prerequisite for the distribution system *Pygame – the module for the GUI display and the game controls/ mechanics* *Time – for incremental countdowns & ticks for the ai* *Os – to be able to quit the GUI display upon escape being pressed* *Sys – to kill the window upon exit*
`` #colour palette declaration - default RED = (150,6,18) BLUE = (6,11,150) BISQUE = (154,255,199) BLACK = (0,0,0) #setting a timeframe clock = pygame.time.Clock() FPS = 60 #sets the gui size & caption screen = pygame.display.set_mode(((SCREEN_WIDTH), (SCREEN_HEIGHT))) pygame.display.set_caption('Control') ```	The colour palette for the background and buttons has been added here, in accordance to the standard palette detailed in *design.* The clock is also defined, but is not yet able to be run as its function is empty. The screen dimensions and caption is set to the title of my project, *Control,* with the screen height resolution being set 4/5 the size of the screen width

```
def draw_background():
    screen.fill(BLUE)
    pygame.draw.line(screen,BLUE,(60,80),(130,100))
    pygame.display.flip()

def Menu():
    playbutton = pygame.draw.rect()


class buttonidentifier():
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)

#controls...
def CheckEvents():
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
            if event.type == pygame.quit:
                run = False
                sys.exit()

#main loop for function declarations
run = True
while run:
    clock.tick(FPS)
    draw_background()
    CheckEvents()
```

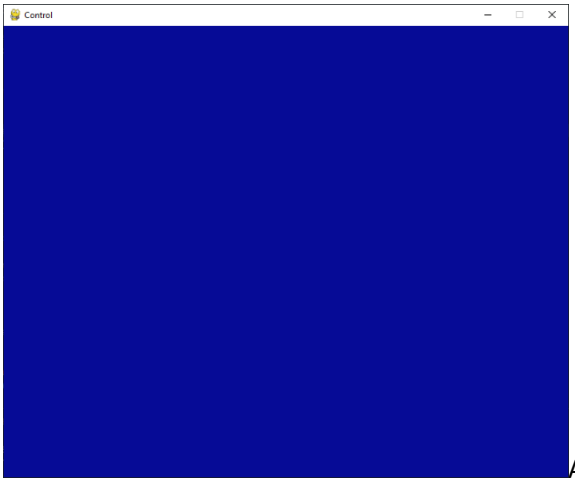
The function draw background is used to set the screen colour, so that the GUI appears upon starting the program.

The menu currently does not do anything, but the function has been created as a starting point for future variables, alongside this, the buttonidentifier class has been created for the menu buttons and later usage for the buttons that define the territories. This is expected to have multiple superclasses.

Checkevents is a function loop that checks if a button has been pressed to perform a certain action, so in this, when escape has been pressed, the GUI closes.

The run loop ensures all the functions are run accordingly, and sets the conditions for the timer.

Result:



As you can see, this prototype is sufficient to begin the foundations of my project.

Main menu design

In starting up the main menu design, I have imported the modules random, pygame, time, os and sys as my main modules – time will be very important during the game timer and decision making process of the ai, meanwhile sys is vital for the gui to be able to be exited, and pygame is the main module to run the GUI on. Random will be used for distribution, especially important with lists.

Test 2 - The GUI can be closed with a 'quit' option & Test 3 - The GUI display buttons are shown upon execution, and they interlink with one another

Main file (update records)

```
class MenuButton():#creates the menu buttons
    def __init__(self, image, x_pos, y_pos, text_input):
        self.image = image
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))#co-ordinates to draw the location of the detection rect
        #change self.rect and self.text_rect co-ordinates to maintain positional consistency
        self.text_input = text_input #allows me to input the text
        self.text = font.render(self.text_input, True, "Black")
        self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))

    def update(self):
        screen.blit(self.image, self.rect)
        screen.blit(self.text, self.text_rect)

    def checkForInput(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
            print("button response confirmed")#shows positive test response once clicked

    def HighlightByMouseHover(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
            self.text = font.render(self.text_input, True, "red") #allows the user to know whether the cursor is by the text
        else:
            self.text = font.render(self.text_input, True, "black")#resets back to black

menu_button = pygame.image.load("button.png") #associates the button image with the class button
menu_button = pygame.transform.scale(menu_button, (400, 150))
```

The screenshot above shows the implementation of a new class, called ”MenuButton”, this serves as a blueprint for the rest of the menu buttons, with each button sharing the properties of positioning, text input and an image of a rectangle. For each menu button, the checkForInput will be used to detect if the button has been clicked, and thus when it should respond to the user, with HighlightByMouseHover changing the colour momentarliy so that the user is able to recognise that their mouse is within the range of the button to click. ’button.png’ is an image of a plain rectangle, serving the purpose of a simple box to be filled with text.

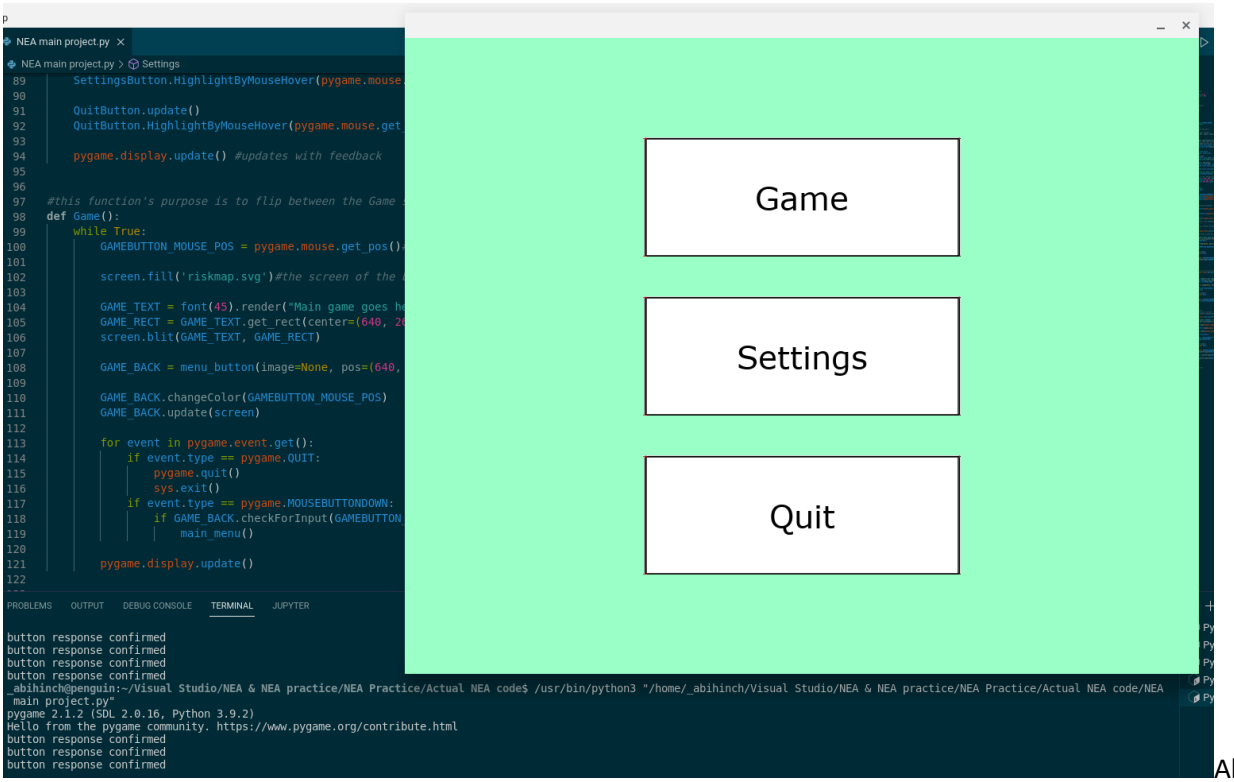
```
menu_button = pygame.image.load("button.png") #associates the button image with the class button
menu_button = pygame.transform.scale(menu_button, (400, 150))

GameButton = MenuButton(menu_button, 500, 200, "Game")#assigns the MenuButton class to the image, position and Text values specific to this instance
#Will need to be repeated for other button additions


while menu == True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            GameButton.checkForInput(pygame.mouse.get_pos())

    screen.fill(BISQUE)

    GameButton.update()
    GameButton.HighlightByMouseHover(pygame.mouse.get_pos())

    pygame.display.update() #updates with feedback

def draw_background():
    screen.fill(BISQUE)
    pygame.draw.line(screen,BLUE,(60,80),(130,100))
    pygame.display.flip()
```

Results so far:

Its current capabilities include producing a rectangle with the text ‘Game’ inside itself, when the mouse hovers above it, the colour changes to red, and once clicked, to confirm that pygame has received a response, it can be seen in the terminal “button response confirmed”.

```
SettingsButton = MenuButton(menu_button,500, 400, "Settings")
QuitButton = MenuButton(menu_button,500,600,"Quit")


while menu == True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:#this bit ensures that the buttons themselves respond to the given input
            GameButton.checkForInput(pygame.mouse.get_pos())#GameButton will check if it is clicked, if so, will output checkforinput value test
            SettingsButton.checkForInput(pygame.mouse.get_pos())
            QuitButton.checkForInput(pygame.mouse.get_pos())

    screen.fill(BISQUE)

    #ensures all update with highlight animation
    GameButton.update()
    GameButton.HighlightByMouseHover(pygame.mouse.get_pos())

    SettingsButton.update()
    SettingsButton.HighlightByMouseHover(pygame.mouse.get_pos())

    QuitButton.update()
    QuitButton.HighlightByMouseHover(pygame.mouse.get_pos())

    pygame.display.update() #updates with feedback
```

In adding the other buttons, I also made it so they achieved the same result in the terminal. Each individual button has been given its own function with its own game loop, so that once one has been activated, the game plays within that loop.



Although this feedback is useful, upon further testing I realised there was a few issues with my code, including my 'font' variable clashing with the main SysFont variable, as well as this, there was a few continuity issues with two conflicting loops. To solve this, I created a return True after the checkforinput variable. However, there also appeared to be an issue on line 160, where the main_menu() function could not interpret the parameters where I tried to define the button itself from within the loop, so the way this was resolved was by removing this, and pre-setting the variable outside of the loop, referencing it under its own individual value. This specific value is what is then referenced inside the parameter in the loop.

Error messages, after this was amended:

```
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 172, in <module>
    Game()
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 83, in Game
    GAME_TEXT = menu_font(45).render("Main game goes here.", True, "Black")#shows the text to be displayed here
TypeError: 'pygame.font.Font' object is not callable
Traceback (most recent call last):
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 166, in <module>
    Game()
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 83, in Game
    GAME_BACK.changeColor(GAMEBUTTON_MOUSE_POS)
AttributeError: 'MenuButton' object has no attribute 'changeColor'
when i press the Game button: Traceback (most recent call last):
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 164, in <module>
    Game()
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 84, in Game
    GAME_BACK.update(screen)
TypeError: update() takes 1 positional argument but 2 were given
Traceback (most recent call last):
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py",
line 164, in <module>
```

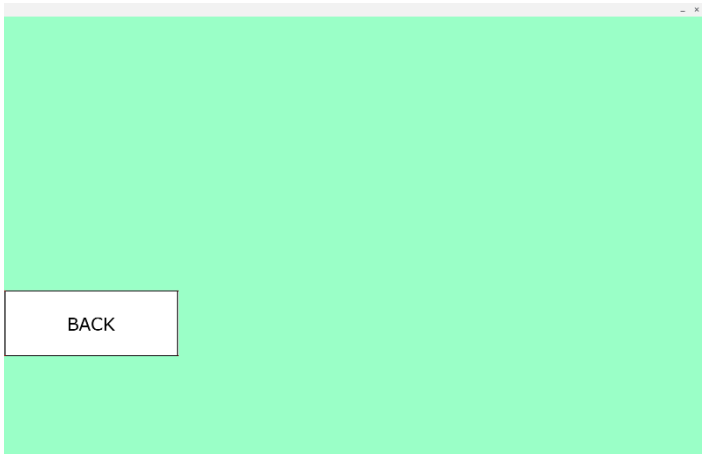
As you can see, as described above, the 'font' variable was clashing, and thus this had to be renamed and reassigned new values.

Solution prototype:

The main game loop got moved down to the bottom of the python file, so that I could call back the other functions, without two conflicting parameters trying to overwrite one another, with mismatched co-ordinates.

```
        for button in [GameButton, SettingsButton, QuitButton]:
            button.HighlightByMouseHover(MENU_MOUSE_POS)
            button.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GameButton.checkForInput(MENU_MOUSE_POS):
                    Game()
                if SettingsButton.checkForInput(MENU_MOUSE_POS):
                    Settings()
                if QuitButton.checkForInput(MENU_MOUSE_POS):
                    pygame.quit()
                    sys.exit()

        pygame.display.update()

while menu == True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:#this bit ensures that the buttons themselves respond to the given input
            if GameButton.checkForInput(pygame.mouse.get_pos()):#GameButton will check if it is clicked, if so, will output checkforinput value test
                Game()
            if SettingsButton.checkForInput(pygame.mouse.get_pos()):
                Settings()
            if QuitButton.checkForInput(pygame.mouse.get_pos()):
                pygame.quit()
                sys.exit()

    screen.fill(BISQUE)

    #ensures all update with highlight animation
    GameButton.update()
    GameButton.HighlightByMouseHover(pygame.mouse.get_pos())

    SettingsButton.update()
    SettingsButton.HighlightByMouseHover(pygame.mouse.get_pos())

    QuitButton.update()
    QuitButton.HighlightByMouseHover(pygame.mouse.get_pos())

    pygame.display.update() #updates with feedback
```

Now, once either 'Game' or 'Settings' had been clicked, a new menu appeared, with a 'back' button to revert back to the previous menu. By using a function with its own game loop, it meant that this could easily be stopped to recall previous game loops. The result is shown in the screenshot produced from this.



Stage 1

Test 4 – Timer Mechanics Prototype version 1

In the screenshot below, you can see that the timermechanics has been added, with a 00:00 format, this makes the timer count down in 1 second increments, in the same way described in the targets in the Analysis section, this will also be able to be displayed and run when the game() function is run after the button is pressed. The 'seconds' variable was set to 59, so as to prevent any continuity issues once the timer hits 0, after reaching the next minute.

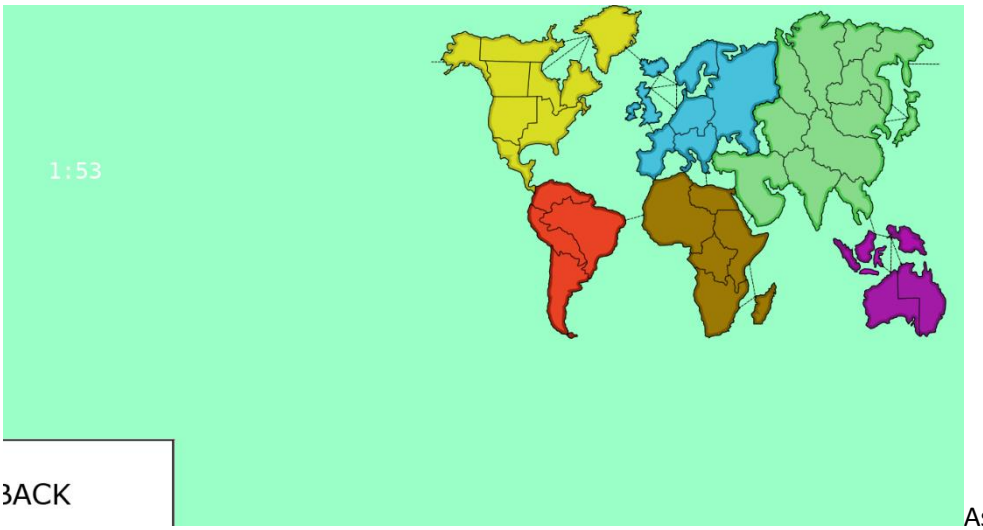
```
                    pygame.quit()
                    sys.exit()

        pygame.display.update()

def timermechanics():#sets up the mechanics for the timer to count down in 1 second increments in a 00:00 format
    Timer = True
    attackphase = True
    while Timer == True and attackphase == True:#while the timer is activated, whilst the attack phase is available
        seconds = 0
        minutes = 2 #set the seconds to 120 seconds, this should default to 2 minutes as shown in the code below
        timer_font = pygame.font.SysFont('Consolas',32)
        timer_text = timer_font.render("{}:{}".format(minutes,seconds),True,('White'),(BISQUE))

        timer_rect = timer_text.get_rect()
        timer_rect.center = 500//2, 500//2

        clock = pygame.time.Clock()

        while not finished:
            clock.tick(1)
            seconds -=1
            screen.blit(timer_text,timer_rect)
            if seconds > 60 or seconds < 0 and minutes != 0:
                seconds = 59
                minutes -=1
            timer_text = timer_font.render("{}:{}".format(minutes,seconds),True,('White'),(BISQUE))

        while seconds != 0 and seconds<0:
            timer = False
            #insert phase mechanic under if,elif or else


            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    quit()

            pygame.display.update()
```


<- Timer result

```
troop_button = pygame.image.load("neutral.png")
troop_button = pygame.transform.scale(troop_button, (200,75))

#map loading
map = pygame.image.load("riskmap.svg")

#loads all the buttons onto the screen, defining their location and text
GameButton = MenuButton(menu_button, 800, 300, "Game")#assigns the MenuButton class to the image, position and Text values specific to this instance
SettingsButton = MenuButton(menu_button,800, 500, "Settings")
QuitButton = MenuButton(menu_button,800,700,"Quit")

ShowTroopVal = ButtonIdentifier(troop_button, 100,100, "x")

#this function's purpose is to flip between the Game screen and menu screen once this function is activated from the button associated being pressed
def Game():
    while True:

        GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()#location of the mouse pointer is assigned to this variable, that is only valid whilst Game() has been activated


        screen.fill(BISQUE)#the screen of the background is filled with bg colour default


        GAME_TEXT = MenuButton(menu_button, 500, 200, "Main Game Goes Here...")#shows the text to be displayed here
        #content of the page goes in here
        screen.blit(map,(750,0))#fills in the map onto the screen
        GAME_BACK = MenuButton(menu_button, 200, 700, "BACK")#menu_button(image=None, pos=(640, 460), text_input="BACK", font=menu_font(75), base_color="White", hovering_color="Green")


        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()


        timermechanics()


        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
                    main_menu()

        pygame.display.update()
```

This screenshot above shows the new images created to define the map, as well as the updates made to the main game loop; however, since the addition of the timermechanics() function inside of the main game loop, the Back Button was rendered unusable, and thus this prototype disrupts all the other events taking place in the main game loop, therefore, the conclusion I have drawn from this is to return to this at a later stage of development.

Test 5 - The map is able to load, with all the buttons in the correct locations



As you can see from this screenshot, there are scaling issues with the map, and button positions. The screen is filled in first, followed by the map, then the timer, then the button. This was done so as to prevent overlapping of features, as shown in the below screenshot. The 'ShowTroopVal', highlighted below, also created the instance of a grey button (by default), so that the button for the main game code can be created. The text inside this box is currently assigned to 'x', however this will be changed later on when the troopvalues are created in co-ordinance with the deploy() phase yet to be added. As well as this, a new subclass of MenuButton, named 'ButtonIdentifier' was helped to amend scaling issues and assign ShowTroopVal as a variable defining this.

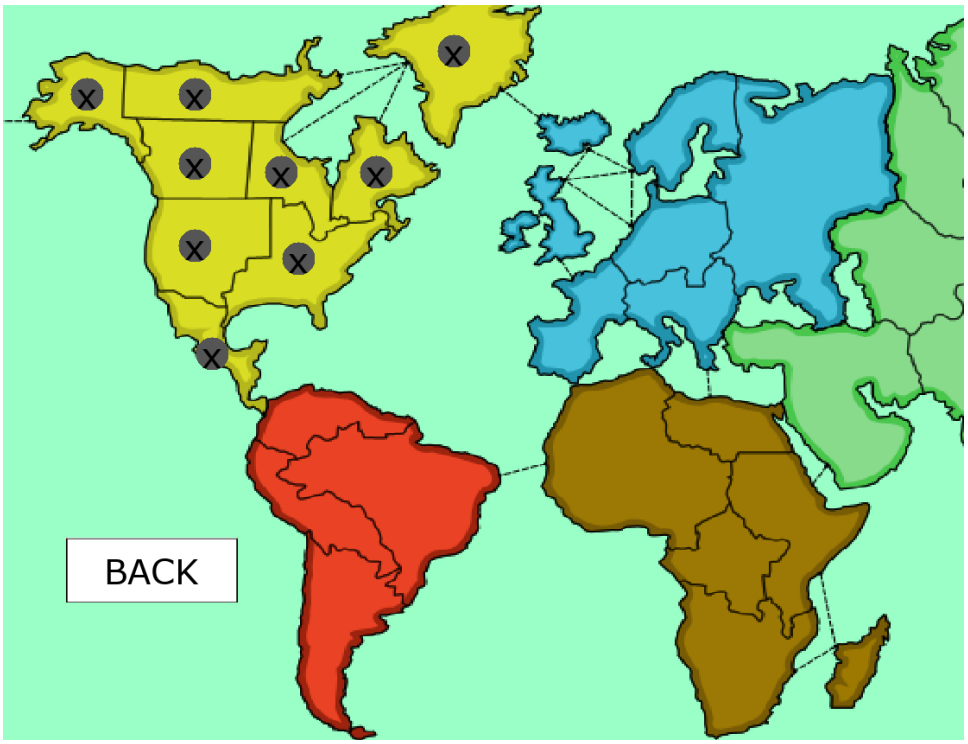
```
troop_button = pygame.image.load("neutral.png")
troop_button = pygame.transform.scale(troop_button, (200,75))

#map loading
map = pygame.image.load("riskmap.svg")

#loads all the buttons onto the screen, defining their location and text
GameButton = MenuButton(menu_button, 800, 300, "Game")#assigns the MenuButton class to the image, position and Text values specific to this instance
SettingsButton = MenuButton(menu_button,800, 500, "Settings")
QuitButton = MenuButton(menu_button,800,700,"Quit")

ShowTroopVal = ButtonIdentifier(troop_button, 50,50, "x")#this is a button so that a pop-up will form

#this function's purpose is to flip between the Game screen and menu screen once this function is activated from the button associated being pressed
def Game():
    while True:

        GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()#location of the mouse pointer is assigned to this variable, that is only valid whilst Game() has been activated


        screen.fill(BISQUE)#the screen of the background is filled with bg colour default


        GAME_TEXT = MenuButton(menu_button, 500, 200, "Main Game Goes Here...")#shows the text to be displayed here
        #content of the page goes in here

        screen.blit(map,(750,0))#puts the map onto the screen
        GAME_BACK = MenuButton(menu_button, 200, 700, "BACK")#menu_button(image=None, pos=(640, 460), text_input="BACK", font=menu_font(75), base_color="White", hovering_color="Green")


        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()

        ShowTroopVal = ButtonIdentifier(troop_button, 800,100,"x")#x needs replacing with the integer from the troop count


        ShowTroopVal.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        ShowTroopVal.update()
```

To fit the image, the screen size scale was changed from SCREEN_WIDTH * 0.8 to SCREEN_WIDTH * 0.6, as scaling was fixed. To add to this, another subclass of MenuButton was created, this time with GameButtonClass, so that the back button can be of a smaller scale than that of the menu buttons, whilst also retaining the same attributes. Screen.blit(map) is also used to bring the image up to the forefront of the screen.

For now, to resolve the issue of the back button not working, the timermechanics() has been code commented out, and added elsewhere. This will be revisited later on in the project, so as to ensure that this does not continually create issues throughout development.

```
        screen.blit(map,(0,0))#puts the map onto the screen
        GAME_BACK = GameButtonClass(game_button, 200, 700, "BACK")#in format image, y,x,text


        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()

        #defining the territories
        Alaska = Territory(troop_button,125,150,"x")
        NWTerritory = Territory(troop_button,250,150,"x")
        Alberta = Territory(troop_button,250,230,"x")
        Ontario = Territory(troop_button,350,240,"x")
        Greenland = Territory(troop_button,550,100,"x")
        WUS = Territory(troop_button,250,325,"x")
        Quebec = Territory(troop_button,460,240,"x")
        EUS = Territory(troop_button,370,340,"x")
        Central_America = Territory(troop_button,270,450,"x")

        #now needs: Eastern US, Central America


        Alaska.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        Alaska.update()
        NWTerritory.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        NWTerritory.update()
        Alberta.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        Alberta.update()
        Ontario.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        Ontario.update()
        Greenland.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        Greenland.update()
        WUS.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        WUS.update()
        Quebec.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        Quebec.update()
        EUS.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        EUS.update()
```

In the screenshot above is the addition of the buttons used to represent the territories of North America, and their placement on the map. Currently the text input is set to “x”, however, this can be amended once the deploy phase is set up, with each location using its own troop count. As well as this, a new class called territory has been created, which is a subclass of ‘buttonidentifier’, this was done so that each location can have its individual characteristics. The output is as follows:



To clarify whether it has detected the mouse pointer, the same highlighting function has been used to change the text colour from black to red, so that it is easier to follow through the development process. For testing, this region will be used as the baseline throughout the development of the phase rules, due to its size and interconnectivity.

Test 6 - The buttons are coloured accordingly upon distribution

```
territories = [
                "Alaska",    "NW Territory",    "Greenland",    "Alberta",    "Ontario",
                 "Quebec",    "Western United States",    "Eastern United States",    "Central America"]

neutral_territories = [
                "Alaska",    "NW Territory",    "Greenland",    "Alberta",    "Ontario",
                 "Quebec",    "Western United States",    "Eastern United States",    "Central America"]

p1_territories = []

#creating the territory randomiser
def territorydistribution():
    territorycount = 0
    if len(neutral_territories) > 0:
        while territorycount !=5:
            random_territory = random.choice(neutral_territories)
            neutral_territories.remove(random_territory)
            p1_territories.append(random_territory)
            territorycount += 1
            #testing out the response
            print(p1_territories,territorycount)
            print(neutral_territories)
```

As you can see from the screenshot above, I have defined the list above, with 'territories' (currently only holding the values of the locations of which I have put buttons on in the GUI. As well as neutral_territories, as a default. This list will be used to remove any elements that fall under the player or enemy territories, so that the locations not picked from this list remain the default neutral colours. The Player1 territory list is empty, so that the elements can be added upon distribution, making use of the random function, which will iterate a specified amount of times, so that the player gains half the list (this is necessary only for testing, these values will be changed as more locations are added).

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
                    main_menu()

        pygame.display.update()

territories = [ "Alaska",    "NW Territory",    "Greenland",    "Alberta",    "Ontario",
                "Quebec",    "Western United States",    "Eastern United States",    "Central America"]

neutral_territories = [
                Territory('Alaska',125,150,"x","ally.png"),Territory('NW',250,150,"x","ally.png"),Territory('Greenland',550,100,"x","ally.png")
                Territory('Ontario',350,240,"x","ally.png"),Territory('Quebec',460,240,"x","ally.png"),Territory('WUS',250,325,"x","ally.png"),
```

After having added the class objects inside the list, including their image, co-ordinates and class type, however, although this has helped in changing the images of the objects dependant on the lists they are placed in, there is a resulting error in relation to an earlier part of the code.

```
pygame 2.1.2 (SDL 2.0.16, Python 3.9.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/module_tester", line 168, in <module>
    ally_button = pygame.transform.scale(ally_button(40,40))
TypeError: 'pygame.Surface' object is not callable
```

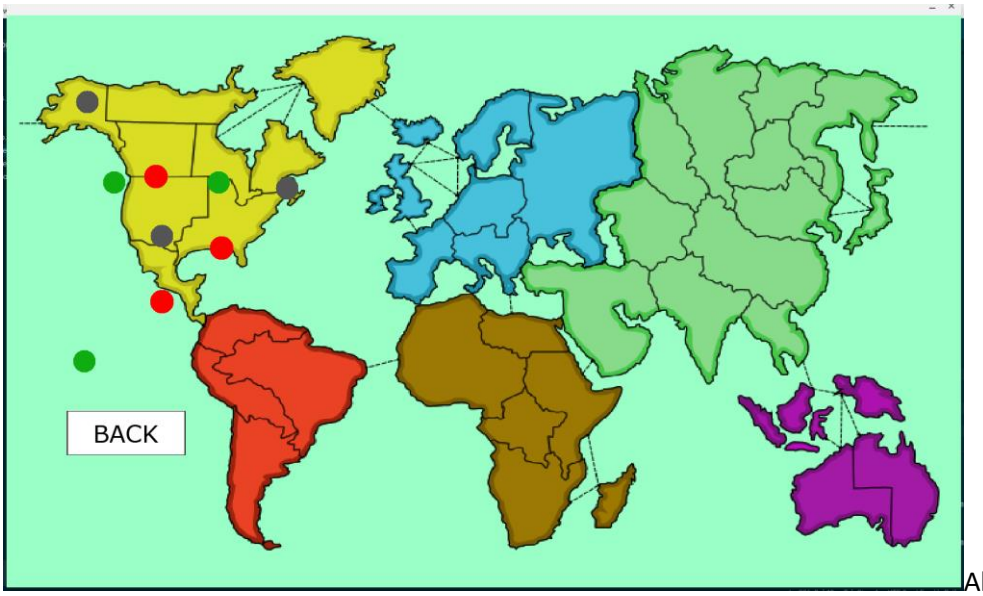
Once I had coded in the visual image for the different button types, to solve the various errors in the distribution code, I added the following code into the main game loop:

```
    while True:
        #pygame.timer.delay(1)
        #timer += 1

        #timermechanics(timer)

        GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()#location of the mouse pointer is assigned to this variable, that is onl


        screen.fill(BISQUE)#the screen of the background is filled with bg colour default



        #GAME_TEXT = MenuButton(menu_button, 500, 200, "Main Game Goes Here...")#shows the text to be displayed here
        #content of the page goes in here

        screen.blit(map,(0,0))#puts the map onto the screen
        GAME_BACK = GameButtonClass(game_button, 200, 700, "BACK")#in format image, y,x,text


        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()

        for territory in territories:
            if territory in p1_territories:
                screen.blit(ally_button, (territory.x_pos, territory.y_pos))
            elif territory in enemy_territories:
                screen.blit(enemy_button, (territory.x_pos, territory.y_pos))
            elif territory in neutral_territories:
                screen.blit(troop_button, (territory.x_pos, territory.y_pos))
```

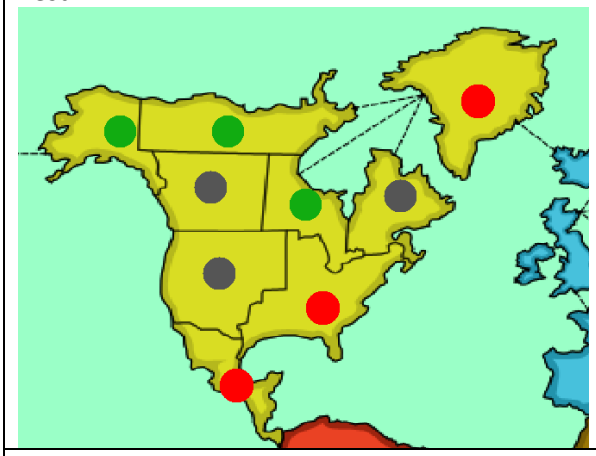
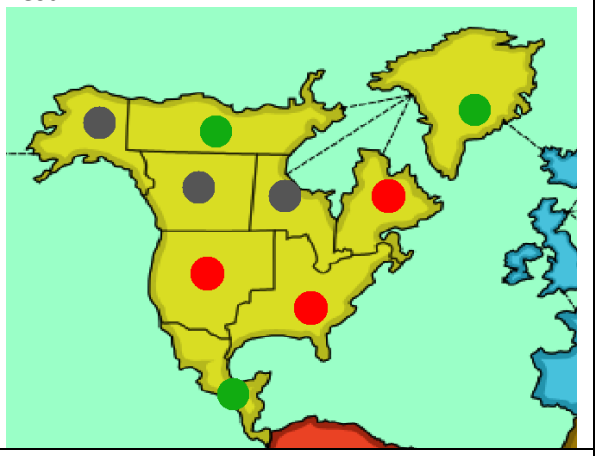
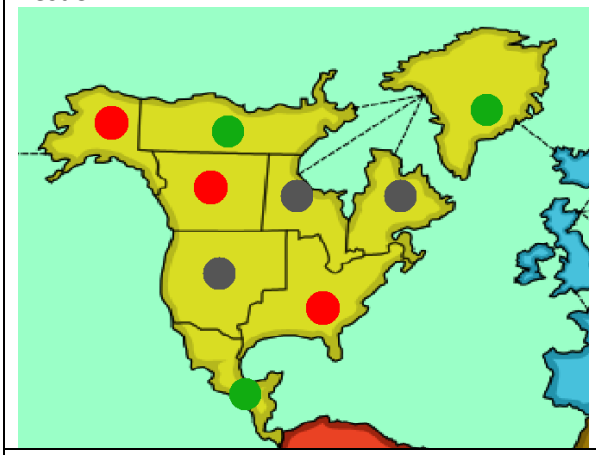
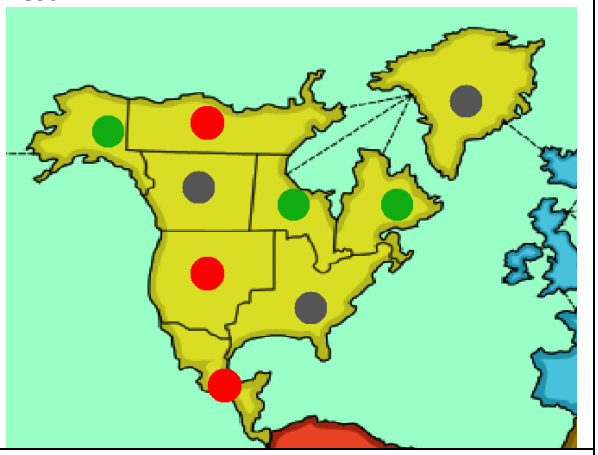
This was done with the purpose of displaying the various buttons onto the screen once the game had been opened. To have this saved upon opening the game each time and not iterating the distributionmechanics() function, I changed the function itself to iterate 3 times, distributing between ally, neutral and enemy.

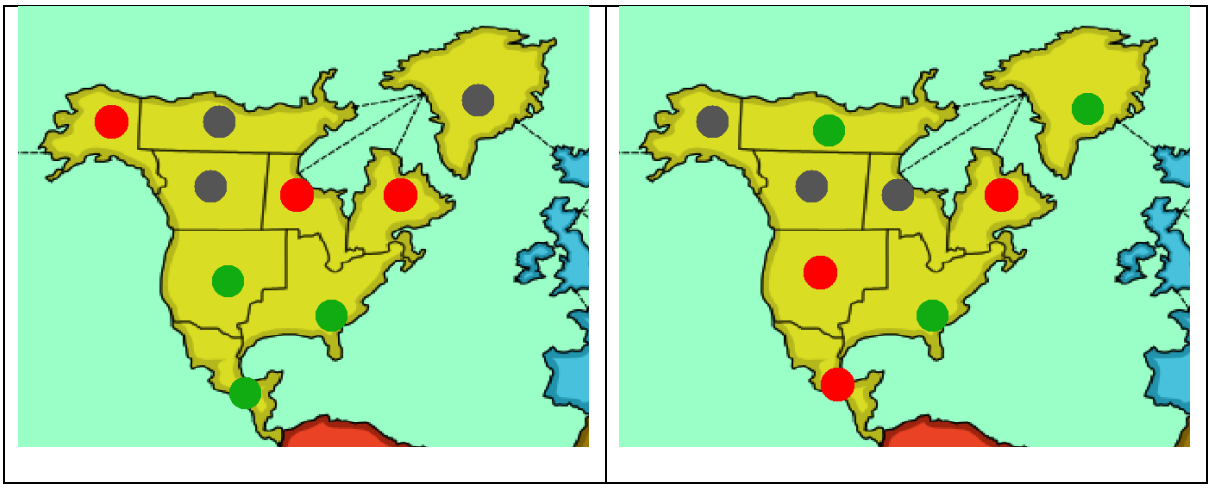
```
#creating the territory randomiser
def territorydistribution():
    global dist_done
    dist_done = True
    territorycount = 0
    if len(neutral_territories) > 0:
        while territorycount !=3:
            random_territory = random.choice(neutral_territories) #for the random territories of the ally
            neutral_territories.remove(random_territory)
            p1_territories.append(random_territory)
            #territorycount += 1
            random_enemy = random.choice(neutral_territories) #for the random territories of the enemy
            neutral_territories.remove(random_enemy)
            enemy_territories.append(random_enemy)
            territorycount += 1
            #testing out the response
            # print(p1_territories,territorycount)
            # print(neutral_territories)
```

Test Screenshot:



Although the formatting is not great, this can easily be resolved by changing the x and y co-ordinates. To test if this is truly random, I will set up a test table below with 6 different instances of the randomiser. (After the co-ordinates have been replaced into their original spacing)

Test 1 	Test 2 
Test 3 	Test 4 
Test 5	Test 6



Now that this has been completed, I will focus on getting the troop distribution mechanics. In accordance to the main game rules, each player starts with 40 troops each. With the neutral territory, the troops are evenly spaced, whereas on the other territories, this will have to meet the base rules of at least one troop per owned territory, with a random value on each location.

Test 7 – The troops are distributed accordingly from setup

```
def troopdistribution():
    #this will be used to distibute the troops randomly
    while p_start_troops >0:
        for territory in territories:
            if territory in p1_territories:
                currenttroopval = p_start_troops - (troop_minimum * len(p1_territories)) #calculates how many troops remain from the original start to
                randomiser = random.randint(0,currenttroopval)
                currenttroopval = currenttroopval-randomiser
                chosen_territory = random.choice(p1_territories)
                for territories in p1_territories:
                    chosen_territory[Territory.text] = Territory.text + randomiser #supposed to assign this new calculated random value to the class'
                    screen.blit(ally_button(territory.x_pos,territory.ypos,territory.text))
                    p_start_troops = currenttroopval #ensures that the loop can escape once this has been done, so that a conclusion can be reached.
                    #this is done towards the end of the loop
```

The purpose of implementing this in the way above (prototype 1) is so that I can create the base rules of troop distribution, with ensuring that there is a maximum initial setup troops (set to 40), which will ensure at least 1 troop is on each territory, hence the ‘troop_minimum’ variable being multiplied by the number of territories residing the player’s list, leaving the total as the value from which the randomiser can choose from.

Result:

```
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/module_tester", line 191, in <module>
    Alaska = Territory(troop_button,125,115,"x")
TypeError: __init__() missing 1 required positional argument: 'troopcount'
_abihinch@penguin:~/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code$
```

Before:

```
class Territory(ButtonIdentifier):
    def __init__(self, image, y_pos,x_pos,text_input,troopcount):
        super().__init__(image, x_pos, y_pos, text_input)
        self.image = image
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.circle = self.image.get_rect()
        self.text_input = text_input
        self.text = menu_font.render(self.text_input,True,"Black")
        self.troopcount = troopcount #the value for defining the number of troops that the territory has
        # self.continent = continent
```

As shown in this and the following screenshot, the addition of ‘troopcount’ inside the class Territory’s parameters has been needlessly defined, thus the removal of this will fix this problem.

After:

```
class Territory(ButtonIdentifier):
    def __init__(self, image, y_pos,x_pos,text_input):
        super().__init__(image, x_pos, y_pos, text_input)
        self.image = image
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.circle = self.image.get_rect()
        self.text_input = text_input
        self.text = menu_font.render(self.text_input,True,"Black")
```

Test Attempt 2:

```
def troopdistribution():
    #this will be used to distibute the troops randomly
    global p1_startup
    p1_startup = True
    global p_start_troops
    while p_start_troops >0:
        for territory in territories:
            if territory in p1_territories:
                currenttroopval = p_start_troops - (troop_minimum * len(p1_territories)) #calculat
                randomiser = random.randint(0,currenttroopval)
                currenttroopval = currenttroopval-randomiser
                chosen_territory = random.choice(p1_territories)
                for chosen_territory in p1_territories:
                    chosen_territory[chosen_territory.text] = chosen_territory.text + randomiser #
                    screen.blit(ally_button(territory.x_pos,territory.ypos,territory.text))
                    p_start_troops = currenttroopval #ensures that the loop can escape once this ha
                    #this is done towards the end of the loop
```

```
Traceback (most recent call last):
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/module_tester", line 426, in <module>
    Game()
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/module_tester", line 223, in Game
    troopdistribution()
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/module_tester", line 315, in troopdistribution
    chosen_territory[chosen_territory.text] = chosen_territory.text + randomiser #supposed to assign this new calculated random value to the class' attributes' troopcount
TypeError: unsupported operand type(s) for +: 'pygame.Surface' and 'int'
```

After this error message, it becomes clear that to resolve this issue would be to create a new prototype, which is able to run off of a system that creates separate lists per player type, from which must loop until specified conditions have been met, then assigned on screen.

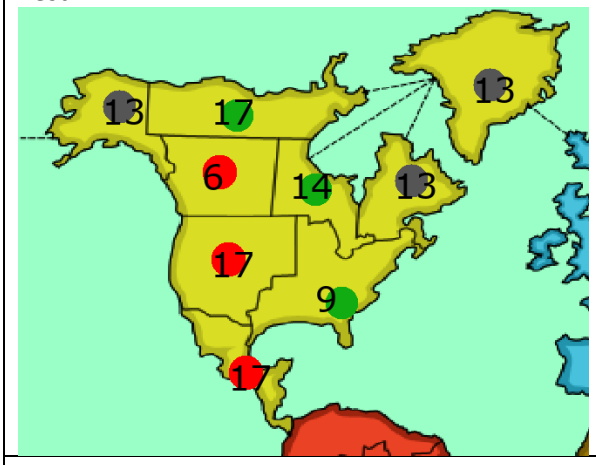
```
def troopdistribution():
    #this will be used to distibute the troops randomly
    global p1_startup
    p1_startup = True
    global p_start_troops, e_start_troops, n_start_troops #Defines the variables for the number of starting troops per player

    p1_territory_troop_vals = [0] * len(p1_territories) #Creates a list of the troopvals foe each territory owned by p1
    enemy_territory_troop_vals = [0] * len(enemy_territories)

    while sum(p1_territory_troop_vals) != p_start_troops: #loop until total troopval is equal to starting number (p1)
        for idx in range(len(p1_territory_troop_vals)): #loops through each p1 territory, randomly assinging troopvals
            p1_territory_troop_vals[idx] = random.randint(1, p_start_troops-len(p1_territories)+1)

    for idx, territory in enumerate(p1_territories):#Assign troopval to p1, update text on screen
        territory.troopval = p1_territory_troop_vals[idx]
        territory.update_troop_text()

    while sum(enemy_territory_troop_vals) != e_start_troops:#loop until total troopval is equal to starting number (enemy)
        for idx in range(len(enemy_territory_troop_vals)):#loops through each enemy territory, randomly assinging troopvals
            enemy_territory_troop_vals[idx] = random.randint(1, e_start_troops-len(enemy_territories)+1)

    for idx, territory in enumerate(enemy_territories):#Assign troopval to enemy, update text on screen
        territory.troopval = enemy_territory_troop_vals[idx]
        territory.update_troop_text()

    neutral_territory_troop_vals = [n_start_troops // len(neutral_territories)] * len(neutral_territories)
    #the line above creates a troop value list for each neutral territory where troops are equally distributed

    for idx, territory in enumerate(neutral_territories):#Assign troopval to neutral territories, updating the text on screen
        territory.troopval = neutral_territory_troop_vals[idx]
        territory.update_troop_text()
```

Result:

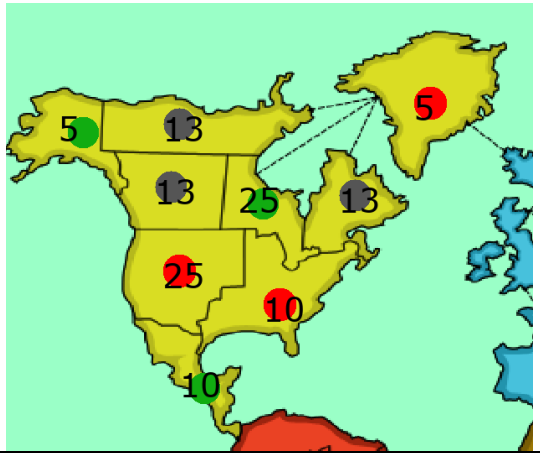
As you can see from the table below, with the running of a few tests in a similar fashion to the territory distribution tests, this module of the prototype is fully functional, following the base rules and not exceeding the starting troopcount. The neutral territories follow an equal amount of troops per territory (in this case 13), with ally and

enemy territories ranging their troopcount, without any errors in count of 0 or below (test 3 and test 6 showing good examples of this).

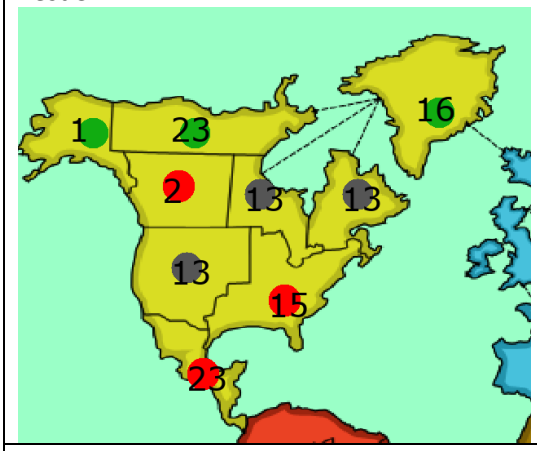
Test 1



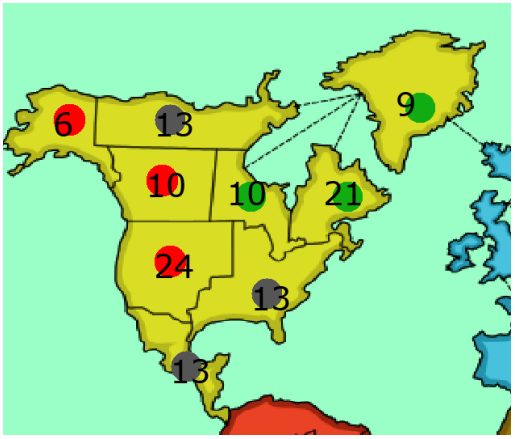
Test 2



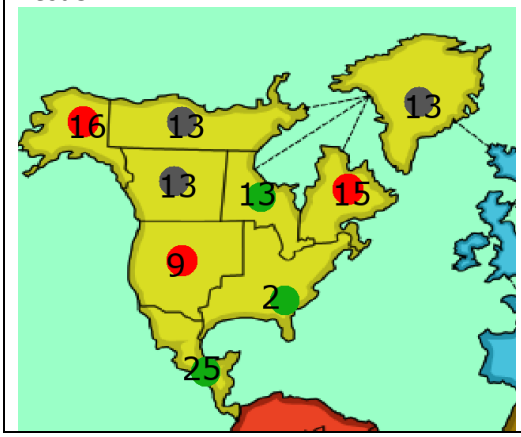
Test 3



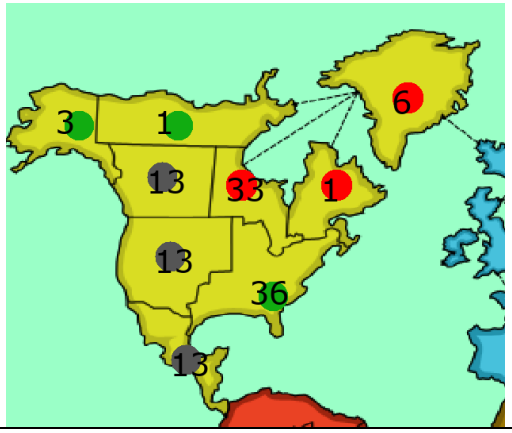
Test 4



Test 5



Test 6



Test 8 - Timer prototype version 2 & 3

```
def timermechanics():
    # start the timer
    timer = 10
    timer_started = False

        # check if the timer has started
    if not timer_started:
        timer_started = True
        start_time = pygame.time.get_ticks()

        # get the elapsed time
    elapsed_time = (pygame.time.get_ticks() - start_time) / 1000

        # update the timer
    if elapsed_time >= 1:
        timer -= 1
        if timer < 0:
            timer = 0
            timer_started = False

            # reset the start time
        start_time = pygame.time.get_ticks()

        # display the timer on the screen
        screen.fill((255, 255, 255))
        timer_text = font.render(str(timer), True, (0, 0, 0))
        timer_rect = timer_text.get_rect()
        timer_rect.center = (SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2)
        screen.blit(timer_text, timer_rect)

        # update the display
        pygame.display.update()
```

In this first draft of the second timer prototype, I have made use of the pygame.Clock() module as opposed to the time module, as this should help to stop the interference with other events. The result of this should produce a blank screen with the timer overlaying the main screen, which can be exited at anytime. This is for the purpose of providing a standalone test.

The logic behind this involves 1000 milliseconds of ticks being equivalent to 1 second of time, so that it may calculate the elapsed time and 1 second increments as the same time scale, so that -1 can be removed from the total once one second of time has passed. Then, once the timer reaches 0, the timer is now 0 and stays 0.

To remove the test conditions, the ‘screen.fill’ will be removed once it is ensured that the timer works as intended.

Test result:

The output produced a countdown from 10 to 0, where the timer proceeded to stop counting down upon hitting 0. This is already an improvement upon the continuity issues in the first prototype 1, where the timer would continue to count down to minus figures.

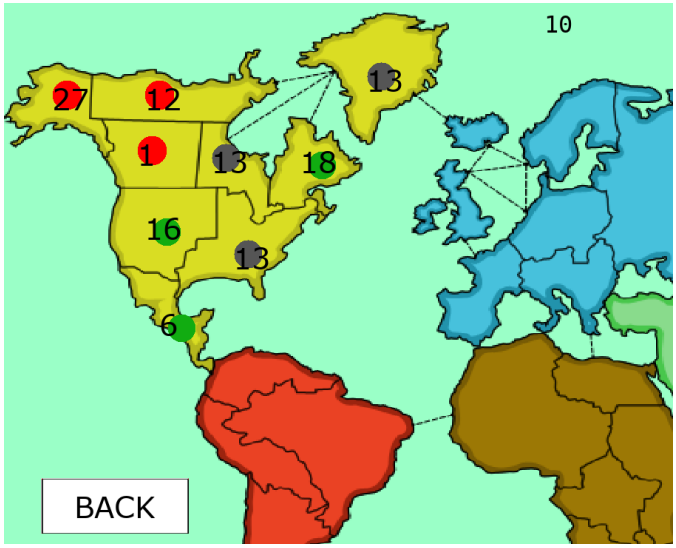
Now, having tested this prototype without the use of while loops and time.sleep modules, if I implement this prototype with similarities in prototype 1, it may be able to create a fully functional version of a timer, which will run in the background with the other processes. The best way to prove this would be to attempt to use the ‘back’ button once the timer has been initiated.

The expected consequences of these changes aim to:

1. Allow the user to interact with the user interface
2. Allow the timer to count down to 0, and nothing below
3. Create a more isolated effect of the timer

```
def timermechanics():
    # start the timer
    timer_started = True
    timer = 10

    # check if the timer has started
    if not timer_started:
        timer_started = True
    start_time = pygame.time.get_ticks()

        # get the elapsed time
    elapsed_time = (pygame.time.get_ticks() - start_time) / 1000

        # update the timer
    if elapsed_time >= 1:
            timer -= 1
    if timer < 0:
            timer = 0
            timer_started = False

            # reset the start time
    #start_time = pygame.time.get_ticks()

        # display the timer on the screen
    timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used
    timer_text = timer_font.render("{}".format(timer),True,('Black'),(BISQUE)) #allows the formatting some flexibility
    timer_rect = timer_text.get_rect()
    timer_rect.center = (SCREEN_WIDTH//2,50) #scales the size of the timer
    screen.blit(timer_text, timer_rect)#shows the timer on the screen

        # update the display
    pygame.display.update()


while menu == True:
    for event in pygame.event.get():
```

The formatting of code shown in this screenshot is very similar to that of the first prototype, but only with different blit formatting. A positive improvement to note is that the menu button text changes colour when I hovered the mouse above it, and it was able to revert back to the previous screen, showing that now it will not interrupt the other processes and I can interact with the interface. However, despite its appearance, the timer itself is now ceasing to count down, thus, an iterative approach may be required, from which the timer function is called every second.



Prototype 3:

Having understood the differences in both previous models, I have devised a broken-down approach to how this should be handled:

Problem 1: Interference/ allowing the timer to run in conjunction with the main game loop

Solution: Run the timer inside of the game() loop itself, instead of another function

Justification for this specific approach: The only other option, involving threading, would be far too complex, and would involve a complete rewrite of my code

Problem 2: Counting down once executed

Solution: pygame.clock() method, making use of the fps and millisecond approach from prototype 2, as the variable will always be changing from execution, it will not require an update command once blitted, instead just blitting as the variable value changes.

Problem 3: Not reducing to any count below 0.

Solution: A max function that allows the data shown to only be within the ranges of 0-60

Problem 4: The time/second ratio

Solution: Similar to the design from prototype 1, instead of writing out the code through individually defining the time segments, making use of the divmod function would return the result of the integer division and the modulus of seconds and minutes.

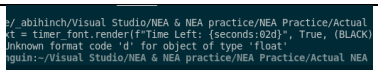
Justification for this specific approach: Reduces the amount of code required for the overall operation of this mechanic.

Having identified the primary targets for this, I have created the following:

```
timer = pygame.USEREVENT + 1
pygame.time.set_timer(timer,1000) #Fires every 1000 milliseconds/ 1 second
time_remaining = 60
```

```
# Substract time since last frame from remaining time
time_remaining -= clock.tick(FPS) / 1000

# Split the remaining time into minutes and seconds. divmod function returns the result of the integer division (minutes) and the mo
# max function ensures timer does not go negative
minutes, seconds = divmod(max(0, time_remaining), 60)
```

Blit tests:

Code	*Result*	*Additional notes*
`timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used` `timer_text = timer_font.render(f"Time Left: {seconds}", True, (BLACK))` `timer_rect = pygame.Rect((400, 200), (1, 1))` `screen.blit(timer_text, timer_rect)`		The timer now functions alongside everything else, however, the float format needs to be changed to 2 decimal places
`timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used` `timer_text = timer_font.render(f"Time Left: {seconds:02}", True, (BLACK))` `timer_rect = pygame.Rect((400, 200), (1, 1))` `screen.blit(timer_text, timer_rect)`		Not much notable change, other than the wrong conversions
`timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used` `timer_text = timer_font.render(f"Time Left: {seconds:02d}", True, (BLACK))` `timer_rect = pygame.Rect((400, 200), (1, 1))` `screen.blit(timer_text, timer_rect)`		:02f is the wrong format, only usable for integers, not float
`timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used` `timer_text = timer_font.render(f"Time Left: {seconds:.2f}", True, (BLACK))` `timer_rect = pygame.Rect((400, 200), (1, 1))` `screen.blit(timer_text, timer_rect)`		Works fully as intended, to 2 decimal places

Now that I know this works, all that needs to be done to finalise this prototype is readjust it to the top centre of the screen.

Test 9: The timer, upon reaching 0, goes into the next phase's timer

To setup the phase mechanics, conditionals and events need to be setup, this can be done with the addition of deployphase, attackphase and fortifyphase variables, all of which are of a boolean value. The addition of other buttons such as 'end turn' and 'card trade in' are to implemented at this stage, to work in conjunction with the phases.

End_Turn Mechanic:

```
if timer_active == True:
    # Substract time since last frame from remaining time
    time_remaining -= clock.tick(FPS) / 1000
# Split the remaining time into minutes and seconds. divmod function returns the result of the integ
# max function ensures timer does not go negative
    minutes, seconds = divmod(max(0, time_remaining), 120)
    timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used
    timer_text = timer_font.render(f"Time Left: {seconds:.2f}", True, (BLACK))

    timer_rect = pygame.Rect((650, 50), (1, 1))
    screen.blit(timer_text, timer_rect)
```

In order to be able to set the timer to 0, the timer function would need to be included inside of an 'if' statement. With this, this can then ensure that while deploy is active, the timer does not require blitting onto the screen.

```
if attackphase == True:
    timer_active = True
    deployphase = False
    END_TURN = GameButtonClass(elongated_game_buttonv1, 500, 900, "END TURN")
    #time_remaining = 10
    if time_remaining == 0:
        fortifyphase = True
elif fortifyphase == True:
    attackphase = False
    #time_remaining = 5
    END_TURN = GameButtonClass(elongated_game_buttonv1, 500, 900, "END TURN")
    if time_remaining == 0:
        deployphase = True
elif deployphase == True:
    fortifyphase = False
    timer_active = False
```

This is the first version of the phases, currently, they only offset one another, however, the end_turn button will need to be given a function of setting the timer to 0, so that the next phase is able to execute. This can be implemented inside of the event loop in the game() function:

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    if event.type == pygame.MOUSEBUTTONDOWN:
        if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
            main_menu()
        elif END_TURN.checkForInput(GAMEBUTTON_MOUSE_POS):
            time_remaining = 0
            print("Collision confirmed")
        # elif Territory.checkForInput(self=Territory,position=GAMEBUTTON_MOUSE_POS):
        #   check_collision(GAMEBUTTON_MOUSE_POS,p1_territories)
        else:
            for territory in territories:
                if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS):
                    # Rest of code here
                    #check_collision()
                    chosen_list.append(territory)

                    for territory in chosen_list:
                        current_troopval = territory.troopval
                        current_name = territory.name
                        print(current_troopval, current_name)
```

For test purposes, a response will be output in the terminal to tell if the button has been registered.

Subsequent Feedback:

The end turn button works as intended.

Terminal output:

```
pygame 2.1.2 (SDL 2.0.18, Python 3.9.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
Collision confirmed
```

Test 10: When a new phase is executed, the new rules are met with the previous ones being overruled (Deploy)

Phase Timer Mechanism prototype (setup)

```
    if attackphase == True:
        attack_timer_active = True
        deployphase = False
        #time_remaining = 10
        if time_remaining <= 0:
            fortifyphase = True
            timer_active = False
            attackphase = False
            attack_timer_active = False
    elif fortifyphase == True:
        fortify_timer_active = True
        if fortify_time_remaining <= 0:
            deployphase = True
            fortifyphase = False
            fortify_timer_active = False
    elif deployphase == True:
        deploy()
```

Located inside of the main() game loop, this enables the timer for both the attack and deploy phase, with the different timers being essentially duplicates of themselves, but with different values. The reason the same timer was not used and the variables were not changed inside of this function was due to the nature of the while loop, which would have set the timer to the intended state, but then reset it without a deploy, leading to a permanent state where the timer would never count down.

```
if timer_active == True:
    # Substract time since last frame from remaining time
    time_remaining -= clock.tick(FPS) / 1000
# Split the remaining time into minutes and seconds. divmod function returns the result
# division (minutes) and the modulus of two inputs (seconds).
# max function ensures timer does not go negative
    minutes, seconds = divmod(max(0, time_remaining), 120)
    timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used
    timer_text = timer_font.render(f"Time Left: {seconds:.2f}", True, (BLACK))

    timer_rect = pygame.Rect((650, 50), (1, 1))
    screen.blit(timer_text, timer_rect)

elif fortify_timer_active == True:
    fortify_time_remaining -= clock.tick(FPS) / 1000

    minutes, seconds = divmod(max(0, fortify_time_remaining), 120)
    timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be used
    timer_text = timer_font.render(f"Time Left: {seconds:.2f}", True, (BLACK))

    timer_rect = pygame.Rect((650, 50), (1, 1))
    screen.blit(timer_text, timer_rect)
```

Using the data from my stakeholders, the attack timer is set to 70 seconds, whereas fortify is 30. This concludes the timer mechanics, and now I will focus on the deploy mechanics.

Territory Selection Development

In order to be able to interact with the territories whilst the various phases are active, a collision system will be required to detect if the user has interacted with the territory, as well as logging the troop value, so that they may be used later for comparisons, which is integral for the dice and statistics mechanics.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:…
    if event.type == pygame.MOUSEBUTTONDOWN:
        if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
            main_menu()
        elif END_TURN.checkForInput(GAMEBUTTON_MOUSE_POS):
            if attack_timer_active == True:
                time_remaining = 0
            elif fortify_timer_active == True:
                fortify_time_remaining = 0

                #if available_troops == 0:
                #   deployphase = False
                # attackphase = True
            #print("Collision confirmed")
        # elif Territory.checkForInput(self=Territory,position=GAMEBUTTON_MOUSE_POS):
        #   check_collision(GAMEBUTTON_MOUSE_POS,p1_territories)
        else:
            for territory in territories:
                if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #if the territory is clicked by the mouse
                    chosen_list.append(territory) #append it to a new list

                for territory in chosen_list: #for that one territory
                    current_troopval = territory.troopval #logs the troopvalue
                    current_name = territory.name #logs the name
                    print(current_troopval, current_name)
```

Located inside of the main game loop, when the mouse clicks on the territory object, the troopvalue is logged, as well as the name, for test purposes, the name has been outputted onto the screen. (the code commented inside the loop was from a previous failed attempt)

Output:

```
13 Central_America
13 Central_America
13 Central_America
```

(Clicked the same territory 3 times for reference, each time it only output the territory once, showing that it has not duplicated the same territory)

As this prototype is functioning, next I will allow the game to validate the clicks depending on what phase the game is currently in. The validation for user clicks will need to be as follows:

- Deploy: Player's territory only
- Attack: First click – only player territory, where troops must be 2 or above Second Click – only enemy territory, that must be neighbouring in one way or another.
- Fortify – Player's territory only, but can only move troops within the capacity of that territory, leaving at least one left on that territory, between neighbouring countries.

Giving the troops to players and validating where they can place the troops down. - DEPLOY

To outline the deploy phase, the mechanics must work as follows:

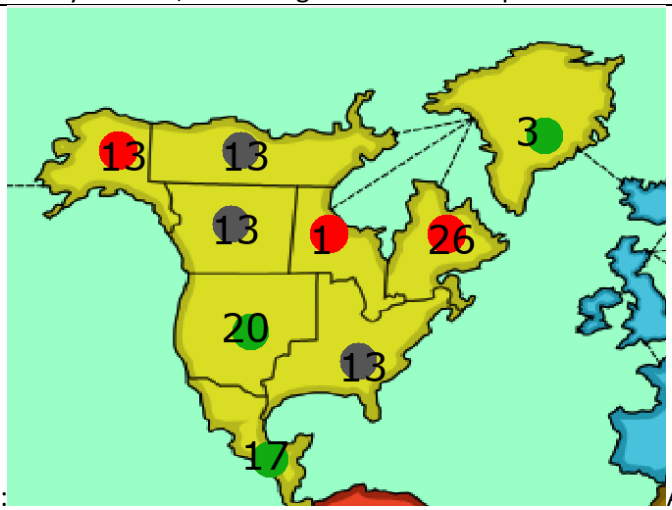
-At the start of every deploy turn, the player is given their troops, including any continental bonuses

- The trade card mechanic is available to gain additional troops

-There is no timer for this, but if there is no more troops to place down, the phase ends, jumping to the attack phase

-The neutral territory automatically gains half the troops each player gains per their turn, and these will be divided equally amid the territories.

Code:
```
def deploy():
    global default, NorthAmericaBonus
    #for territories in NAterritories:
    if  p1_territories in NAterritories:
        available_troops = default + NorthAmericaBonus
    else:
        NorthAmericaBonus = 0
        available_troops = default + NorthAmericaBonus
    print(available_troops)
```
NorthAmericaBonus is automatically set to 5, but changed to 0 if the requirements are not met.

Screen:


As you can see, the player does not own all the territories, therefore, the expected test output would be 3.

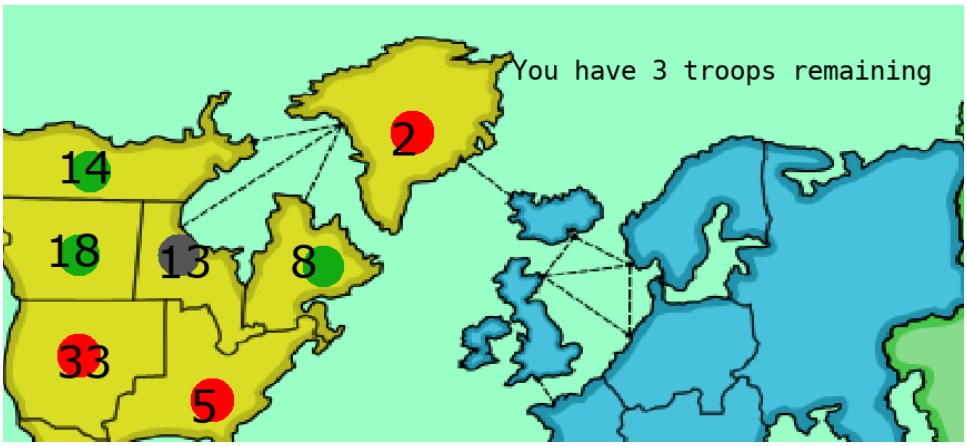
Output:
```
3
```

Putting it onto the screen:

```
elif fortify_timer_active == False and attack_timer_active == False and deployphase == True: #if the conditions are in place...
    deploy_font = pygame.font.SysFont('Consolas',24)
    deploy_text = deploy_font.render(f"You have {available_troops} troops remaining",True,(BLACK)) #formats the troops available
    screen.blit(deploy_text,timer_rect)#outputs the troops available
```
This created the conditions for it to be output, and the deployphase condition was updated, as I noticed I had wrongly indented an 'else' statement, which caused the available_troops to return as None. The fix is as follows:

Before:
```
def DeployTroops():
    global default, NorthAmericaBonus, available_troops,deployphase,attackphase,attack_timer_active
    #for territories in NAterritories:
    if  p1_territories in NAterritories:
        #if generate == True:
        available_troops = default + NorthAmericaBonus
        print(available_troops)
        if available_troops == 0:
            deployphase = False
            attackphase = True
            attack_timer_active = True
        else:
            NorthAmericaBonus = 0
            available_troops = default + NorthAmericaBonus
            print(available_troops)
```

After:
```
def DeployTroops():
    global default, NorthAmericaBonus, available_troops,deployphase,attackphase,attack_timer_active
    #for territories in NAterritories:
    if  p1_territories in NAterritories:
        #if generate == True:
        available_troops = default + NorthAmericaBonus
        #print(available_troops)
        if available_troops == 0:
            deployphase = False
            attackphase = True
            attack_timer_active = True
            #ensures that deploying stops
    else:
        NorthAmericaBonus = 0
        available_troops = default + NorthAmericaBonus
        if available_troops == 0:
            deployphase = False
            attackphase = True
            attack_timer_active = True
available_troops = DeployTroops()
```



Card Trade In mechanic setup

In setting up the Card mechanics, a new class would need to be created in the classes.py directory, the idea behind this is so that the players are given the cards, for which they then have the choice to trade in to gain further troop bonuses. The Card trade in button has also been given its own function to execute.

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:…
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
                    main_menu()
                elif END_TURN.checkForInput(GAMEBUTTON_MOUSE_POS):
                    if attack_timer_active == True:
                        time_remaining = 0
                    elif fortify_timer_active == True:
                        fortify_time_remaining = 0
                if deployphase == True:
                    if TRADE_IN.checkForInput(GAMEBUTTON_MOUSE_POS):
                        trade_in()
```

As you can see, now if the button is clicked, it sends the user to the trade_in loop, as detailed below, which has all the basic setup requirements for a new screen:

```
def trade_in():
    global GAMEBUTTON_MOUSE_POS
    while True:
        screen.fill (BISQUE)

        GAME_BACK = GameButtonClass(game_button, 200, 900, "BACK")#in format ima


        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()


        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if pygame.K_ESCAPE:
                    Game()
                #if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
                #   Game()
        pygame.display.update()
```

As shown below, you can see the beginnings of my cards class, this will include the image, as well as other essential data such as card type, and title, which will be further used for a more dynamic deploy phase for the purposes of troop variation.

```
class Cards():
    def __init__(self, image, type, titlename):
        self.image = image #the image
        self.rect = self.image.get_rect() #the clickbox
        self.type = type
        self.titlename = titlename #this is to identify if the title matches self.territoryname later on in development

    def draw(self, screen, x, y): #draws the object onto the screen
        self.rect.x = x
        self.rect.y = y
        #puts the image onto the screen
        screen.blit(self.image, self.rect)

    def check_for_input(self, GAMEBUTTON_MOUSE_POS):  #checks to see if object clicked by mouse
        return self.rect.collidepoint(GAMEBUTTON_MOUSE_POS)
```

The way the cards will work involves a queue system. When a card is received (through randomisation), it is then added to a main queue. This will be my first steps in achieving a card mechanics screen. This would require a slot size variable, as well as a variable recording the number of allowed slots in the main queue (15), and the number in the priority queue (3), as well as its various positions, and the fonts.

```
#Card mechanics variables
slot_size = 5 #the size of the individual slots
num_slots_main = 15 #number of slots in the main queue
num_slots_priority = 3 #number of slots in the priority queue
queue_position = (50, 300) #the main position of the queue
priority_queue_position = (500, 50)
capacity_indicator_position = (1100, 550) #position of the recording stuffs
capacity_font_size = 24
capacity_font = pygame.font.SysFont('Arial', capacity_font_size)
```

Another class, ‘Queues’, accompanies the cards, and this defines the simple addition and removal mechanics as the cards are swapped out between the different queues, based off user input, as can be seen below, this will be the main blueprint for the actual queues themselves.

```
class Queue:
    def __init__(self, num_slots, position):
        self.num_slots = num_slots #defines the number of available slots
        self.position = position
        self.slots = [] #defines the slots themselves as an empty list
        self.capacity_indicator = capacity_font.render(f"{len(self.slots)}/{self.num_slots}", True, (0,0,0))
        #creates a surface, for testing purposes, so that i may see if the cards fulfil its capacity limits

    def add(self, obj): #defines the adding mechanics between queues
        if len(self.slots) < self.num_slots: #checks to see if the slot number is less than the number of available slots
            self.slots.append(obj) #if there are sufficient slots available, append the object
            self.capacity_indicator = capacity_font.render(f"{len(self.slots)}/{self.num_slots}", True, (0,0,0)) #records the capac
            return True
        else:
            return False #if it exceeds capacity, do nothing

    def remove(self, obj): #defines the removal mechanics between queues
        if obj in self.slots: #if the object is to be found in the slot position
            self.slots.remove(obj) #remove the object
            self.capacity_indicator = capacity_font.render(f"{len(self.slots)}/{self.num_slots}", True, (0,0,0))

    def draw(self, screen):
        for i, obj in enumerate(self.slots):
            x = self.position[0] + (i % 5) * (slot_size + 200) #draw each card 200 x co-ordinates away from one another
            y = self.position[1] + (i // 5) * (slot_size + 200) #for each new line in the queues, add 200 y co-ords of space
            obj.draw(screen, x, y) #draw the objects onto the screen in their assigned places
        screen.blit(self.capacity_indicator, capacity_indicator_position) #put the objects onto the screen
```

To define the cards themselves is simple, as it follows the same principle as any other class object used in my game, and similar to the territory distribution mechanics, it follows a similar format to code.

```
def CardDistribution():
    global cards_done, main_queue,cards
    cards_done = True
    cardcount = 0
    if len(cards) > 0:
        while cardcount !=4: #set this to 1, then every time the turn begins again, set the cardcount back down to 0, so that it can enact this loop again
            #currently set to 4, for testing purposes on capacity
            random_card = random.choice(cards) #for the random cards of the ally
            cards.remove(random_card)
            plcards.append(random_card)
            main_queue.add(random_card)
            cardcount += 1
            print(cards)
            print(cardcount)
```

Currently, the cardcount is set to 4, so that I can test if capacity has been reached from the priority list once the Card Trade in function is coded, as well as this, the event loop has been modified in the trade in function, so that the changes can register.

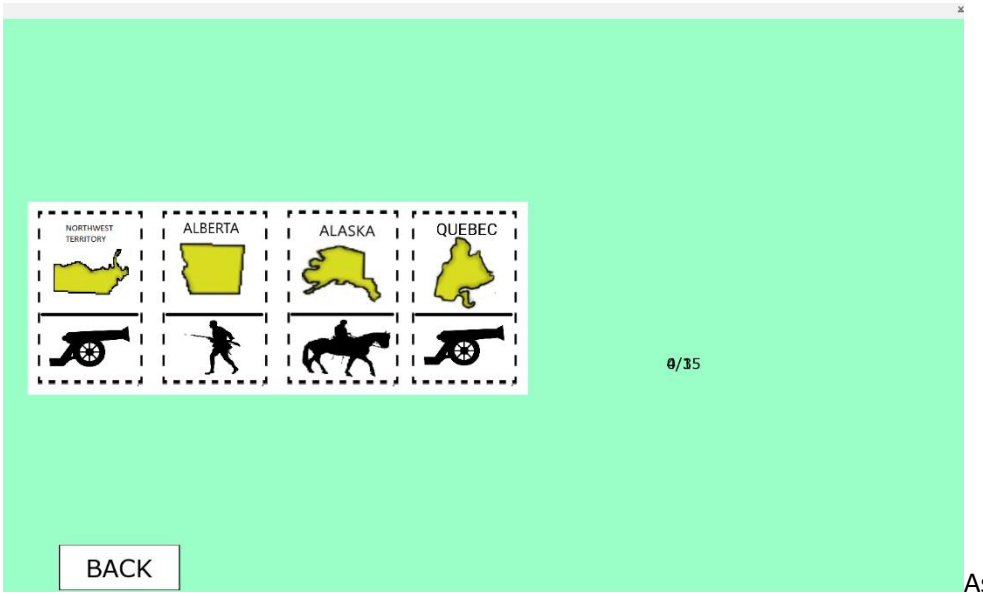
```
    GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
    GAME_BACK.update()


    main_queue = Queue(num_slots_main, queue_position) #defines the queue with 15 slots, and their positions
    priority_queue = Queue(num_slots_priority, priority_queue_position) #3 slots and their location

    CardDistribution()


    #clicked_obj = None
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            if pygame.K_ESCAPE:
                Game()
            #mouse_pos = pygame.mouse.get_pos()
        # Check if an object in the main queue was clicked
            for obj in main_queue.slots:
                if obj.check_for_input(GAMEBUTTON_MOUSE_POS):
                    #selected_obj = obj
                    priority_queue.add(obj)
                    main_queue.remove(obj)
                    #selected_obj = None
                    break
            # Check if an object in the priority queue was clicked
            for obj in priority_queue.slots:
                if obj.check_for_input(GAMEBUTTON_MOUSE_POS):
                    main_queue.add(obj)
                    priority_queue.remove(obj)
                    break
                if obj and priority_queue.add(obj):
                    main_queue.remove(obj)
                elif obj and main_queue.add(obj):
                    priority_queue.remove(obj)
```

When I run this prototype, I get an error message stating 'IndexError: Index out of range', this may be because the cards list may be found as empty before it is accessed. Therefore, moving it to be inside of the CardDistribution function may solve this problem. Also, due to the nature of the code above, I ran into a few errors:

- The background kept being placed over the buttons
    - Fix: place the screen fill function before everything else
- The card distribution kept looping itself, rearranging everything the moment it was generated
    - Fix: place behind a cards_done boolean variable, so that it is automatically false, then set true when complete.

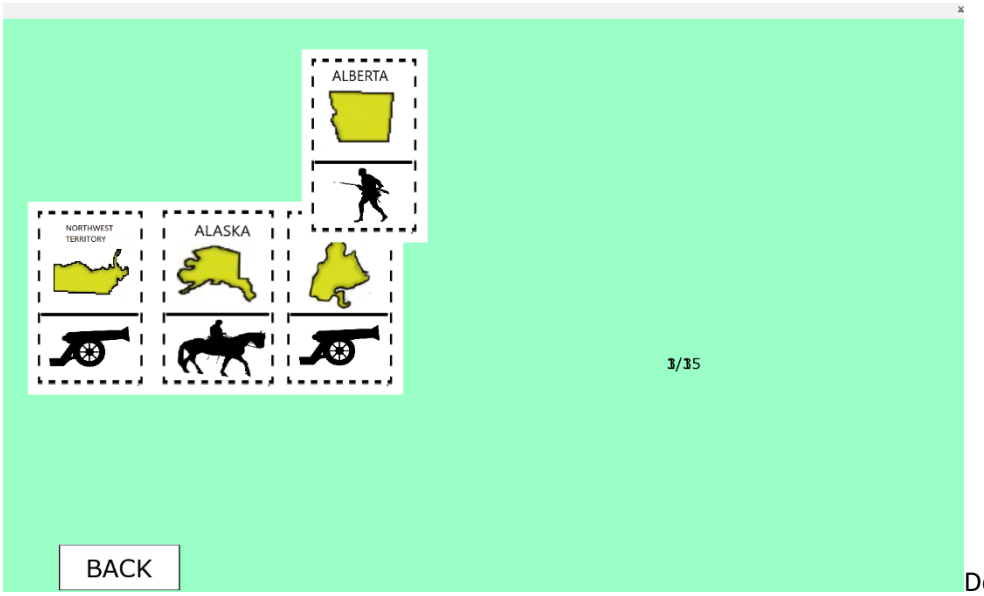
Output for card mechanics:

Upon first load:



As you can see, it picked 4 cards from the list of 6, and the font to the right of this is a counter for the capacity.

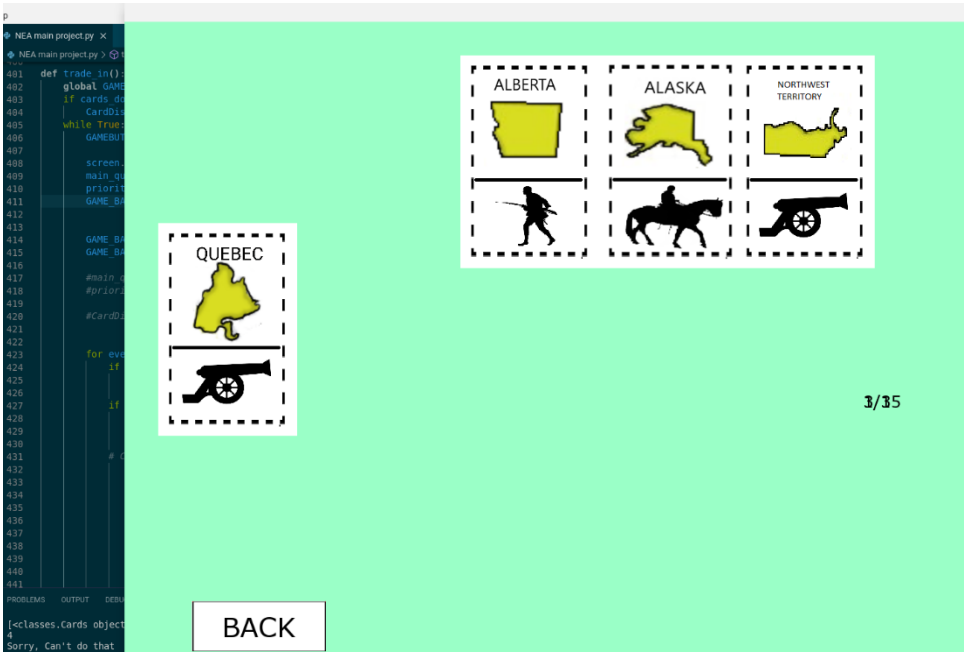
Clicking an object (in this example, Alberta:)



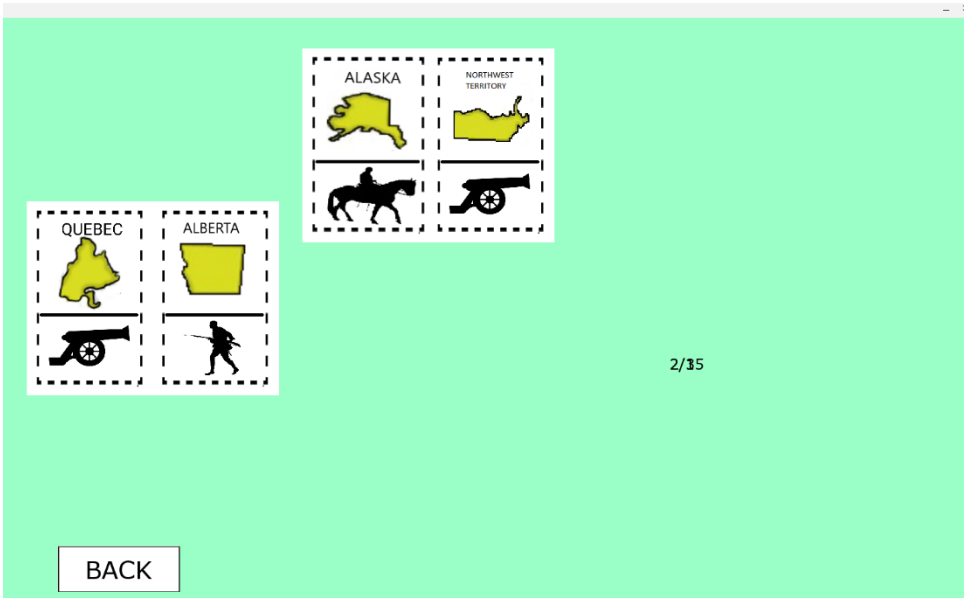
Despite the overlay issue, this can be easily changed by adjusting the y co-ordinates in the class. You can also see here that the font has updated, and everything to the right of alberta has been moved to the left.

Checking Capacity:

To check if capacity has been reached on the priority queue, I have included a print message to say that once the slots have reached 3 in the priority queue:



Checking deselection (in this case, Alberta, leaving 2 in the priority and 2 in the main queue:)



So now that the fundamental mechanics of this system are complete, I am now going to add the validation, as the above screenshots have demonstrated that the user is able to transfer any object from any list to the other location, regardless of order in the list.

Filtering out the selection

In order to setup a validation system so that only cards with the same type can pass through, the following will need to be met, since this will prevent different types from being added to the queue at any one time, it will only involve checking objects clicked from the main queue:

- If the length of the priority queue is between 1 and 3, check if the object type matches
- If the priority queue's type is set to none, set it to the object type
- If the priority queue is empty, set the priority queue's type to none
- If the priority queue is full, remove all its contents and add 10 troops to the deploy_troop variable

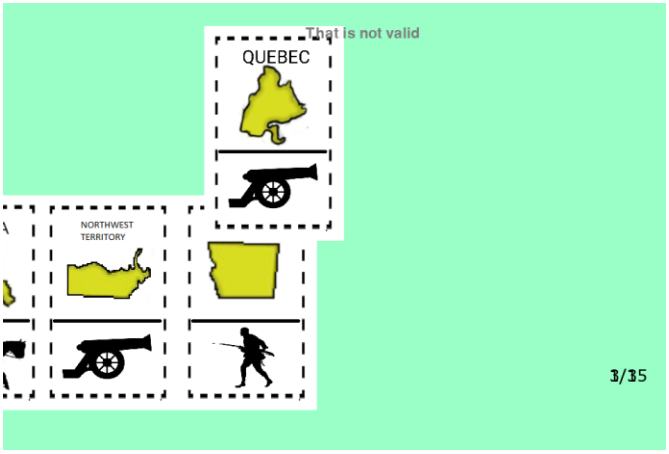
```
            break
    if clicked_obj:
        if clicked_obj in main_queue.slots:
            if len (priority_queue.slots)!=3 and len(priority_queue.slots)>=1: #if the length ranges from 1-3..
                if clicked_obj.type == first_obj.type:#if the type is the same
                    #deploy_troops += 10
                    main_queue.remove(clicked_obj) #remove from the main queue
                    priority_queue.add(clicked_obj)#add to priority
                #elif priority_queue.type == None: #if there is no set type
                #   priority_queue.type = clicked_obj.type #change the type to whatever is the object's type
                #   priority_queue.add(clicked_obj)
                #   main_queue.remove(clicked_obj)
                else:
                    feedback = True
                    #return False
            #this is the setup statement, as it always starts with 0 in.
            elif len(priority_queue.slots) == 0:#if there is nothing in the priority queue, reset
                priority_queue.type = None
                first_obj = clicked_obj
                main_queue.remove(clicked_obj)
                priority_queue.add(clicked_obj)
                priority_queue.type = first_obj.type
            else:
                print("queue is full") #this works
            #if len(priority_queue.slots)==3:
            #    print("Sorry, Can't do that")
            #else:
            #   main_queue.remove(clicked_obj)
            #   priority_queue.add(clicked_obj)
        elif clicked_obj in priority_queue.slots:#can remove anything from priority queue
            priority_queue.remove(clicked_obj)
            main_queue.add(clicked_obj)
```

As you can see from this screenshot, there is an excessive number of failed and commented out lines as I tested the validation system to little effect until this point, it did, however, occur to me that the first instance that I should be looking for is the condition for when the priority queue is 0, as if it is not able to add or remove anything then there will be no starting point for the list. From this, it is essentially the same process if the process is met, if not, then I have added the variable 'feedback', so

that I may blit text onto the screen, telling the user that the combination is invalid. The first_obj variable is also used to set the benchmark, as anything can be the clicked_obj, so this prevents overriding the type system.

```
global GAMEBUTTON_MOUSE_POS, main_queue,feedback
if cards_done == False:
    CardDistribution()
while True:
    GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()

    screen.fill (BISQUE)
    main_queue.draw(screen)
    priority_queue.draw(screen)
    GAME_BACK = GameButtonClass(game_button, 200, 900, "BACK")#in format image, y,x,text


    GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
    GAME_BACK.update()

    #main_queue = Queue(num_slots_main, queue_position) #defines the queue with 15 slots, and their positions
    #priority_queue = Queue(num_slots_priority, priority_queue_position) #3 slots and their location

    #CardDistribution()
    if feedback == True:#if something not valid triggered...
        response_rect = pygame.Rect((650, 50), (1, 1))
        response_font = pygame.font.SysFont('Sans Serif',32)
        response_text = response_font.render("That is not valid",True,(125,125,125))#create the text surface
        #print("not valid") #if the type's don't match then not valid
        screen.blit(response_text,response_rect)#put onto the screen
        time.sleep(3)#after 3 seconds have passed...
        if len(priority_queue.slots) == 0:#if the queue is reset
            feedback = False#message disappears
        else:
            feedback = False #message goes away
    else:
        feedback = False
```

Result: The text goes away after 3 seconds of screentime, and you can also place cards of the same type.

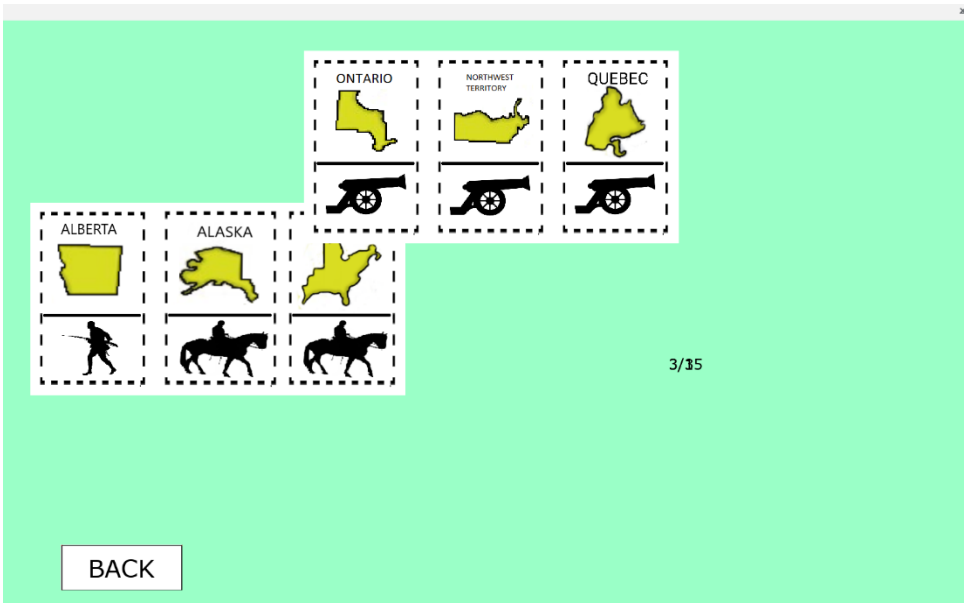


Finalising the card selection: If 3/3 of the same type are in the priority queue

```
            clicked_obj = obj
            break
    if clicked_obj:
        if clicked_obj in main_queue.slots:
            if len (priority_queue.slots)!=3 and len(priority_queue.slots)>=1: #if the length ranges from
                if clicked_obj.type == first_obj.type:#if the type is the same
                    main_queue.remove(clicked_obj) #remove from the main queue
                    priority_queue.add(clicked_obj)#add to priority
                else:
                    feedback = True
            #this is the setup statement, as it always starts with 0 in.
            elif len(priority_queue.slots) == 0:#if there is nothing in the priority queue, reset
                priority_queue.type = None
                first_obj = clicked_obj#starts off the priority list type
                main_queue.remove(clicked_obj)
                priority_queue.add(clicked_obj)
                priority_queue.type = first_obj.type
            elif len(priority_queue.slots) == 3:#if it has reached capacity
                priority_queue.type = None
                for obj in priority_queue.slots:#all the objects in the priority queue...
                    priority_queue.remove(obj)#are individually deleted...
                    available_troops = available_troops + 3#and for each card, the player gets 3 troops
                    cards.append(obj)#these cards are then added back to the list for later randomisation
        elif clicked_obj in priority_queue.slots:#can remove anything from priority queue
            priority_queue.remove(clicked_obj)
            main_queue.add(clicked_obj)
```

This should check to see if capacity is reached, if so, the troopcount will update and the cards expended will be returned to its original list, with the user gaining additional troops. This will be tested with a randomiser of 6, to ensure the user gets at least one set.

Result:



In checking to see why this has errored, I have realised that although the length is checked to see if it is full, it is only done so under the condition that something is being clicked, and it happens to be another card of the same type. This issue can be resolved through moving that specific piece of code towards the top of the if statements, so that it doesn't require any user input in order to finish.

Change:

```
#main_queue = Queue(num_slots_main, queue_position) #defines the queue with 15 slots, and their p
#priority_queue = Queue(num_slots_priority, priority_queue_position) #3 slots and their location

#CardDistribution()

if feedback == True:#if something not valid triggered...
    response_rect = pygame.Rect((650, 50), (1, 1))
    response_font = pygame.font.SysFont('Sans Serif',32)
    response_text = response_font.render("That is not valid",True,(125,125,125))#create the text
    #print("not valid") #if the type's don't match then not valid
    screen.blit(response_text,response_rect)#put onto the screen
    time.sleep(1)
    if len(priority_queue.slots) == 0:#if the queue is reset
        feedback = False#message disappearsy
else:
    feedback = False

if len(priority_queue.slots) == 3:#if it has reached capacity
    priority_queue.type = None
    for obj in priority_queue.slots:#all the objects in the priority queue...
        priority_queue.remove(obj)#are individually deleted...
        available_troops = available_troops + 3#and for each card, the player gets 3 troops
        cards.append(obj)#these cards are then added back to the list for later randomisation

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    if event.type == pygame.MOUSEBUTTONDOWN:
        if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
            Game()
        #GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()
```

The statement has been moved to the main while loop, and although this now works, I received the following terminal response:

```
  File "/home/_abihinch/Visual Studio/NEA & NEA practice/NEA Practice/Actual NEA code/NEA main project.py", line 435, in trade_in
    available_troops = available_troops + 3#and for each card, the player gets 3 troops
UnboundLocalError: local variable 'available_troops' referenced before assignment
```

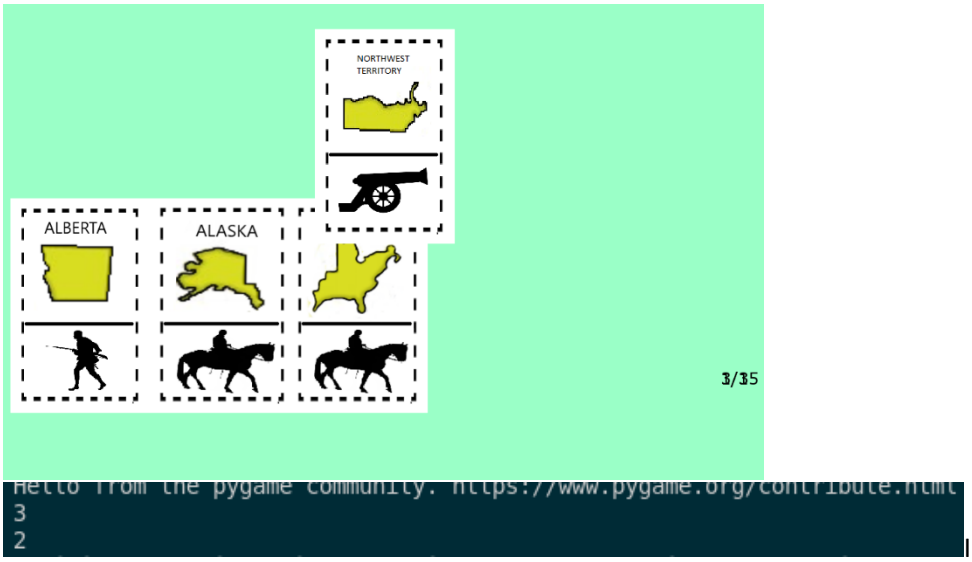
A suspected reason for this could be due to the fact that once an object gets removed from that queue, it is no longer 3, and therefore the rest wouldn't take place. Perhaps a while loop could fix this to ensure that it goes down 3 times.

After a series of debugging and rearranging my phase mechanics, I found a fix to this issue:

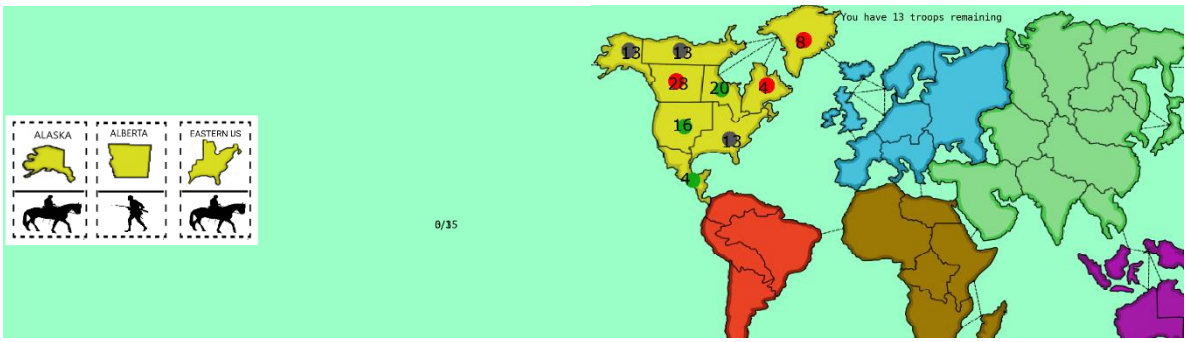
```
def DeployTroops():
    global default, NorthAmericaBonus, available_troops,deployphase,attackphase,attack_timer_active,card_handin,troop_done
    #for territories in NAterritories:
    if troop_done == False: #if it not unactivated
        if card_handin == False: #and there is no additionals
            if  p1_territories in NAterritories:#if the 1st player's territories cannot be found in the NA list as a whole
                #if generate == True:
                available_troops = default + NorthAmericaBonus#make the total 8
                troop_done = True#make done
                return available_troops
                #print(available_troops)
            else:
                NorthAmericaBonus = 0
                available_troops = default + NorthAmericaBonus#make total 3
                troop_done = True
                return available_troops
        else:
            available_troops = available_troops + 10 #add 10 to the total once.
            troop_done = True
            return available_troops
available_troops = DeployTroops()
```

By setting yet another boolean, it is able to limit the while loops continuous iterations, whilst also still checking. Thus, when the cards are handed in, in its current state, the available troops should be 13.

Another issue is that one of the cards are stil remaining, despite the troopcount going up, so if i run a print statement on the next test, i can attempt to pinpoint where the issue arises.



Interestingly, the length recorded does appear to stop at 2, perhaps indicating that although the list initially starts with 3, as the object is being removed during iteration, it may cause the program to terminate earlier. The fact it stops at 2 shows that when attempting to iterate over the 3rd object, the loop is unable to find it. I will make a copy of this with the .copy() module and hopefully this will help with resolving this problem, as the items will not be removed from that list whilst it iterates.



I can now conclusively say that that is the conclusion to the card mechanics, as other than introducing more cards, there is little more to add for this mechanic.

Final part of the deploy phase: troop deployment

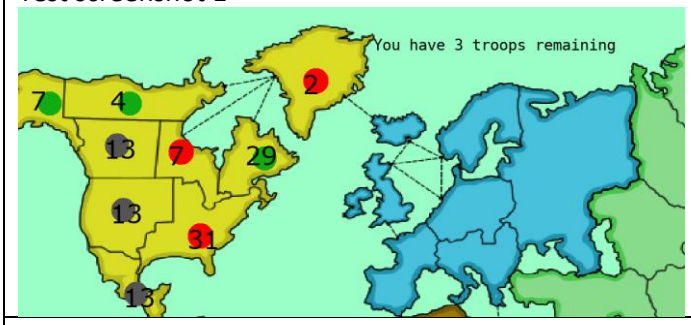
Seeing as I have already coded the territory selection mechanics, it would be useful now to combine the mechanics developed in this phase with the previous tests, so that the deploy phase may be completed, the aim is so that when

a territory is selected, and it belongs to the ally, the troops remaining goes down, and the territory's self.troopval goes up in 1 troop increments.

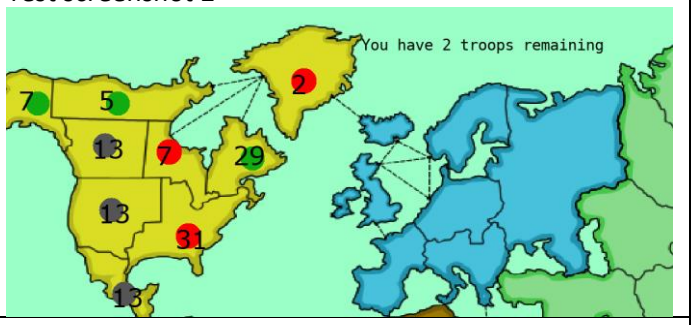
```
        #else:
            for territory in territories:
                if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #if the territory is clicked by the mouse
                    chosen_list.append(territory) #append it to a new list
                    for territory in chosen_list: #for that one territory
                        #if deployphase == True:#if the deploy phase is active..
                        if territory in pl_territories: #validates which territory the player can add to
                            territory.troopval += 1
                            available_troops -= 1
                            current_troopval = territory.troopval #logs the troopvalue
                            current_name = territory.name #logs the name
                            print(current_troopval, current_name)
                            territory.update_troop_text()
                            chosen_list.remove(territory)
                            #print(len(chosen_list))
                            #screen.blit(territory.trooptext,(territory.x_pos,territory.y_pos))
                        else:
                            print("You cannot access this at this time")
                    #screen.blit(territory.trooptext,(territory.x_pos,territory.y_pos))
```

In this screenshot, you can see that the details of the territory is logged when selected by the mouse, with the count edited down by 1 and 1 added onto the selected one, to prevent exponential increase of troops, the chosen item is removed after this has been completed, and the update troop text method is called, so that the changes will show on screen as it is done.

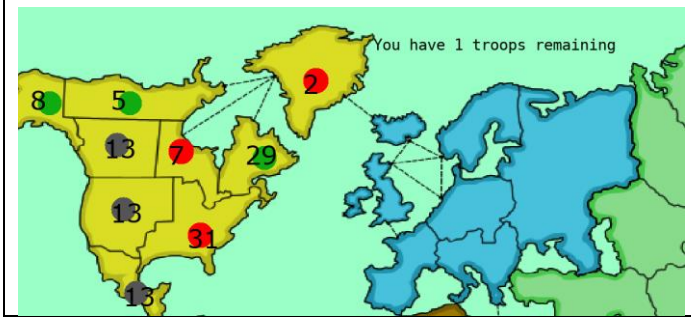
Test screenshot 1



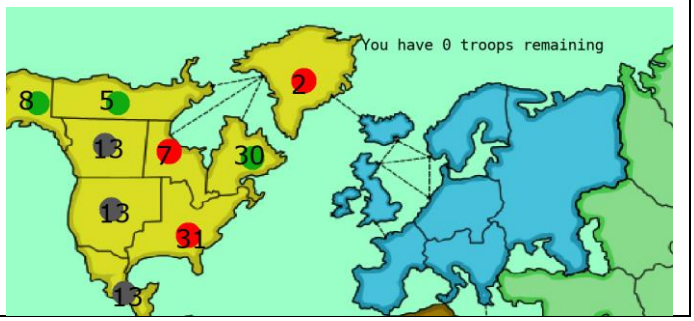
Test screenshot 2



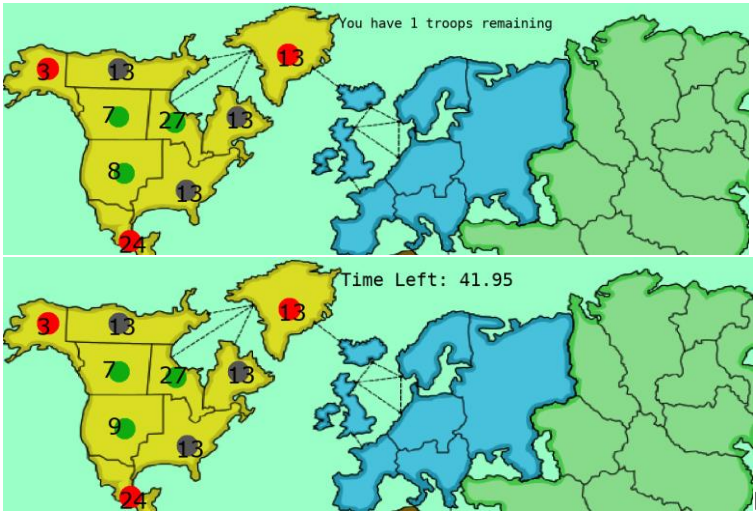
Test screenshot 3



Test screenshot 4



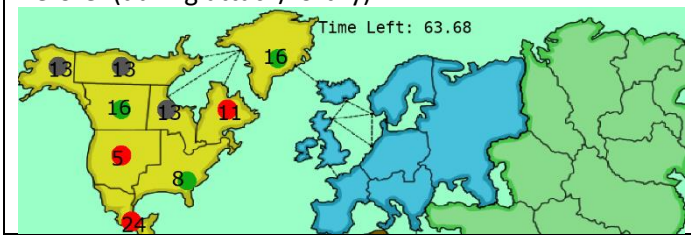
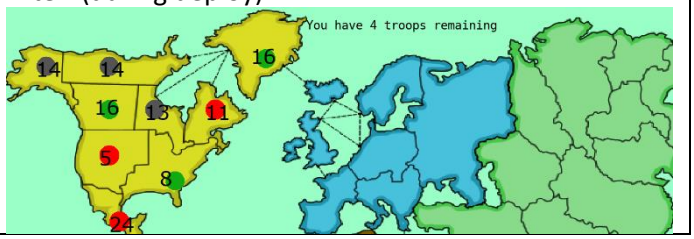
Seeing as I have not added any validation yet for the troop system, I will ensure that once the troopcount is reduced to 0, the attack phase will begin. To ensure that it will not continue to place down the troops beyond its capacity, even if the count is on 0, I have added an 'and' statement in the collidepoint territory statement, so that it will only work if there is not 0, with the terminal response saying that it 'cannot be accessed at this time'.



Before: 1 troop remaining, after: 1 troop added, and automatically begins next phase.

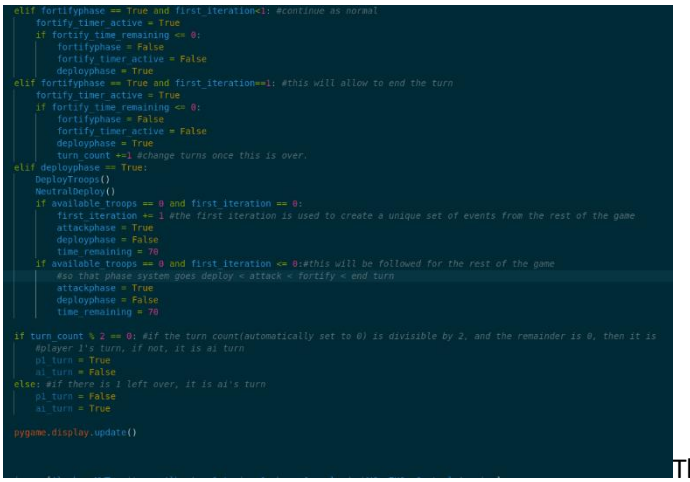
```
def NeutralDeploy(): #adds half of what the player recieves, randomly adding it
    global available_troops, n_done
    neutral_available_troops = available_troops//2  #has half val, available currently set 4 for default
    for territory in neutral_territories: #if neutral
        if neutral_available_troops != 0 and n_done == False: #when not 0 or finished
            random_territory = random.choice(neutral_territories) #select..
            chosen_list.append(random_territory)
            territory.troopval += 1#add 1 to the selected
            neutral_available_troops -= 1#remove from total
            territory.update_troop_text()#update the text
            chosen_list.remove(random_territory) #prevents constant endless iteration
        if neutral_available_troops == 0:#prevents endless iteration
            n_done = True
```

In line with the deploy phase, the neutral territory also gains half of what the player does, so this is very similar to the player's troop mechanics, but instead randomised.

Before: (during attack/fortify)	After: (during deploy)
	

The final part of this stage will be to add the turn based settings before I complete the rest of the mechanics, as it will allow for the AI setup.

For now, this and a 'first iteration' variable has been created, to ensure the game follows a unique set of events before commencing the usual deploy < attack < fortify < next player's turn



```
elif fortifyphase == True and first_iteration<1: #continue as normal
    fortify_timer_active = True
    if fortify_time_remaining <= 0:
        fortifyphase = False
        fortify_timer_active = False
        deployphase = True
elif fortifyphase == True and first_iteration==1: #this will allow to end the turn
    fortify_timer_active = True
    if fortify_time_remaining <= 0:
        fortifyphase = False
        fortify_timer_active = False
        deployphase = True
        turn_count +=1 #change turns once this is over.
elif deployphase == True:
    DeployTroops()
    NeutralDeploy()
    if available_troops == 0 and first_iteration == 0:
        first_iteration += 1 #the first iteration is used to create a unique set of events from the rest of the game
        attackphase = True
        deployphase = False
        time_remaining = 70
    if available_troops == 0 and first_iteration <= 0:#this will be followed for the rest of the game
        #so that phase system goes deploy < attack < fortify < end turn
        attackphase = True
        deployphase = False
        time_remaining = 70

if turn_count % 2 == 0: #if the turn count(automatically set to 0) is divisible by 2, and the remainder is 0, then it is
    #player 1's turn, if not, it is ai turn
    pl_turn = True
    ai_turn = False
else: #if there is 1 left over, it is ai's turn
    pl_turn = False
    ai_turn = True

pygame.display.update()
```

This is the updated set of parameters, to include the turns. The turns will be added, with duplicate versions of this event loop being sent out, with slight changes depending on who is playing. Currently, all the phases work as intended, yet the rules of the attack and fortify mechanics still need to be coded in.

Review of Stage 1

What has been done so far

So far in my program, when executing the file, the user interface appears, giving the user the option to start up the game, go into their settings or quit the game, with each additional menu having the option to revert back to the previous one. Upon starting the program, a timer is started, counting down from 70 as the game launches itself first into the attack phase, having set up the territory and troop distribution per player. Each player has at least 1 troop per territory, and so far the territories are evenly split between players. When in the attack phase, the player can choose to skip and advance to the next phase if they wish to do so, with the use of a 'end turn' button. In this case this will take them to the fortify phase, where the timer is currently set to 30 as per stakeholder feedback from analysis, although these two phases currently do nothing except for count down.

When you go onto the deploy phase, the player is given at least 4 troops, however, this can be subjected to increase given a set of requirements which are met. During this phase, the player has access to a new mechanic which could previously not be accessed: Card mechanics. Upon clicking this button, (although I need to change how many the player gets per turn), this brings up a menu, displaying the cards visually in a [] [] [] [] [] format, with 3 rows of 5 cards (although I only currently have 6 created in total), the cards themselves are automatically placed into a default queue, but, if clicked, they are transferred to a priority queue. From this, the card types are validated so that only the same type can be added, once all 3 of the same type are added, they are removed from the player's cards and added to the troopcount. The card type of the priority queue can also be changed if all cards inside of it are removed.

A visual display of how many available troops the player has is also shown on screen, and if the player clicks on territories that are their own, the screen will update by increments of 1 on the troop value per click, removing 1 from the total of available troops. Once the troops are expended, it automatically launches back into the attack phase, and subsequently the fortify phase, where, once completed, that turn has ended.

How it was tested

With each process that could be physically represented onto the screen (as a significant proportion of events happened to run in the background), this was tested through the user interface, where often the interacted elements would be updated when the requirements were met. For aspects such as the deployment of troops and the card mechanics, this involved collision with specific objects in order to determine an output, and this helped to highlight issues such as exponential deployment, minus values in troops and random duplication of cards.

For background processes, the use of the terminal and print statements helped to keep track of events, this came in especially useful for aspects such as the setting up of the collision detection system, and the deployment territory validation, as well as values such as the length of lists (especially useful when working with random values and such) and checking to see if processes are needlessly iterating (ie if I had accidentally misplaced a process inside a while loop, or mistakenly defined an important integer variable in a while loop, so that it would result in never changing).

How it meets success criteria

In terms of user requirements, this current version of my program is able to set up my game, with most of the foundations laid for future development, the criteria is listed in the section below, features such as my clock meets my success criteria as the values are based off of responses from my users, other features such as my card system add a level of dynamicity to my solution.

Criteria met/ being met

Fully met:

- **An on-screen clock should be displayed on screen**
- **Territories must be divided equally, randomly and fairly.**
- **I would like to add a button that can skip phases and end turns prematurely.**
- **A simple menu should be used to link together features of my project**
- **The game can be quit at any given time, with the use of a 'Quit' button**
- **A card system should be implemented into the game**

Changes in design from original plans

The timer mechanism as a function has proven to become problematic to the development of my code, thus this has had to be replaced by a system that runs under the requirements of an 'if' statement, being continually called until a boolean value is completed as per the events running.

Card mechanics format: the original plan was to create a grid system, however, due to logistics, it became easier to implement a system that worked off of initial x and y values, and for each card added it would add it a set value either x or y of the original value, additionally, if anything inside this co-ordinate was interacted with, it would get added to a separate form of 'queue', if any objects were added or removed from a certain queue, the order would switch up to ensure that all the gaps between the cards were filled, and considering this, they could be selected regardless of order.

Overall summary

At the moment, the user interface sets up the start of the territories and troops, with the user able to add troops, a timer that can count down from varying values, and a card mechanic which adds values to the troopcount. No usability features have been implemented just yet, however, during stage 2, there are plans to do this after completing the attack phase mechanics, as this will involve features like changing the allegiance of a territory once it has been conquered, and thus the image will need to be changed to the colourblind mode dependant on what is selected, as well as this, a rules menu and login system will be added after the rules are fully implemented.

Stage 2

Test 11 & 12: Attack phase & Dice Mechanics

In creating the mechanics for the territory selection process for attacking, I created two separate lists, going by the same method used for the deploy phase, however, its overall implementation is different, as this adds comparisons and the neighbouring mechanics are used for validation upon attack, to keep a track of this, I will setup the notifications box, so that the messages ie "[territory a] is selected, now selected an enemy/ neutral territory."

```
                    print("You cannot access this at this time")
            if attackphase == True: #if the attack phase is active
                #logging territory a
                for territory in p1_territories:#if the territory is player 1's
                    if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and gets clicked....
                        if len(territory_a) != 1: #if the length of the territory is not 1
                            territory_a.append(territory) #add it to the list
                            for territory in territory_a: #for the listed territory
                                territory_a_name = territory.name #log the name (for testing purposes)
                                territorya = territory #logs the territory (for comparison values)
                                print(territory_a_name)
                        else:
                            territory_a.remove(territorya) #if it gets to larger than 1, deselect
                for territory in enemy_territories or neutral_territories: #if the territory is not te player's
                    #logging territry b
                    if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and it gets clicked
                        if len(territory_b) != 1: #check if empty
                            territory_b.append(territory) #add to the list
                            for territory in territory_b:
                                territory_b_name = territory.name #log the name (for test purposes)
                                territoryb = territory #logs the name
                                print(territory_b_name)
                        else:
                            territory_b.remove(territoryb)#if it is 1 then remove it - so deselects
                pass
```

In testing to see if I can select and deselect a territory for attacking and defending during the attack phase, I will first run a set of tests, and start implementing a method called is_neighbour() and add_neighbour() under the territory class, as well as a changeAllegience() (for future use when conquering other territories).

```
def update_troop_text(self):
    self.trooptext = menu_font.render(str(self.troopval),True,"Black")

def update(self):
    screen.blit(self.image,self.rect)
    screen.blit(self.text, self.text_rect)

def checkForInput(self, position):
    if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
        return True

def HighlightByMouseHover(self, position):
    if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
        self.trooptext = menu_font.render(self.troopval, True, RED) #allows the user to know whether the cursor is by the text
    else:
        self.text = menu_font.render(self.troopval, True, "black")#resets back to black

def ChangeAllegience(self):
    if self.image == 'ally.png' and self.troopval <= 0: #if this is the image it occupies but the troops have been expended
        self.image = 'enemy.png' #change to enemy
        self.troopval += 1 #then add 1 so that it doesn't loop in diff colours
    elif self.image == 'neutral.png' and self.troopval <= 0:
        if p1turn == True:
            self.image = 'ally.png'
            self.troopval += 1
        elif not p1turn:
            self.image = 'enemy.png'
            self.troopval += 1

def add_neighbour(self,territory):
    self.Neighbours.append(territory)

def is_neighbour(self,territory):
    return territory in self.Neighbours
```

<-- Territory class methods (updated)

```
#neighbouring defining - North America
Alaska.add_neighbour(NWTerritory)
Alaska.add_neighbour(Alberta)
NWTerritory.add_neighbour(Alaska)
NWTerritory.add_neighbour(Alberta)
NWTerritory.add_neighbour(Greenland)
NWTerritory.add_neighbour(Ontario)
Greenland.add_neighbour(NWTerritory)
Greenland.add_neighbour(Ontario)
Greenland.add_neighbour(Quebec)
Alberta.add_neighbour(Alaska)
Alberta.add_neighbour(NWTerritory)
Alberta.add_neighbour(Ontario)
Alberta.add_neighbour(WUS)
Ontario.add_neighbour(NWTerritory)
Ontario.add_neighbour(Greenland)
Ontario.add_neighbour(Alberta)
Ontario.add_neighbour(Quebec)
Ontario.add_neighbour(EUS)
Ontario.add_neighbour(WUS)
Quebec.add_neighbour(Greenland)
Quebec.add_neighbour(Ontario)
Quebec.add_neighbour(EUS)
WUS.add_neighbour(Alberta)
WUS.add_neighbour(Ontario)
WUS.add_neighbour(EUS)
WUS.add_neighbour(Central_America)
EUS.add_neighbour(Quebec)
EUS.add_neighbour(Ontario)
EUS.add_neighbour(WUS)
EUS.add_neighbour(Central_America)
Central_America.add_neighbour(EUS)
Central_America.add_neighbour(WUS)
```

<--- Defining the neighbours

To also update the detection system to validate not only the allegiance, but also the neighbouring status and troopvalue validation for attack, I will run a few tests on the terminal.

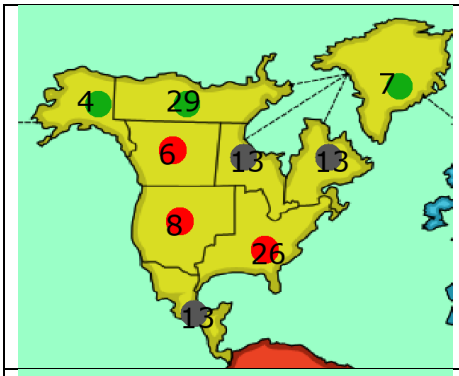
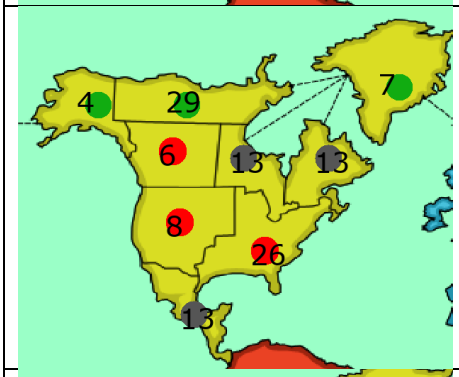
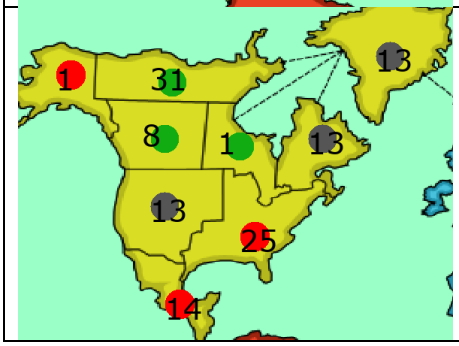
```
    #logging territory a
    for territory in p1_territories:#if the territory is player 1's
        if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and gets clicked....
            if len(territory_a) != 1: #if the length of the territory is not 1
                territory_a.append(territory) #add it to the list
                for territory in territory_a: #for the listed territory
                    territory_a_name = territory.name #log the name (for testing purposes)
                    territorya = territory #logs the territory (for comparison values)
                    territoryatroops = territory.troopval #logs the troop value
                    if territoryatroops >= 2:
                        print("attacker selected:",(territory_a_name))
                        selection = True
                    else:
                        print("sorry",(territory_a_name),"does not have enough troops, please select another")
                        territory_a.remove(territorya)
                        selection = False
            else:
                print("attacker deselected:",(territory_a_name))
                territory_a.remove(territorya) #if it gets to larger than 1, deselect
                selection = False
    for territory in enemy_territories or neutral_territories: #if the territory is not te player's
        #logging territry b
        if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and it gets clicked
            if len(territory_b) != 1: #check if empty
                territory_b.append(territory) #add to the list
                for territory in territory_b:
                    territory_b_name = territory.name #log the name (for test purposes)
                    territoryb = territory #logs the name
                    print("defender selected:",(territory_b_name))
                    second_selection = True
            else:
                print("defender deselected:",(territory_b_name))
                territory_b.remove(territoryb)#if it is 1 then remove it - so deselects
                second_selection = False
    if selection == True and second_selection == True:
        if territorya.is_neighbour(territoryb): #if the territory a is neighbours with the selected enemy territory..
            print("they are neighbours")
        else:
            print("they aren't neighbours")
```

To explain this screenshot, inside of the events loop in the main game function, it checks to see if a player 1 territory has been selected, and thus it checks to see if the list of the logged territory is either empty or already containing an object, in the event of this

already containing an object, the previously added object is removed from the list, and selection is set to False, so that the program does not error on an incomplete comparison. For now, it logs the name in order to output to the terminal to test if the selection is detected. In addition to this, the troopvalue is logged for the later comparisons, as well as to validate if the territory has sufficient available troops to use up in the attack (only applies to the attacker). A similar process is done if the territory selected to be attacked is not neighbouring the player, setting the selection boolean value to true or false where appropriate.

If the validated territories are not neighbouring, then it will display if they are/ aren't.

Testing (All Validation – selection, neighbours and troopvalue):

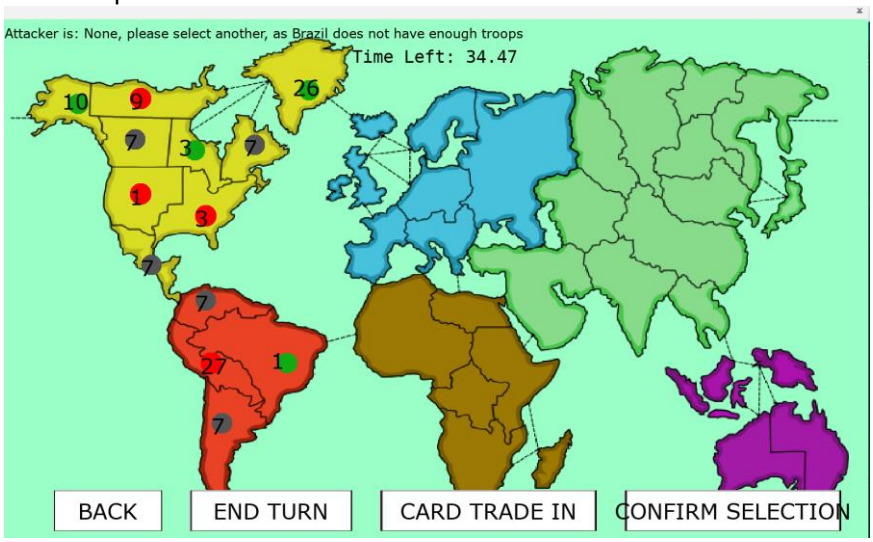
	If we wanted to select Alaska: (as attacker) `attacker selected: Alaska` And deselect Alaska: `attacker deselected: Alaska` Instead selecting North West territory `attacker selected: NWTerritory`	If we wanted to select the defender (Alberta): `defender selected: Alberta`
	If we wanted to check if the selected territories, Alaska and NorthWest Territory are neighbours: `attacker selected: NWTerritory` `defender selected: Alberta` `they are neighbours`	If we wanted to deselect a territory to ie Alberta, and select a territory that doesn't neighbour, for example, Western US: `defender deselected: Alberta` `defender selected: WUS` `they aren't neighbours`
	If we clicked on Ontario to attempt to attack: `sorry Ontario does not have enoug`	

As can be seen from this, the validation is recognised of these territories, therefore, I can move onto the implementation of my notifications box, (AKA text that wraps inside its predefined measurements, displaying which territories have been clicked, as well as any error messages) so that it can output to the user. Following this, I will start up the dice mechanics to be used, and add the next continent: South America, for the start of the player territory allegiance turnover and the actual attack mechanisms which will affect how many troops will reside on which territory. It would be pointless to show the new code for the South America class, as this is essentially a repeat of the code used for the North America.

In order to blit the results onto the screen, there must first be a default text setting to avoid trying to call a variable that is nonexistant, therefore, I will create an attacker text, and have it say "Attacker is None", this can then be updated accordingly during selection, replacing the print statements. Once this has been done, a similar process will

be created for the defender. This is then blitted onto the screen inside of an if statement within the main loop, so that it may disappear after the attack phase has passed.

```
#logging territory a
for territory in p1_territories:#if the territory is player 1's
    if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and gets clicked....
        if len(territory_a) != 1: #if the length of the territory is not 1
            territory_a.append(territory) #add it to the list
            for territory in territory_a: #for the listed territory
                territory_a_name = territory.name #log the name (for testing purposes)
                territorya = territory #logs the territory (for comparison values)
                territoryatroops = territory.troopval #logs the troop value
                if territoryatroops >= 2:
                    #print("attacker selected:",(territory_a_name))
                    attacker_text = font.render(f"Attacker is: {territory_a_name}", True, (BLACK))
                    selection = True
                else:
                    #print("sorry",(territory_a_name),"does not have enough troops, please select another")
                    attacker_text = font.render(f"Attacker is: None, please select another, as {territory_a_name} does not have enough troops", True, (BLACK))
                    territory_a.remove(territorya)
                    selection = False
        else:
            #print("attacker deselected:",(territory_a_name))
            attacker_text = font.render("Attacker is: None", True, (BLACK))
            territory_a.remove(territorya) #if it gets to larger than 1, deselect
            selection = False
```

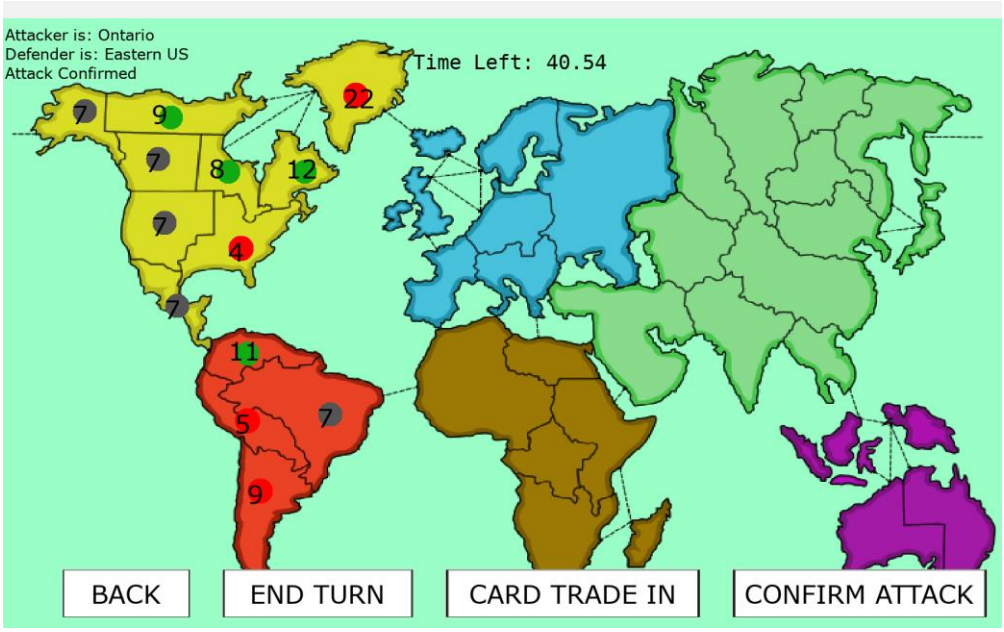
Test: Output if insufficient 	Although the formatting of this text isn't the most ideal, I have decided to leave this as is, as this will be the longest text output, which will not be repeated for the defenders, as there is no validation other than checking to see if is neighbouring.
Test: Selected Territory 	Works as intended, text is readable however screenshot looks misleading in terms of font size.

Test: Default/ deselected territory 	Works as intended, read above for conclusion of this.

Now I will implement the same for the defender, however this will not require tests, and I will move on to the dice mechanics. Button & display implementation.

Here are the final screenshots of the GUI & code before commencing with the dice mechanics (and territory handovers):

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:…
    if event.type == pygame.MOUSEBUTTONDOWN:
        if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):…
        elif END_TURN.checkForInput(GAMEBUTTON_MOUSE_POS):…
        if deployphase == True:…
        if attackphase == True:#if the attack phase in in progress
            if CONFIRM_SELECTION.checkForInput(GAMEBUTTON_MOUSE_POS): #if the confirm selection has been clicked
                if selection == True and second_selection == True: #as long as both values have been submitted
                    if territorya.is_neighbour(territoryb): #if the territory a is neighbours with the selected enemy territory...
                        #print("they are neighbours")
                        output = font.render("Attack Confirmed",True,(BLACK)) #change the text to attack confirmed
                        #diceroll()
                    else:
                        #print("they aren't neighbours")
                        output = font.render("Error: Territories aren't neighbours.",True,(BLACK)) #otherwise tell the user that the
                        #resets the territories selected
                        territory_a.remove(territorya)
                        territory_b.remove(territoryb)
                        #updates the text to notify the user
                        attacker_text = font.render("Attacker is: None", True, (BLACK))
                        defender_text = font.render("Defender is: None", True, (BLACK))
                        #prevents the cycle from erroring
                        selection = False
                        second_selection = False
            #   diceroll()
```



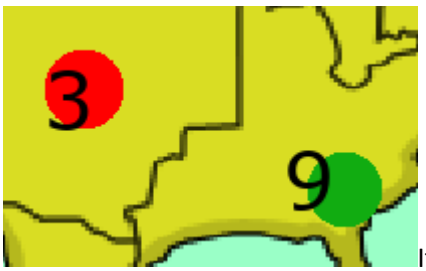
Dice mechanics

In order to best run the dice mechanics, now that I have my validation mechanisms in place, I can start a function that will determine how many dice each player is entitled to roll, and this can be done with the implementation of a attacker_dice_count and defender_dice_count variables.

```
def diceroll(): #function for the start of the troop decrements
    # Determine how many dice to roll for the attacker and defender
    if territorya.troopval == 2: #if the player has only 2 troops available
        attacker_dice_count = 1 #they only get one attack dice
    elif territorya.troopval == 3: #if there are 3 troops, give them 2 dice
        attacker_dice_count = 2
    else:
        attacker_dice_count = 3#otherwise they get 3 dice

    if territoryb.troopval == 1:#if the defender has only 1 troop, they are entitled to
        defender_dice_count = 1
    else:
        defender_dice_count = 2#whereas anything more than 1 allows them to have 2

    #picks a random value from 1 to 6, for every time each playr have a dice
    attacker_rolls = [random.randint(1, 6) for i in range(attacker_dice_count)]
    defender_rolls = [random.randint(1, 6) for j in range(defender_dice_count)]

    # Render the results as text
    attacker_dice = f"Attacker: {attacker_rolls}"
    defender_dice = f"Defender: {defender_rolls}"

    print(attacker_dice,defender_dice) #print statement used for testing purposes
```

By validating how many dice each player is entitled to, it makes sense to then integrate that number of rolls.

Test run 1: Does the dice register?



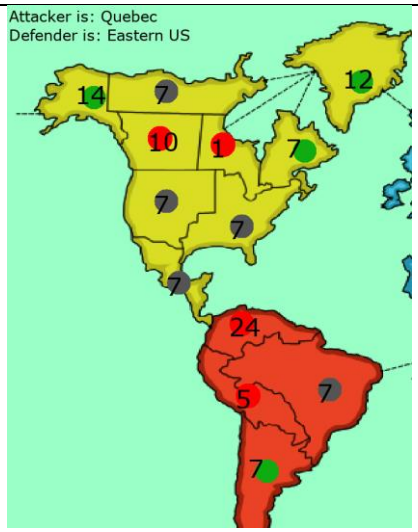
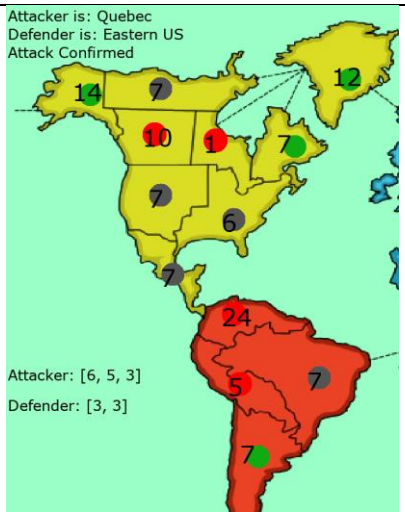
If these are the territories, it means that the player would have 3 dice to roll, and the defender would have 2.

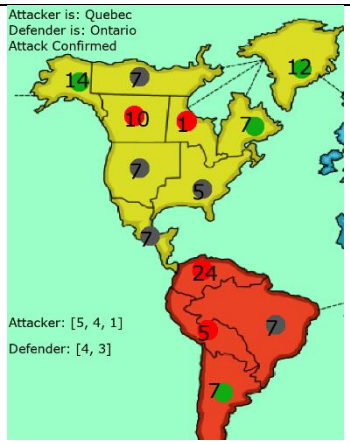
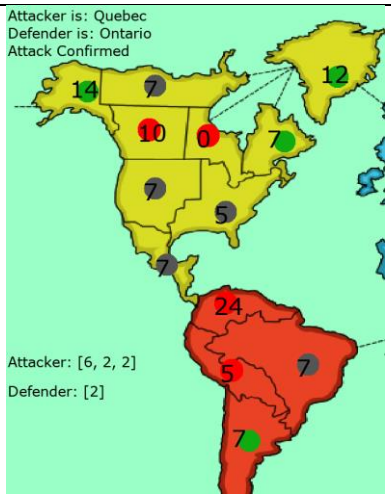
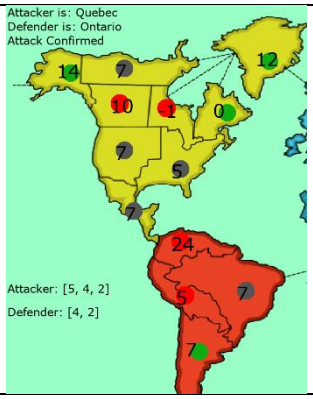
As shown here, this shows a successful response.

Now that it has been established that the dice will roll as appropriate, in order to integrate the comparison data, it would be logical to make use of the sorted() function in python to sort the numbers in descending order, therefore, this will help to pitch the highest values against one another, as it means I will be able to compare the 1st indexed element with each other in the separate rolls, so I can accordingly alter the troopvalues and update them. Alongside this, I will make the addition of blitting the dice rolls onto the screen in text form, so that the user is informed of what has been rolled.

```
global attacker_surface, defender_surface
# Determine how many dice to roll for the attacker and defender
if territorya.troopval == 2: #if the player has only 2 troops available
    attacker_dice_count = 1 #they only get one attack dice
elif territorya.troopval == 3: #if there are 3 troops, give them 2 dice
    attacker_dice_count = 2
else:
    attacker_dice_count = 3#otherwise they get 3 dice

if territoryb.troopval == 1:#if the defender has only 1 troop, they are entitled to 1 dice
    defender_dice_count = 1
else:
    defender_dice_count = 2#whereas anything more than 1 allows them to have 2

#picks a random value from 1 to 6, for every time each playr have a dice
attacker_rolls = sorted([random.randint(1, 6) for i in range(attacker_dice_count)],reverse=True)
#reverse = True ensures that it is sorted into descending order so that its first element is the highest
defender_rolls = sorted([random.randint(1, 6) for j in range(defender_dice_count)],reverse=True)

# Render the results as text
attacker_dice = f"Attacker: {attacker_rolls}"
defender_dice = f"Defender: {defender_rolls}"

#print(attacker_dice,defender_dice) #print statement used for testing purposes
attacker_surface = font.render(attacker_dice, True, (0, 0, 0))
defender_surface = font.render(defender_dice, True, (0, 0, 0))

if attacker_rolls[0]>defender_rolls[0]:#if the attacker rolls more than the defender
    territoryb.troopval -=1 #attacker wins, so defender loses a troop
    territoryb.update_troop_text()
elif attacker_rolls[0] == defender_rolls[0]: #if they are equal, only the attacker loses a troop
    territorya.troopval -= 1
    territorya.update_troop_text()
else:
    territorya.troopval -= 1 #if the defender is higher, attacker loses a troop
    territorya.update_troop_text()
```

Test table: Updating the troop values upon winning/ losing:

Test type	GUI - Before	Actual Result	Test write-up
Attack between two territories with more than 1 troop on the defender's side. (X vs X)			From this test, we can see that Eastern US, which had 7 troops originally, rolled with 2 dice, and the highest result, 3, was compared against the defender's highest result, 6, thus resulting in a –1 deduction to the overall troopcount for the defending territory. However, it is important to note that this should have been –2, but since we are currently only testing the first indexed value, this test has successfully performed

(X vs 1) no. of troops (see if it continues to attack if the enemy has no troops)			Test Failed (The text of the previous attack is still on the screen for the first screenshot – this is overlooked at the moment, the top text is the most relevant) As you can see, when the attacker rolled 6 and the defender rolled 2, it decremented the troopvalue by 1, but didn't change the state of occupancy.
0 vs x (see if it will decrement to minus figures without extra validation)	N/A		Test Failed. Despite having no troops to attack with, the dice was still rolled and the troops were deducted accordingly.

In light of these tests, the first issue to validate out would have to be the changing allegience of the territory, since the method is already created in the Territory class, it would be wise to include that in my assessment of whether the troopcount can decrement below 0.

```
if territorya.troopval == 1: #if the attacker only has 1 troop left, do not let them attack
    territory_a.remove(territorya) #thus remove it from the selected
    attacker_text = font.render("Attacker is: None", True, (BLACK)) #update the display to show this
elif territoryb.troopval == 0:#if the defender runs out of troops
    territoryb.ChangeAllegience() #change the allegience
    territoryb.troopval += 1 #ensures that there is never 0 on that territory - will later be changed on the input
    territoryb.update_troop_text()
    p1_territories.append(territoryb) #ensures that now the player owns this
    if territoryb in enemy_territories:
        enemy_territories.remove(territoryb) #ensures that the enemy no longer owns this
        territory_b.remove(territoryb) #deselect the territory so thar it can no longer be attacked by player
        defender_text = font.render("Defender is: None", True, (BLACK)) #update the text on the display
        territoryb = None
        second_selection = False#so that territories can be reselected
        territory_a.remove(territorya)
        attacker_text = font.render("Attacker is: None", True, (BLACK))
        territorya = None
        selection = False
    else:
        neutral_territories.remove(territoryb)
        territory_b.remove(territoryb) #deselect the territory so thar it can no longer be attacked by player
        defender_text = font.render("Defender is: None", True, (BLACK)) #update the text on the display
        territoryb = None
        second_selection = False
        territory_a.remove(territorya)
        attacker_text = font.render("Attacker is: None", True, (BLACK))
        territorya = None
        selection = False
```

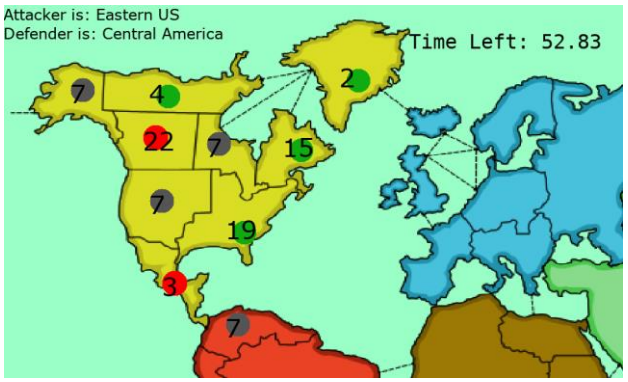
This is my method of validation to meet the following requirements:

1. All territories are deselected
2. The territory that has been defeated are switched to the other side accordingly
3. The image updates when 0 has been reached
4. 1 or an integer value is added to the troop value of that territory to ensure that it always exceeds a capacity of 1

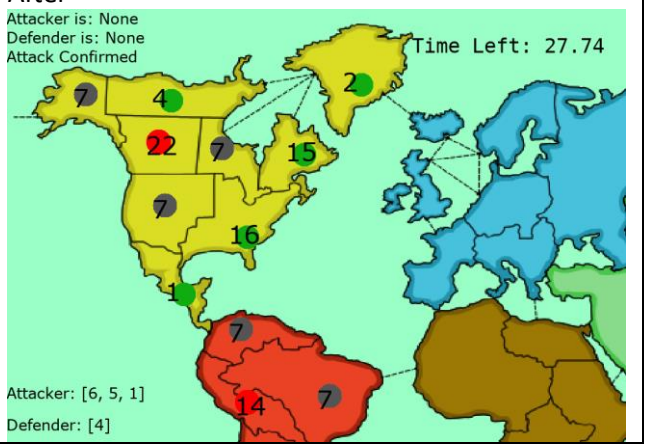
5. The text on screen is updated
6. The selection boolean values are set to false, so that the confirm selection button does not error when pressed.

Test for changing allegience

Before

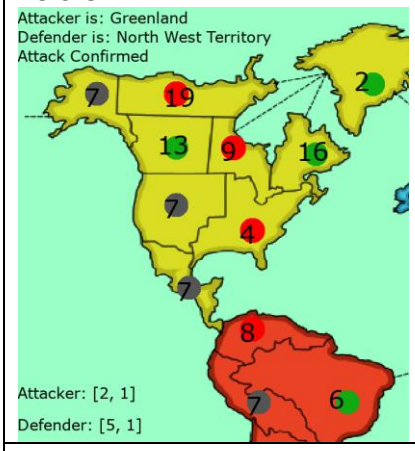
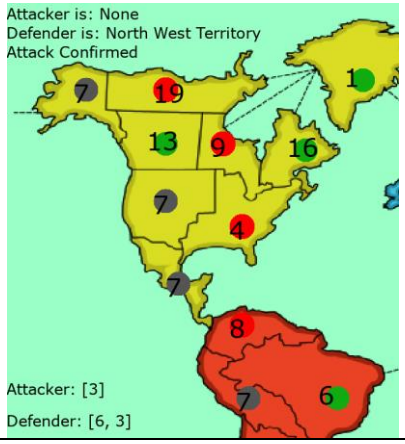
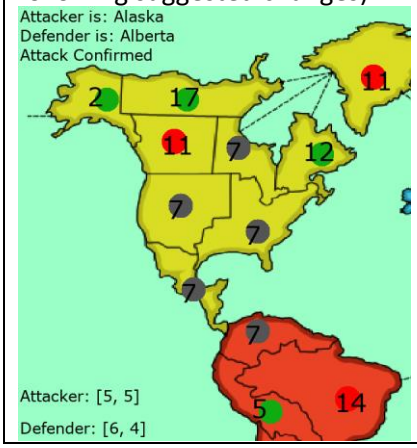


After



As shown above, this test was successful, as it changed the allegiance visually, and also met all the above requirements.

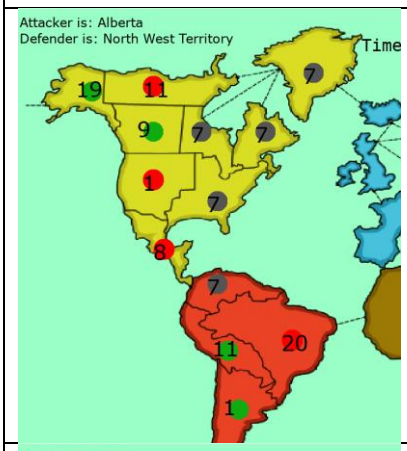
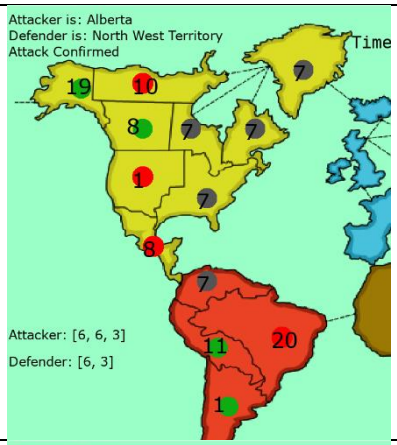
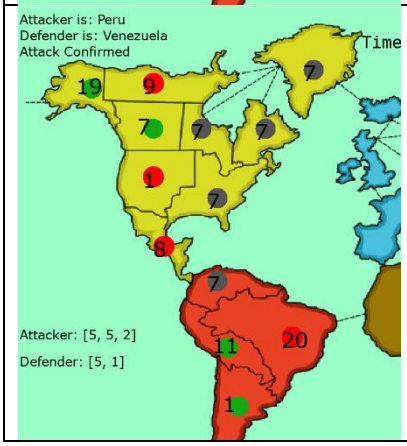
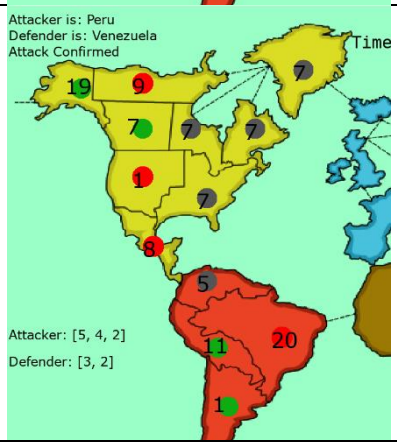
Test to see if the attacker is forced to halt their attack due to lack of troops:

Before 	After 	Test Write-up: Despite the attacker text updating, when I attempted to press confirm attack, it appears that the territory hadn't been properly deselected, as I got an error message involving my program finding difficulty finding that previous territory, therefore, to improve this, if I follow a similar approach to the allegiance changes, then it should resolve this issue.
Test amendment: (attempt 2 following suggested changes) 	After 	Test write-up: Even if I continued to press confirm selection, nothing happened, thus showing that this test is successful, and validation is fully implemented.

As shown here, nearly all attack mechanics are complete, with the only two remaining tasks left to do of this phase is create the second comparison for the dice roll & finalizing the transfer mechanics, so that the player can choose to move a certain number of troops after conquering the territory.

```
if attacker_rolls[0]>defender_rolls[0]:#if the attacker rolls more than the defender
    territoryb.troopval -=1 #attacker wins, so defender loses a troop
    territoryb.update_troop_text()
elif attacker_rolls[0] == defender_rolls[0]: #if they are equal, only the attacker loses a troop
    territorya.troopval -= 1
    territorya.update_troop_text()
else:
    territorya.troopval -= 1 #if the defender is higher, attacker loses a troop
    territorya.update_troop_text()

if attacker_dice_count >= 2 and defender_dice_count == 2:
    if attacker_rolls[1]>defender_rolls[1]:#if the attacker's 2nd highest is greater than defender's 2nd highest
        territoryb.troopval -=1 #attacker wins, so defender loses a troop
        territoryb.update_troop_text()
    elif attacker_rolls[1] == defender_rolls[1]: #if they are equal, only the attacker loses a troop
        territorya.troopval -= 1
        territorya.update_troop_text()
    else:
        territorya.troopval -= 1 #if the defender is higher, attacker loses a troop
        territorya.update_troop_text()
```

Test for both rolls:

Before	After	Observations
		With the values both being 11 and 9 respectively, when both the attacker and defender rolled a 6, only the attacker lost a troop, which follows the rules of the game. In the same iteration, it compared 6 against 3, and subsequently, the defender lost a troop, which resulted in the expected outcome of 10 & 8.
		With Peru having 11 troops and Venezuela occupying 7 (ignoring the log of the previous dice roll), when the attacker's highest value was 5, and the defender's was 3, the expected outcome was to decrement 1 from the defender, subsequently, the 2nd highest value was also greater on the attacker's side, therefore resulting in an overall decrement of –2 for the defender.

Now I can conclude this section of the attack phase as complete, as the dice mechanics now fully work.

Test 13: Transfer Mechanic & Fortify phase (Failed prototype)

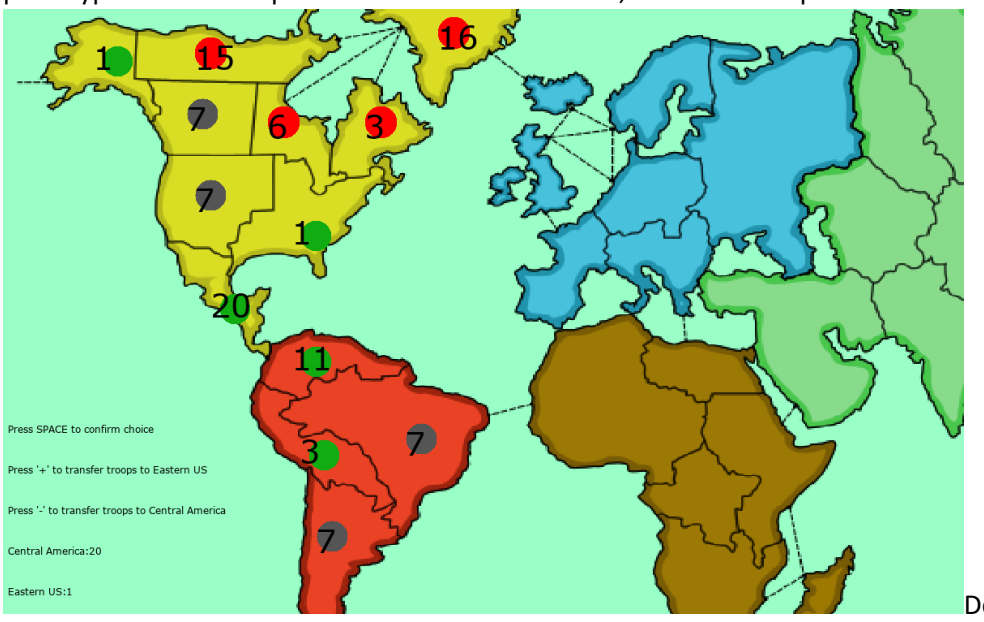
To start off the transfer mechanics, especially for moving the troops between the territories, it is important that I make it a reusable module, that is able to differ slightly dependant on the action. The plan for this is, in order to continue to advance an attack on an enemy territory, upon further problem decomposition is as follows:

When the validation for 0 troops on an enemy territory, and all the subsequent actions are triggered, a function will be called, this function will have the following requirements:

- Filling the screen, adding text instructions onto it
- This box will detail instructions for moving troops
- If a “-” is pressed, it will remove the total number of troops
- If a “+” is pressed, it will increment the number of transferred troops by 1
- If SPACE is pressed, it will confirm these choices
- The text will have the Territory name of both territories, and will increment/decrement according to the keys pressed
- The text will update on the screen as these changes are made

```
            territory_b.remove(territoryb) #deselect the territory so thar it can no longer be attacked by player
            defender_text = font.render("Defender is: None", True, (BLACK)) #update the text on the display
            territory_a.remove(territorya)
            attacker_text = font.render("Attacker is: None", True, (BLACK))
            transfer = False
            TroopTransfer()#triggers the text to be blitted to the screen

def TroopTransfer():
    global transfer, territorya, territoryb, transfer_text,transfer_text_2,instruction_text,instruction_text_2,instruction_text_3
    if transfer == False: #setting all the text values...
        transfer_font = pygame.font.SysFont('Arial',12)
        transfer_text = transfer_font.render(f"{territory_a_name}:{territorya.troopval}",True,BLACK)
        instruction_text = transfer_font.render("Press SPACE to confirm choice",True,BLACK)
        instruction_text_2 = transfer_font.render(f"Press '+' to transfer troops to {territory_b_name}",True,BLACK)
        instruction_text_3 = transfer_font.render(f"Press '-' to transfer troops to {territory_a_name}",True,BLACK)
        transfer_text_2 = transfer_font.render(f"{territory_b_name}:{territoryb.troopval}",True,BLACK)

    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.unicode == "+": #if the + is pressed
                if territoryb.troopval == 1 and territorya.troopval > 1: #as long as there are enough troops
                    territorya.troopval = territorya.troopval - 1 #remove 1
                    territoryb.troopval = territoryb.troopval + 1#to add 1
                    territorya.update_troop_text()#update both..
                    territoryb.update_troop_text()
                    print(territorya.troopval, territoryb.troopval)#terminal output (to see if command has reached thus far)
            elif event.key == pygame.K_MINUS or event.key == pygame.K_KP_MINUS:#repeat same as above
                if territorya.troopval == 1 and territoryb.troopval > 1:
                    territorya.troopval+= 1
                    territoryb.troopval -= 1
                    territorya.update_troop_text()
                    territoryb.update_troop_text()
            elif event.key == pygame.K_SPACE:#to confirm the final changes.
                transfer = True
                territorya = None
                selection = False
                territoryb = None
                second_selection = False#so that territories can be reselected
```

As you can see, this prototype was able to produce text onto the screen, but failed to update:



Despite having pressed the + and – accordingly, nothing happened. In an attempt to correct this, I attempted to move the event loop to the main game() loop in the event that the events cancelled one another out, yet, this still did not reach it, and when I changed the indentation of the keys to the same level as the ’button down’ event, the attack mechanics ceased to function altogether, after hours of testing, it has been concluded that this is not a feasible approach. As well as this, the blit function was added under a boolean value in the main game() loop, yet this still wasn’t successful in anything other

than a one-time blit. Everything else can be reused in the next prototype, aside for the commands, as it appears that the only limitation was through the key mechanism.

Test 14: Transfer Mechanic & Fortify phase

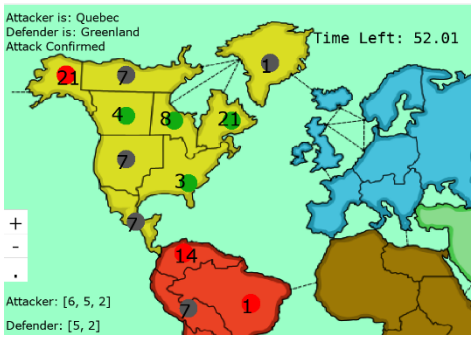
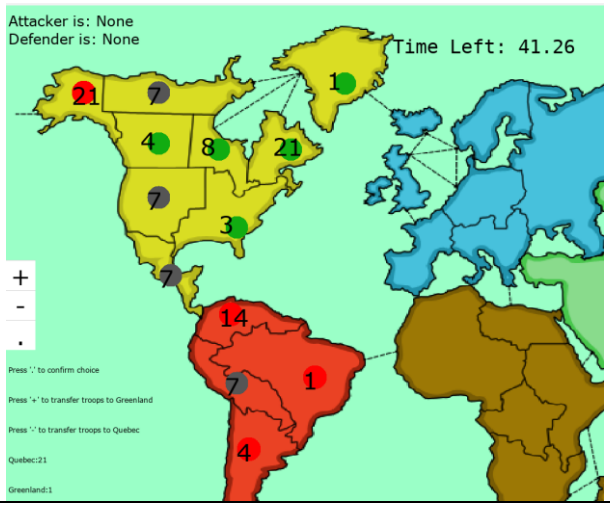
Having assessed the previous failed tests, perhaps a button approach is more feasible, as the game already understands button inputs, especially for updating text. This will also continue to meet the requirements as outlined in the previous prototype.

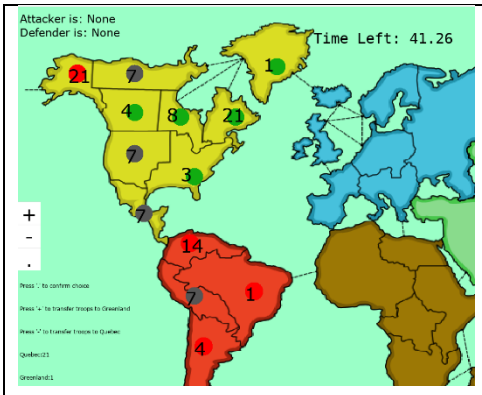
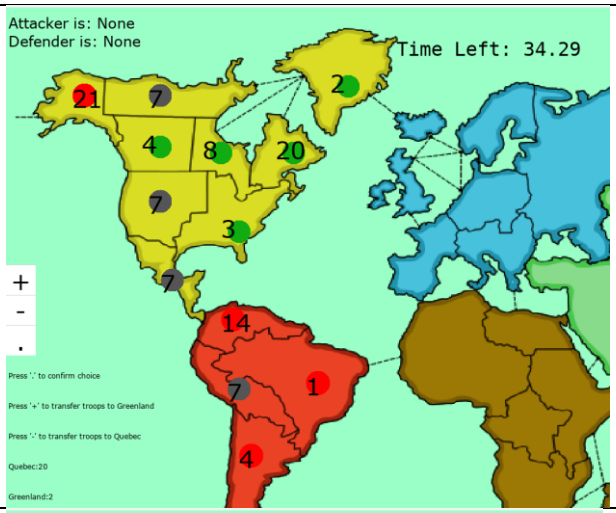
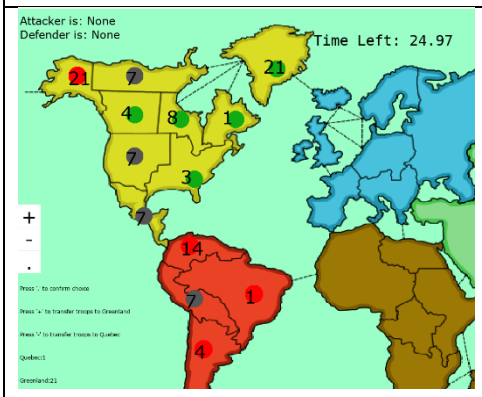
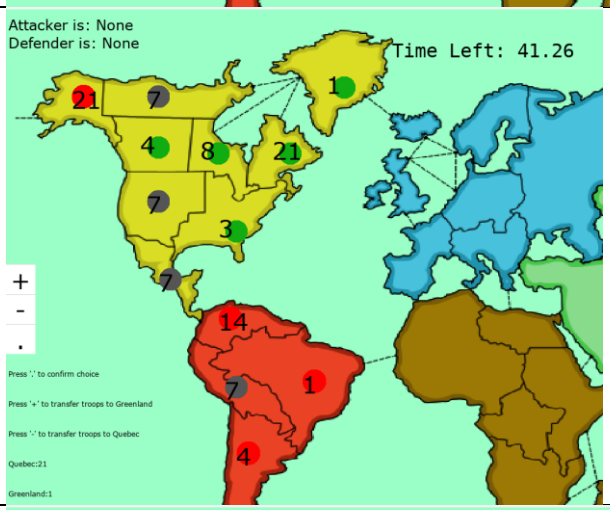
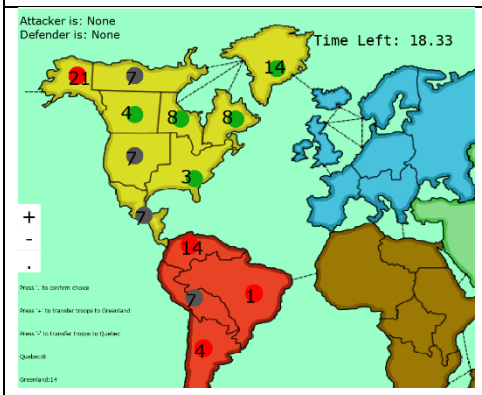
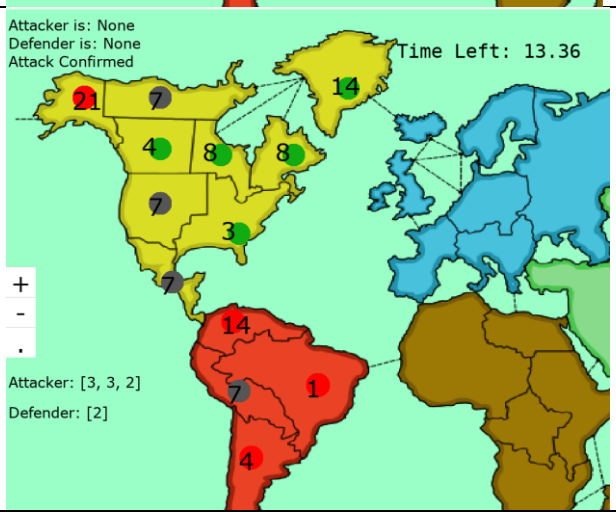
Updated code:

```
            else:
                #print("they aren't neighbours")
                output = font.render("Error: Territories aren't neighbours.",True,(BLACK)) #otherwise tell the user that they cannot do this
                #resets the territories selected
                territory_a.remove(territorya)
                territory_b.remove(territoryb)
                #updates the text to notify the user
                attacker_text = font.render("Attacker is: None", True, (BLACK))
                defender_text = font.render("Defender is: None", True, (BLACK))
                #prevents the cycle from erroring
                selection = False
                second_selection = False
    if TRANSFER_FINALIZE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the user wishes to finalize movement...
        if transfer == False: #if transfer is active
            territorya = None#set territory a to none, so that it cannot be altered now
            selection = False #set selection to False, so as to not error the confirm attack button if pressed
            territoryb = None #set territory b to none
            second_selection = False #same as selection
            transfer = True #set to true, so that these buttons can no longer be used until next transfer
    if PLUS.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '+' is pressed...
        if transfer == False: #if the troops are in the prcess of being moved..
            if territoryb.troopval >= 1 and territorya.troopval > 1: #as long as at least 1 troop on territory being moved from..
                territoryb.troopval += 1#add 1 to the other
                territoryb.update_troop_text()#update to screen
                territorya.troopval -= 1 #remove from current
                territorya.update_troop_text()
    elif REMOVE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '-' is pressed...
        if transfer == False: #if the troops are in the process of being moved
            if territorya.troopval >= 1 and territoryb.troopval > 1: #as long as there is at least 1 troop on the territory it is being moved from...
                territoryb.troopval -= 1 #remove one
                territoryb.update_troop_text() #update the value
                territorya.troopval += 1 #add to the other territory
                territorya.update_troop_text()


    for territory in p1_territories:#if the territory is player 1's…
```

This code has been updated, so that buttons are used instead of keyboard presses. The mechanics act the same as described in the previous prototype.

Test Data:

Before	After	Comments
		This works as intended: Conquered territory validated with +1 on territory, as well as this, the allegience is changed, and the territory system is deselected so you cannot attack your own territory.

		Transfer (testing the + button) As you can see, after pressing the '+', it added 1 troop to Greenland, removing another from Quebec. Thus it passed this test.
		This test checked to see if the troop value capped at 1 on either side. This was also successful, as it did not go below 1 on either territory.
		This is to check to see if the move got finalised, once the '.' was pressed, the resulting test proved that the territory was no longer able to be manipulated.

Now that this is complete, I can conclude the attack phase, as all the mechanics are in place.

Fortify phase

Considering I now have the prerequisites for moving troops through specific buttons, and collision detection of the mouse on the button, a lot of the mechanics in this phase will be very similar to that of the other territories, with the exception of differing validation, however, parts of existing modules can be reused.

This is planned to be as follows:

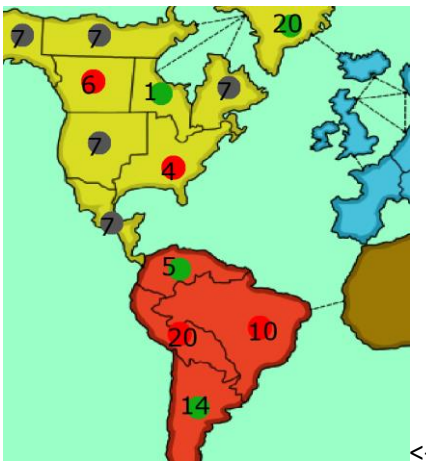
1. A selection system for clicked_territory, with gets reset with every clicked player 1 aligning territory

2. The two territories must both be in the same list, however, they must not be the same territory, this can be done by checking to see if the first indexed element [0] matched with the second [1].
3. These two will then be automatically neighbour checked, however, this is only limited to transferring between two neighbouring territories, as I am limited by programming ability to create a vast system of interconnectivity.
4. The buttons '+', '-' and '.' are then reused, as well as the text instructions
5. At the end of the fortify round, if fortifyphase reaches False, the most recent chosen element is wiped, and the space in [0] is set to None.

To run a test of this in the terminal, the following code was added:

```
if fortifyphase == True:
    for territory in p1_territories:
        if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS):
            clicked_territory = territory #the collided territory becomes selected
            if len(chosen_list) == 0: #if the list is empty
                chosen_list.append(clicked_territory) #add the selected territoriy to the list
                print(chosen_list[0].name) #print the name of the first indexed object
                #print(clicked_name)
            elif len(chosen_list) == 1: #if the list already has an object
                chosen_list.append(clicked_territory) #add the next
                if chosen_list[1] == chosen_list[0]: #if both indexed elements are the same
                    print(f"Not valid:{chosen_list[1].name} must be different to {chosen_list[0].name}")
                    chosen_list.remove(clicked_territory) #remove from the list
                elif chosen_list[0].is_neighbour(chosen_list[1]): #validate if they are neighbours
                    print(f"They match {chosen_list[0].name} is neighbours with {chosen_list[1].name}")
                else: #if they aren't neighbours, remove them
                    chosen_list.remove(clicked_territory)
                    print("Not valid: territories are not neighbours")
                    print(f"{chosen_list[0].name}")#show the only remaining element in the list
            elif len(chosen_list) == 2:#if the list is already full
                chosen_list[0] = chosen_list[1] #element 0 is now element 1
                chosen_list[1] = None #reset element 1
                print(f"first element in list reset {chosen_list[0].name}")
```

Note that at the minute, only print statements were used, just to check to see if the data attempting to be added to the list is valid.



<-- Test iteration of territories

Validation	Terminal output	Works? (Yes/No)
Does the list add only elements in p1_territories when clicked?	`Greenland`	Yes, the program output nothing to the terminal when I pressed an enemy territory
If the list only has 1 element, and the 2nd element is the same as the first, does it get validated out?	`Not valid:Greenland must be different to Greenland`	Yes
If [0] and [1] match under the method .is_neighbour(), does the terminal acknowledge this?	`They match Greenland is neighbours with Ontario`	Yes

If [0] and [1] aren't neighbours, does [1] get removed	Western US Not valid: territories are not neighbours Western US	Yes, but for the first version of the list only. No, if list gets run after element [0] gets replaced with element [1], and [0] is set to None, as it appears the length of the list is still technically 2, so it won't iterate round the list.
If the list is full (with 2 elements), does it reset element [1], so that element [0] becomes element [1], and there is a free space?	first element in list reset Ontario	Yes

Altered code:

```
            elif chosen_list[0].is_neighbour(chosen_list[1]): #validate if they are neighbours
                print(f"They match {chosen_list[0].name} is neighbours with {chosen_list[1].name}")
            else: #if they aren't neighbours, remove them
                chosen_list.remove(clicked_territory)
                print("Not valid: territories are not neighbours")
                print(f"{chosen_list[0].name}")#show the only remaining element in the list
        elif len(chosen_list) == 2:#if the list is already full
            chosen_list[0] = chosen_list[1] #element 0 is now element 1
            chosen_list[1].remove(clicked_territory) #remove element 1, so that the length is now 1
            print(f"first element in list reset {chosen_list[0].name}")
```

The last object is removed from the [1] space in the list, so as to ensure the list is always reset to 1 object.

Now that all terminal responses are fixed, I will set all this out as visual on-screen outputs. (all print statements will be code commented out)

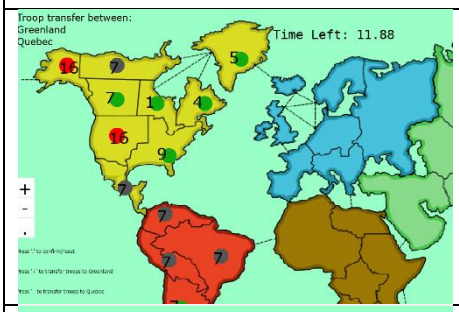
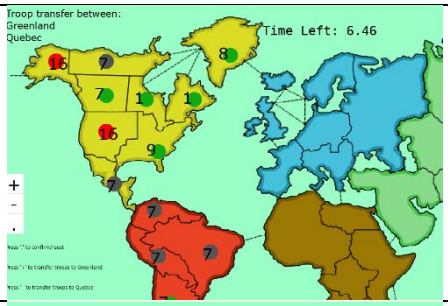
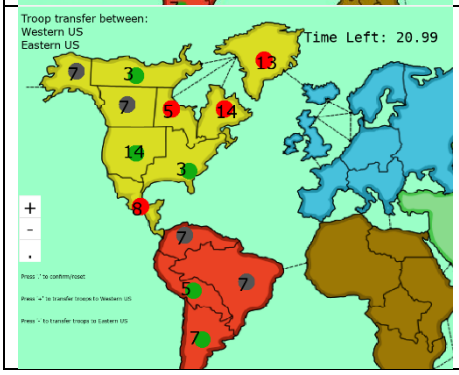
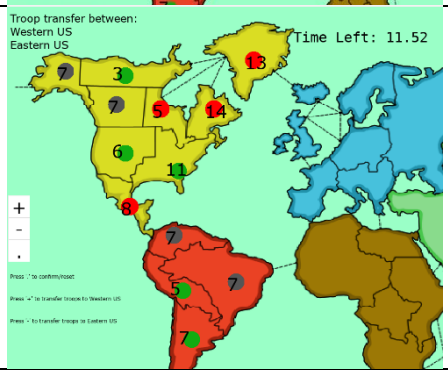
```
                territory_b.remove(territoryb)#if it is 1 then remove it - so deselects
                second_selection = False
    if fortifyphase == True:
        for territory in p1_territories:
            if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS):
                clicked_territory = territory #the collided territory becomes selected
                if len(chosen_list) == 0: #if the list is empty
                    chosen_list.append(clicked_territory) #add the selected territoriy to the list
                    #print(chosen_list[0].name) #print the name of the first indexed object
                    #transfer_between = font.render(f"Troop transfer between:",True,BLACK)
                    element_zero = font.render(f"{chosen_list[0].name}",True,BLACK)
                elif len(chosen_list) == 1: #if the list already has an object
                    chosen_list.append(clicked_territory) #add the next
                    if chosen_list[1] == chosen_list[0]: #if both indexed elements are the same
                        #print(f"Not valid:{chosen_list[1].name} must be different to {chosen_list[0].name}")
                        chosen_list.remove(clicked_territory) #remove from the list
                    elif chosen_list[0].is_neighbour(chosen_list[1]): #validate if they are neighbours
                        #print(f"They match {chosen_list[0].name} is neighbours with {chosen_list[1].name}")
                        element_one = font.render(f"{chosen_list[1].name}",True,BLACK)
                    else: #if they aren't neighbours, remove them
                        chosen_list.remove(clicked_territory)
                        #not_valid = font.render("Not valid: territories are not neighbours",True,BLACK)
                        #print(f"{chosen_list[0].name}")#show the only remaining element in the list
                elif len(chosen_list) == 2:#if the list is already full
                    chosen_list[0] = chosen_list[1] #element 0 is now element 1
                    #chosen_list[1] = None
                    chosen_list.remove(chosen_list[1]) #remove element 1, so that the length is now 1
                    element_one = font.render(f"None",True,BLACK)
                    #print(f"first element in list reset {chosen_list[0].name}")
        if len(chosen_list) =
```

All declared font surfaces are called inside an if statement, embedded in a while loop of the game() function:

```
if fortifyphase == True:
    screen.blit(transfer_between,(10,10))
    screen.blit(element_zero,(10,40))
    screen.blit(element_one,(10,70))
    #screen.blit(not_valid(10,100))
```

```
            if len(chosen_list) == 2:
                fortifying = True
        if TRANSFER_FINALIZE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the user wishes to finalize movement...
            if fortifying == True: #if transfer is active
                chosen_list.remove(chosen_list[1])#set first selected none, so no alteration
                chosen_list.remove(chosen_list[0]) #set territory b to none
                fortifying = False
                element_zero = font.render(f"None",True,BLACK)
                element_one = font.render(f"None",True,BLACK)
        if PLUS.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '+' is pressed...
            if fortifying == True: #if the troops are in the prcess of being moved..
                if chosen_list[0].troopval >= 1 and chosen_list[1].troopval > 1: #as long as at least 1 troop on territory being moved from..
                    chosen_list[1].troopval -= 1#add 1 to the other
                    chosen_list[1].update_troop_text()#update to screen
                    chosen_list[0].troopval += 1 #remove from current
                    chosen_list[0].update_troop_text()
        elif REMOVE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '-' is pressed...
            if fortifying == True: #if the troops are in the prcess of being moved..
                if chosen_list[1].troopval >= 1 and chosen_list[0].troopval > 1: #as long as there is at least 1 troop on the territory it is being moved from...
                    chosen_list[0].troopval -= 1 #remove one
                    chosen_list[0].update_troop_text() #update the value
                    chosen_list[1].troopval += 1 #add to the other territory
                    chosen_list[1].update_troop_text()
```

As you can see from this screenshot, the principle remains very similar to the transferral. Here is the test result from this:

GUI before	GUI after	Comments
		The troop cap was validated, after doing this, was still able to select and deselect the transferring locations.
		Able to move troops between two selected locations

Now that it has been established that this mechanism works, that will conclude the phases for player 1.

Test 15: The game can be won upon elimination of opposing territories

To test out if the game can be won, I am creating a new function called won_game(), which will be triggered when the length of either the player's list or enemy's list is 0. For now, to set up the conditions to test if a player has won, I will set Player 1's troops to 25, and then when the attacking commences, I can overtake all the territories and check to see if these preconditions are met.

The preconditions for winning the game:

When making the temporary change in default, I noticed a bug with the neutral deployment, as when the default value exceeded 11, it endlessly iterated through the territories, and was able to give itself an infinite number of troops, thus, I have had to fix this bug by replacing the if statement with a while loop:

Before	After

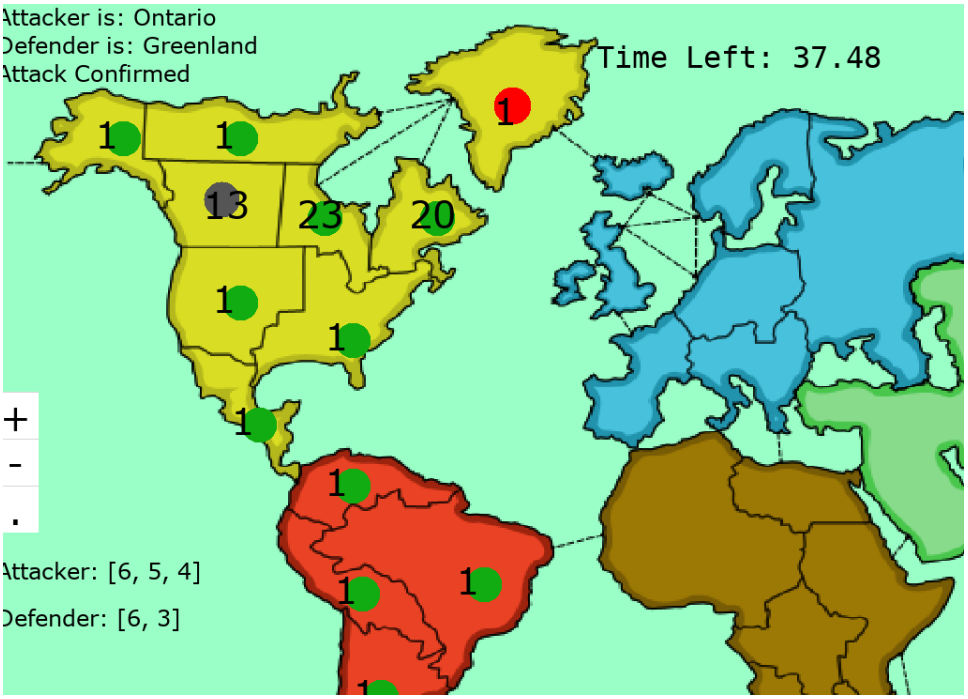
```
def NeutralDeploy(): #adds half of what the player recieves, randomly adding it
    global available_troops, n_done
    neutral_available_troops = available_troops//2  #has half val, available currently set 4 for default
    for territory in neutral_territories: #if neutral
        if neutral_available_troops > 0 and n_done == False: #when not 0 or finished
            random_territory = random.choice(neutral_territories) #select..
            chosen_list.append(random_territory)
            territory.troopval += 1 #add 1 to the selected
            neutral_available_troops -= 1#remove from total
            territory.update_troop_text()#update the text
            chosen_list.remove(random_territory) #prevents constant endless iteration
            print(neutral_available_troops)
        if neutral_available_troops == 0:#prevents endless iteration
            n_done = True
```

```
def NeutralDeploy(): #adds half of what the player recieves, randomly adding it
    global available_troops, n_done
    neutral_available_troops = available_troops // 2
    while neutral_available_troops > 0 and n_done == False:#while the requirements arent met...
        random_territory = random.choice(neutral_territories) #pick a random territory
        chosen_list.append(random_territory)#add it to the current edited territory
        random_territory.troopval += 1 #add to the troopvalue
        neutral_available_troops -= 1#decrement the available troops by 1
        random_territory.update_troop_text() #update the text on screen
        chosen_list.remove(random_territory)#remove it from the list
        random_territory = None#set to none, so it wont duplicate itself in the list
        #print(neutral_available_troops)
        if neutral_available_troops == 0:#if it reaches 0, disable the while loop
            n_done = True
```

Besides this, this is the code that I used to check the requirements and thus fill the screen accordingly when the game has been ‘won’ (the function is called from within the event loop in the game() function:

```
def win_screen():
    if len(p1_territories) > len(enemy_territories):
        winningalias = "Player"
    else:
        winningalias = "AI Player"
    while True:
        screen.fill(BISQUE)

        winner_font = pygame.font.SysFont('Arial',72)
        win_text = winner_font.render(f"{winningalias} Wins!", True,BLACK)
        win_subtext= winner_font.render("Close screen to exit",True,BLACK)

        screen.blit(win_text,(800,900))
        screen.blit(win_subtext,(800,1000))
```

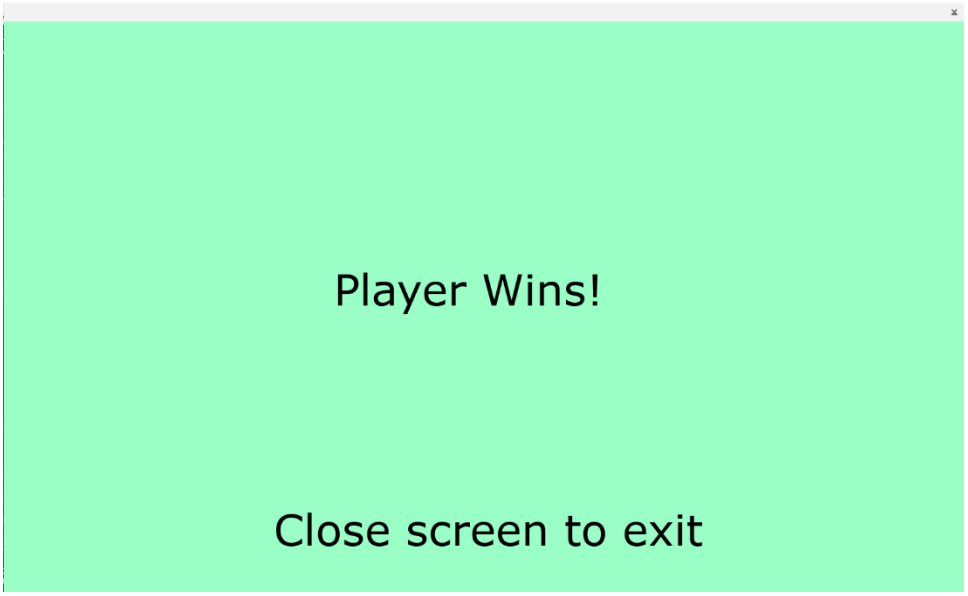
RESULT: The game froze, when attacking the final of the enemy player’s territory, this may have been in part due to the while loop in place.



Amended code:

```
def win_screen():
    if len(p1_territories) > len(enemy_territories):
        winningalias = "Player"
    else:
        winningalias = "AI Player"
    while True:
        screen.fill(BISQUE)

        winner_font = pygame.font.SysFont('Arial',72)
        win_text = winner_font.render(f"{winningalias} Wins!", True,BLACK)
        win_subtext= winner_font.render("Close screen to exit",True,BLACK)

        screen.blit(win_text,(550,400))
        screen.blit(win_subtext,(450,800))


        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        pygame.display.update()
```

Test trial 2:

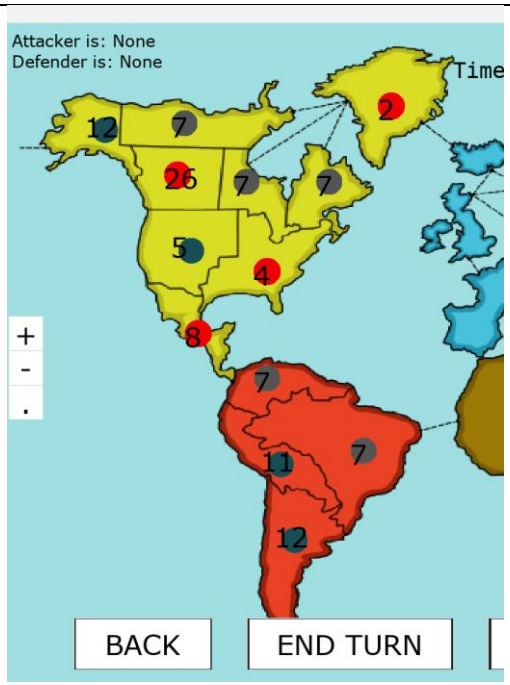
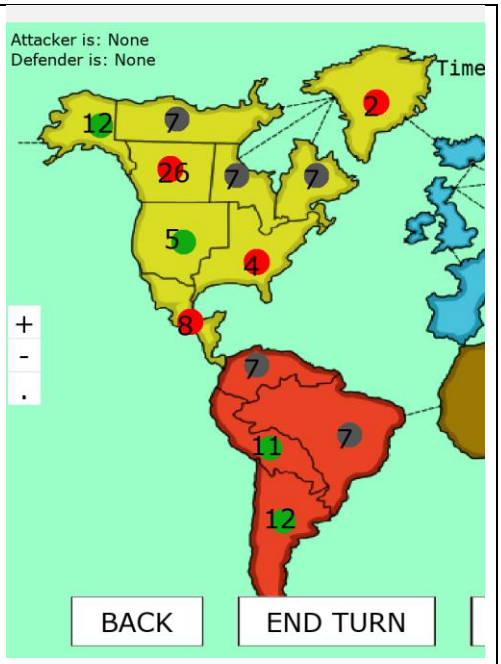
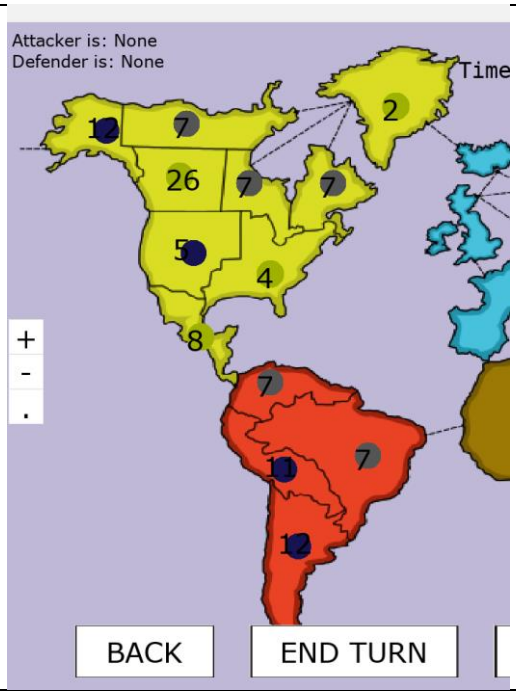
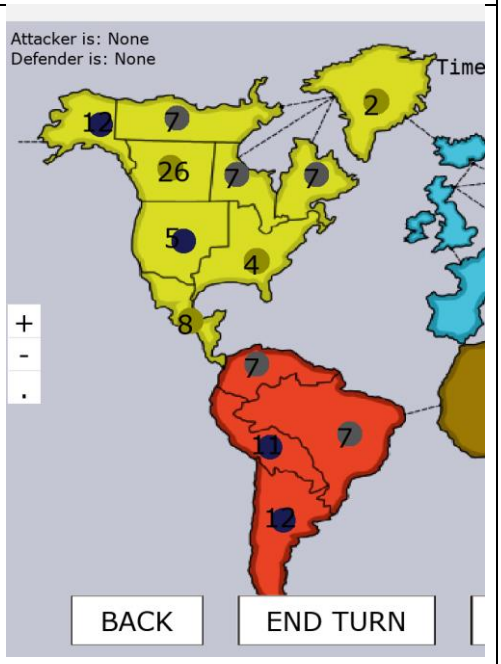


This is a very basic screen, and now the main game mechanics are completed (until further bugs are revealed post-development tests).

Test 16: Settings – Colourblind settings & rules

To add in usability features, making use of new methods inside of the Territory class in classes.py will be the best way to optimise my code for efficiency and reusability, the colour change for the allied territories can be altered through swapping out the surface image on the territories, once a button has been pressed. This should be relatively quick to implement, as the technique to do this is essentially the same as any other menu in my solution. Here is a sample of this (the code does vary for each button, but the theory is applied the same):

```
Rules = MenuButton(menu_button,800,700,"Rules") #rules button
Rules.HighlightByMouseHover(SETTINGS_MOUSE_POS)
Rules.update()

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    if event.type == pygame.MOUSEBUTTONDOWN:
        if SETTINGS_BACK.checkForInput(SETTINGS_MOUSE_POS):
            main_menu()
        if Tritanopia.checkForInput(SETTINGS_MOUSE_POS): #if the tritanopia button is clicked..
            BISQUE = (159,223,225) #change the background colour
            ally_button = pygame.image.load("Images/trit_ally.png")
            ally_button = pygame.transform.scale(ally_button,(60,60)) #change the colour scheme for the ally
            enemy_button = pygame.image.load("Images/trit_enemy.png")
            enemy_button = pygame.transform.scale(enemy_button, (40,40)) #change the colour scheme for ai
        if Deuteranopia.checkForInput(SETTINGS_MOUSE_POS):#if the deuteranopia button is clicked..
            BISQUE = (191,184,215)#change the background colour
            ally_button = pygame.image.load("Images/deutro_test.png")
```

Tritanopia -->		Default -->	
Deuteranopia -->		Protanopia -->	

To conclude my stage 2, I am adding my rules to the settings, the plan is for the user to be able to cycle through with < and > interactable buttons like a slideshot, which will demonstrate what every thing is for.

As this applies very similarly to every other menu I have created, I will show a sample of the code, as well as the different slides.

Review of stage 2

What has been done so far (excluding components from Stage 1)

Now, following the deploy phase, you can now engage in the attack mechanics with any opposing players. A key difference from stage 1 is the addition of another continent: South America. The mouse click is validated to check that only two neighbouring territories that are either player (territory a) or neutral/ ai (territory b) can face off against one another. Once these have been selected, if you press 'CONFIRM ATTACK', it rolls the dice, comparing the first two elements (if applicable, otherwise it is the first elements) against one another, the dice rolls are sorted so as

to ensure that the first element in the list is the highest, if two players highest or 2nd highest produce the same score, only the player loses a troop, so as to keep in line with validation rules of minimum 1 troop on a territory at all times.

If it is found that the player only has 1 troop to attack with, the territory is automatically deselected, and they have to choose another valid option. If an attack is won, the enemy territory transfers over to a player territory, and will automatically have 1 troop added onto it, from here, the player can add or remove troops between the two territories, provided that it leaves at least 1 troop remaining on either side. When the player is satisfied with the transfer, they must press the '.'. Throughout this part, instructions will pop up onto the side of the screen where the dice results were output previously.

When the attack phase is finished, the game launches into the fortify phase, where they can only transfer between territories that they own, and are connected with one another. Like the transfer mechanics, instructions are output onto the screen, and the buttons are reused for the player to use, where the troops can be moved up until only 1 is remaining on either side, however, movement is also bi-directional. When the player wishes to finish, they press the '.', they can then select another set to move troops between if they so wish as long as the timer has not run out.

A player can now win the game if either the territories of the ai or player is extinguished (excluding the neutral territory, as it cannot attack), from here a screen will be blitted, thus ending the game, and from here it displays the winner, and outputs instructions on how to close the window.

As well as this, usability has been added in the form of colourblind modes, which was prompted by my stakeholders, these can be changed at any time, and the colour scheme can always be reset to default if the user so wishes. As well as this, the rules box has been added, but as of yet it does not do anything due to time constraints faced in designing my program, as is reflected in my evaluation.

Finally, I have also fixed a few bug issues regarding the deploy mechanics, so now the neutral player can add more troops than territories own without an exponential (and broken) addition of infinite troops, as well as this, once the first iteration is passed, it automatically launches into the next phases, not halting. The turn based counter has also been improved, ready for implementation of the ai player.

How was it tested

Yet again, with issues involving numerical values and lists, a terminal response has been used in the form of print() statements, so as to keep a record of the flows of data. In terms of GUI, I have added additional text that is blitted to the screen, especially for the attack and deploy, as well as dice mechanics, so it was easier to keep track of whether the rules of the game were being followed through appropriately.

How it meets success criteria

In its current state, it now is a fully functioning prototype in terms of player experience, lacking only in the ai player's turn (as the neutral player is now fully complete in terms of capabilities). The continued tweaking of the turn based system have allowed me to fix a few development-related bugs, which has allowed me to create a flowing system of phases. My notifications outputs has also helped to enhance user experience, as it offers instruction on the ongoing processes. The fixing of the neutral force has provided a level of power to mitigate the rapid expansion of any player, which is another success criteria requirement. As well as this, my dice function also meets user requirements in order to be able to advance through the attack stage.

Criteria fully met:

- **An on-screen clock should be displayed on screen**
- **Territories must be divided equally, randomly and fairly.**
- **I would like to add a button that can skip phases and end turns prematurely.**
- **A simple menu should be used to link together features of my project**
- **The game can be quit at any given time, with the use of a 'Quit' button**

- **A card system should be implemented into the game**
- **The neutral force should be powerful, but not unbeatable**.

Criteria being partially met:

- **I would like the game to follow a turn-based structure**
- **It would be suitable to add a notifications box.**
- **The dice function & probability statistics should be in accordance with one another – the dice function is fully implemented, the probability statistics are to be included in stage 3.**

Changes in design from original plans

A few details about the transfer of troops had to be changed, due to my program being unable to interpret keyboard pressed, thus I had to adapt to this through the implementation of user interactive buttons. As well as this, more instructions were blitted to the screen, so as to ensure that the user and myself would be able to follow the actions of the game.

As well as this, due to the amount of validation inside of my project, I have decided to forgo my plans for user logins and save data, as this would be far too complex at my current stage, and I also lack the time constraints to thoroughly implement and test this, thus it was decided that the overall worth this would have brought to my project in its current state is fairly underwhelming.

Overall summary

Overall, my solution produces a user interface, and a thoroughly tested experience for all the phases (for the player), the neutral player is now complete, alongside the player, leaving just the ai. Next to be added is the rest of the territories, as well as the statistics, so that comparisons and appropriate conclusions can be reached by the ai player. The ongoing development of the turn-based structure means that I can now get started on the ai player.

Stage 3 – AI development

Test 17: Statistical probability

To calculate statistical probability, I will reuse many of the variables from the dice roll, including attacker_dice_count and the territorya and territoryb troopcount. (territory a and b will differ depending on whose turn it is)

The general formula that seems most quantifiable for producing a win % likelihood is:

Likelihood = (attack/total troops from both territories) x 100

However, the attack and defence troops used will be adjusted in this calculation from a simulation of dice rolls that works like normal, which will affect the overall %. To prevent too much complexity and mathematical uncertainty involved, I am limiting this simulated dice roll to the number of necessary dice per turn. (so 3 for attacker = 2 dice is rolled, this is then compared against the highest element/two elements of the defender).

Prototype 1: Modularity

```
def win_probability():
    global territorya, territoryb
    #Reusing the dice mechanics for the dice roll
    if territorya.troopval == 3:
        attacker_dice_count = 2
    elif territorya.troopval == 2:
        attacker_dice_count = 1
    else:
        attacker_dice_count = territorya.troopval - 1

    if territoryb.troopval == 1:
        defender_dice_count = 1
    else:
        defender_dice_count = 2

    # Roll the dice for each side - a simulation
    attacker_rolls = sorted([random.choice(range(1, 7)) for x in range(attacker_dice_count)], reverse=True)
    defender_rolls = sorted([random.choice(range(1, 7)) for y in range(defender_dice_count)], reverse=True)

    # Compare the dice rolls and remove the appropriate number of troops (the simulated results)
    for i in range(min(attacker_dice_count, defender_dice_count)):
        if attacker_rolls[i] > defender_rolls[i]:#if attacker's result is highest
            territoryb.troopval -= 1#remove from defender
        else:
            territorya.troopval -= 1 #otherwise -1 from attacker

    # Return the win probability for the attacker
    likelihood = f"{(territorya.troopval / ((territorya.troopval) + territoryb.troopval) * 100)}%"
    #print (str(likelihood))
    return likelihood
```

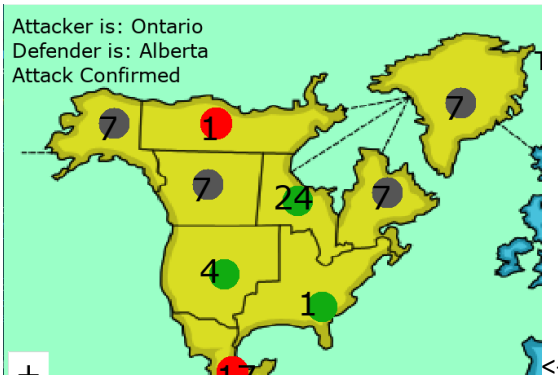
```
            self.text = menu_font.render(self.text_input, True, RED) #allows the user to know whether the cursor is by the text
        else:
            self.text = menu_font.render(self.text_input, True, "black")#resets back to black

    def HighlightStats(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
            self.text = str(likelihood)
            text_surface = font.render(self.text,True,BLACK)
            pygame.draw.rect(text_surface,BISQUE,[self.x_pos, self.y_pos + 20])
```

Having coded this solution, due to the modular nature of my files of classes.py and NEA_main_code.py, I realise that despite integrating two aspects of these files, to get a pop-up for the user would require at least an additional 100 lines of code, and will further complicate my solution, therefore, this function will solely be used for ai comparison data instead, so I will remove the method .HighlightStats(), and instead use a way of calling the win_probability function for every comparison that the ai makes.

For now, to test the win probability, I have added it to the selection iteration:

```
                    #print(len(chosen_list))
                    #screen.blit(territory.trooptext,(territory.x_pos,territory.y_pos))
                else:
                    print("You cannot access this at this time")
if attackphase == True:#if the attack phase in in progress
    if CONFIRM_SELECTION.checkForInput(GAMEBUTTON_MOUSE_POS): #if the confirm selection has been clicked
        if selection == True and second_selection == True and transfer == True: #as long as both values have been submitted
            if territorya.is_neighbour(territoryb): #if the territory a is neighbours with the selected enemy territory...
                #print("they are neighbours")
                output = font.render("Attack Confirmed",True,(BLACK)) #change the text to attack confirmed
                win_probability()
                diceroll()
            else:
```

Response (Terminal)



<-- GUI as reference (24 vs 7)

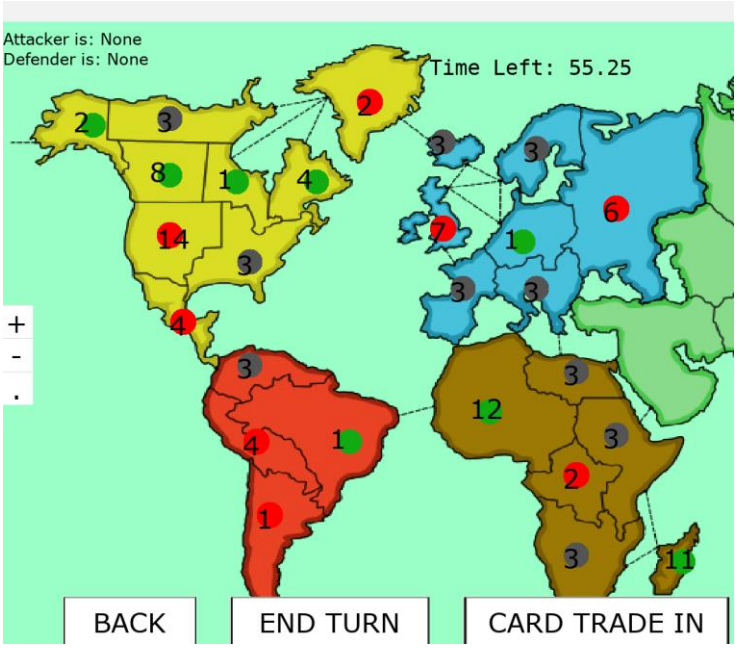
```
chances of attacker winning is:83.87096774193549%
```

This single response is all I need from the calculation stage, therefore, this statistics stage is concluded.

Test 18: Adding more territories and ai deploy

Due to all the territories being added for finishing the map and their neighbouring validation, it is once again a repetition of code already previously seen, thus I am not going to screenshot it here. The code can be found in the concluding copy of my code project at the end of this document, marked under its respective section.

Here is an updated version of the map (and all I need to test the AI player):



Now, the deploy mechanics for the ai player is a hybrid of the troop receival experienced by the player, but also the distribution enacted by the neutral player.

A lot of changes had to be made to ensure that the ai could run smoothly, as it requires duplicate variables which are assigned under different names.

```
            n_done = True

def AIDeploy(): #adds half of what the player recieves, randomly adding it
    global ai_available_troops, ai_done,attackphase,deployphase,time_remaining
    while ai_available_troops > 0 and ai_done == False:#while the requirements arent met.
        random_territory = random.choice(enemy_territories) #pick a random territory
        chosen_list.append(random_territory)#add it to the current edited territory
        random_territory.troopval += 1 #add to the troopvalue
        ai_available_troops -= 1#decrement the available troops by 1
        random_territory.update_troop_text() #update the text on screen
        chosen_list.remove(random_territory)#remove it from the list
        random_territory = None#set to none, so it wont duplicate itself in the list
        if ai_available_troops == 0:#if it reaches 0, disable the while loop
            attackphase = True
            deployphase = False
            time_remaining = 70
            ai_done = True
```

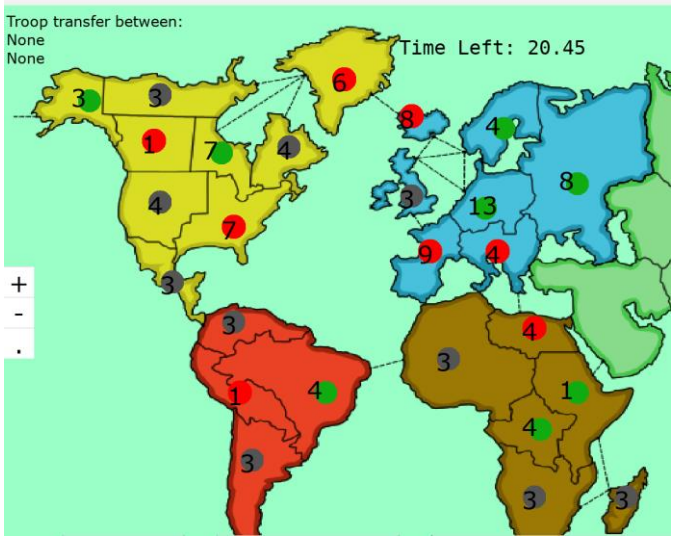
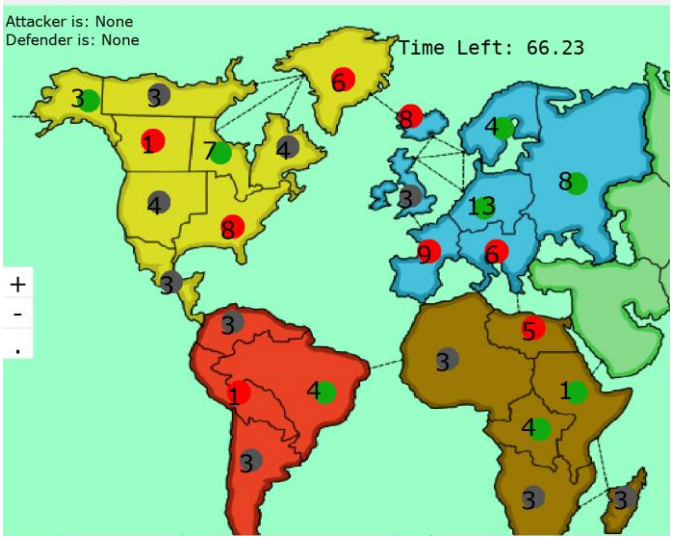
As you can see here, a new function called AIDeploy was created, it is a near-duplicate to the NeutralDeploy() function, with the exception of a different number of troops available, however, due to the integration of new variables, I had to edit the existing phase statements to include more boolean arguments.

(
```
TypeError: '>' not supported between instances of 'NoneType' and 'int'
```
)

```
            else:
                available_troops = available_troops + 10 #add 10 to the total once.
                troop_done = True
                return available_troops
    elif ai_turn == True:
        if troop_done == False: #if it not unactivated
            if  enemy_territories in NAterritories:#if the 1st player's territories cannot be found in the NA list as a whole
                #if generate == True:
                ai_available_troops = default + NorthAmericaBonus #make the total 8
                troop_done = True#make done
                return ai_available_troops
            if enemy_territories in SAterritories:
                ai_available_troops = default + SouthAmericaBonus
                return ai_available_troops
            else:
                NorthAmericaBonus = 0
                SouthAmericaBonus = 0
                ai_available_troops = default + NorthAmericaBonus + SouthAmericaBonus#make total 3
                troop_done = True
                return ai_available_troops


available_troops = DeployTroops()
ai_available_troops = DeployTroops()
```

One such example was the integration of a new defining value inside of the troop calculation loop, specifically suited to troops residing in the enemy_territories list. Once this was done, I had to manually edit every loop for the phases, however, this will require further editing as I advance through the other phases.

```
                        #print("defender deselected:",(territory_b_name))
                        defender_text = font.render("Defender is: None", True, (BLACK))
                        territory_b.remove(territoryb)#if it is 1 then remove it - so deselects
                        second_selection = False
            if fortifyphase == True:…

        if attackphase == True and first_iteration<1:…
        if attackphase == True and first_iteration>=1:…
        elif fortifyphase == True and first_iteration<1 and p1_turn == True: #continue as normal…
        elif fortifyphase == True and first_iteration>=1 and p1_turn == True: #this will allow to end the turn…
        elif fortifyphase == True and first_iteration>=1 and ai_turn == True: #this will allow to end the turn…
        elif deployphase == True and p1_turn == True:…
        elif deployphase == True and ai_turn == True:…

        if turn_count % 2 == 0: #if the turn count(automatically set to 0) is divisible by 2, and the remainder is 0, then it is…
        else: #if there is 1 left over, it is ai's turn…

        pygame.display.update()

erritories = [
  Alaska, NWTerritory, Alberta, Ontario, Quebec, Greenland, WUS, EUS, Central_America, Venezuela, Peru, Brazil,
  Argentina, Iceland, GB, North_Europe, West_Europe, Scandinavia, South_Europe, East_Europe, N_Africa, Egypt, Congo
  ,South_Africa, East_Africa, Madagascar]
utral_territories = [
```

Deploy before AI's turn	Deploy after AI's turn
	 In this example, the ai was given 4 troops, and chose to add them to Eastern US (+1), Southern Europe (+2) and Egypt (+1)

After running this, I ran through the next phases and found that it only adds to the ai territories once every 2 turns (player and ai's), where the player could not edit the ai and the ai could not edit the player's, thus, this can conclude the ai deploy phase as a success.

Test 19: AI attack & AI attack validation

Due to having completed the win statistics function so that it will output a number (%), I can now implement this alongside the attack validation. Once again, this attack validation is a mirror of what the player experiences in terms of territorya, territoryb, however, there are differences, as instead of territory in p1_territories being assigned to attacker(territorya), instead it will be territory in enemy_territories being assigned to attacker, and vice versa.

```
if ai_turn == True and attackphase == True:#if it is the ai's turn and the attack phase has commenced
    if time_remaining != 0:#as long as the time is not 0
        random_territory = random.choice(enemy_territories)#picks a random territory
        territorya = random_territory#assigning it to territorya
        print(territorya.name)#prints out the name (for test purposes) - to keep track of terminal responses
        random_territory = None#resets to None, so will not endlessly iterate
        random_territory = random.choice(p1_territories+neutral_territories) #picks a new territory
        territoryb = random_territory#assigns this to territory b
        print(territoryb.name)#print this for testing purposes
        random_territory = None
        if territorya.is_neighbour(territoryb):#if they are neighbours
            win_probability()#calculate the win probability
            print(likelihood)
            if likelihood > 30:#if the win probility is greater tha 30%
                #print(likelihood)
                diceroll()#make the move
            else:
                time.sleep(0.1) #time.sleep in place to put limitations on ai
        else:
            time.sleep(0.1)
```

Test data (to see if this works):

GUI window,

```
Eastern US
Quebec
```

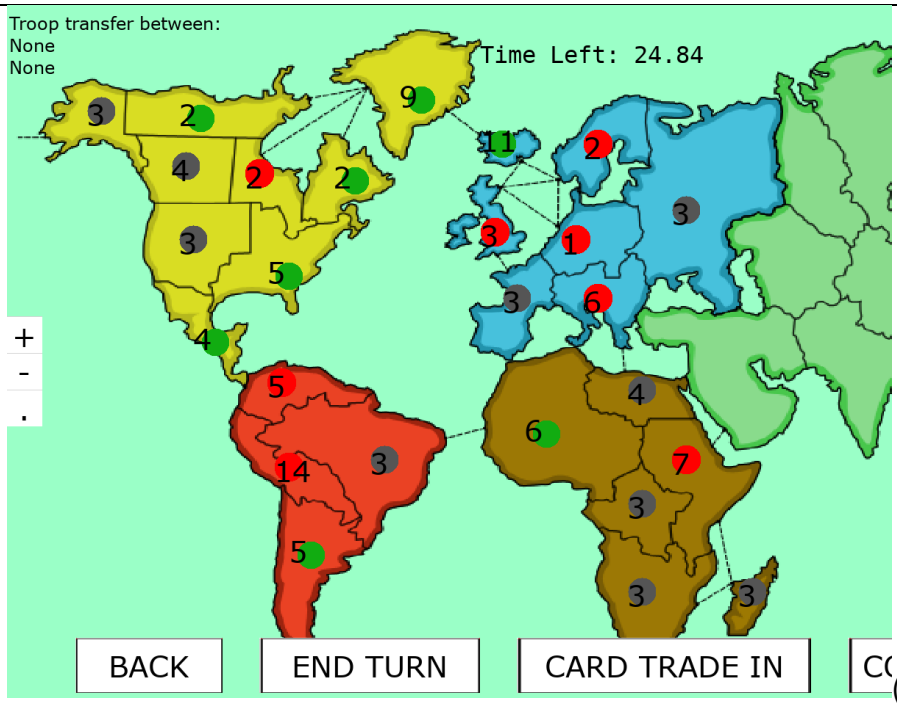
```
    likelihood = (territorya.troopval / ((territorya.troopval) + territoryb.troopval) * 100)
ZeroDivisionError: division by zero
```

When running this test, the game errored due to selecting a territory which had only 1, due to the win_probability calculation deducting 1 from the total number of troops, this would have caused an error result of 0, both due to Qubec having no troops, thus breaking the validation rule, and also making the calculation unable to perform.

```
attacker_rolls = sorted([random.choice(range(1, 7)) for x in range(attacker_dice_count)], reverse=Tr
defender_rolls = sorted([random.choice(range(1, 7)) for y in range(defender_dice_count)], reverse=Tr

attacker_troops = territorya.troopval #creates a copy - preventing erroring
defender_troops = territoryb.troopval #copy

if territorya.troopval or territoryb.troopval != 1:
    # Compare the dice rolls and remove the appropriate number of troops (the simulated results)
    for i in range(min(attacker_dice_count, defender_dice_count)):
        if attacker_rolls[i] > defender_rolls[i]:#if attacker's result is highest
            attacker_troops -= 1#remove from defender
        else:
            defender_troops -= 1 #otherwise -1 from attacker
else:
    pass

# Return the win probability for the attacker
#likelihood = f"{(territorya.troopval / ((territorya.troopval) + territoryb.troopval) * 100)}%"
#print (f"chances of attacker winning is:{likelihood}")
likelihood = (territorya.troopval / ((territorya.troopval) + territoryb.troopval) * 100)
return likelihood
```

<-- This is what I have changed the code to.

Test set

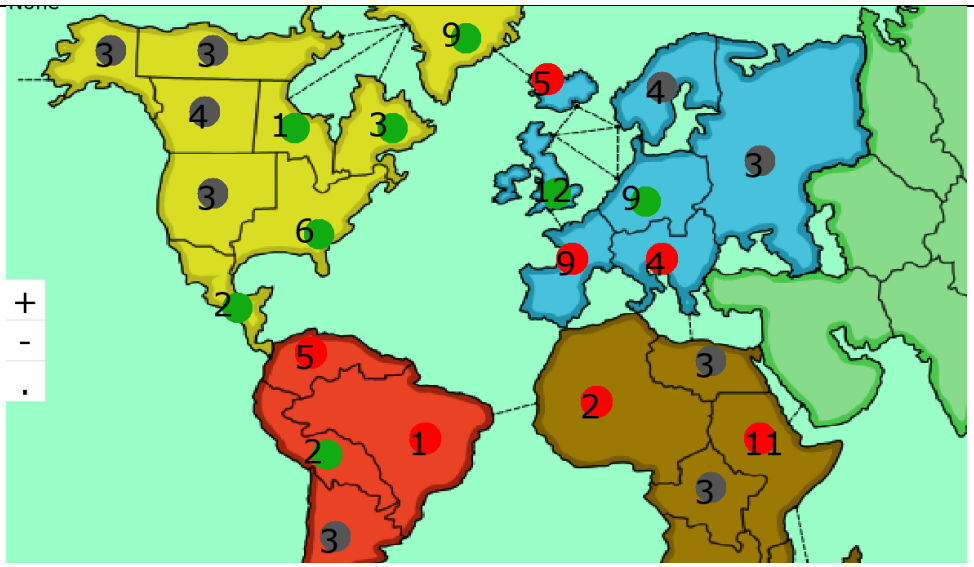
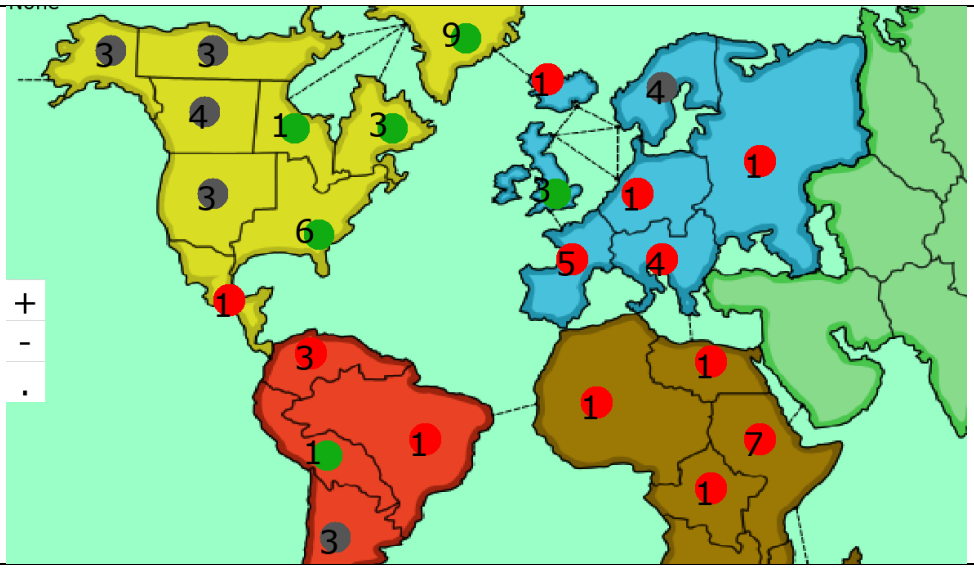
	Test 1
Before	 (screen pre attack on transfer on p1 turn)

After	 (screen following ai attack phase, in the ai fortify phase)
Comments	The dice does roll, if a valid attack it is shown in terminal. Does change allegience, however, futher validation required to ensure that result of win probability is always above 0. Also, some territories end up with –1 and 0 troops, this needs validating out too. Overall, when a valid combination is found, it does ultimately attack the enemy, meaning that the overall effect is good. The issue here is mainly due to the removal of the transfer boolean.

Code amendments

```
CONFIRM_SELECTION = GameButtonClass(elongated_game_buttonv2, 1350,900, "CONFIRM ATTACK")

CONFIRM_SELECTION.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
CONFIRM_SELECTION.update()

if ai_turn == True and attackphase == True:#if it is the ai's turn and the attack phase has commenced
    if time_remaining != 0:#as long as the time is not 0
        random_territory = random.choice(enemy_territories)#picks a random territory
        territorya = random_territory#assigning it to territorya
        if territorya.troopval == 1:
            territorya = None
            random_territory = None
            random_territory = random.choice(enemy_territories)#picks a random territory
            territorya = random_territory
            print(territorya.name, territorya.troopval)
        elif territorya.troopval <= 0:
            territorya.troopval = 1
        else:
            print(territorya.name,territorya.troopval)#prints out the name (for test purposes) - to keep track of terminal responses
        random_territory = None#resets to None, so will not endlessly iterate
        random_territory = random.choice(p1_territories+neutral_territories) #picks a new territory
        territoryb = random_territory#assigns this to territory b
        print(territoryb.name, territoryb.troopval)#print this for testing purposes
        random_territory = None
        if territorya.is_neighbour(territoryb):#if they are neighbours
            win_probability()#calculate the win probability
            print(likelihood)
            if likelihood > 30 and territorya.troopval != 1:#if the win probility is greater tha 30%
                #print(likelihood)
                diceroll()#make the move
            else:
                time.sleep(0.05) #time.sleep in place to put limitations on ai
        else:
            time.sleep(0.05)

if transfer == False: #setting all the text values  w
```

	Test 2

Before	
After	
Comments	When the attack is utilised, it transfers 1 troop. From there, it does not duplicate troops and there are no longer any errors involving troopcounts of 0 or less. It also cannot attack itself, and is forced to follow the rules of the game.

Now that the attack has been established as working and following the base game rules, I can conclude the end of the AI player's attack, as a final note before moving on however, to slow the progress of the AI's advances, I have limited the timer by half of what the player gets. This helps me to keep in line the success criteria of the game being winnable against the AI player.

Review of Stage 3

What has been done so far (excluding components from Stage 1 & 2)

Once player 1's turn is over (after the first 2 iterations), both the ai player and neutral player deploy their troops – the neutral territory gains half of the available troops of the ai player. Once these have been expended, the game will then launch into a reduced time for attack (due to the decision making of the ai being far more proficient than the player). From here, two territories are picked, and these are then validated to see if they meet the requirements for attack, additionally, once a valid move is identified, the win probability is calculated to see the chance of the ai winning their attack. If the value calculated has more than a 30% chance of success, it will roll the dice and attack. When the player/neutral territories' troopcount reaches 0, the ownership it automatically updated to become the ai's owned territory. Unlike the player, in the attack phase, the ai cannot transfer a select number of troops to their new territory; this was done due to uncertainty reasons.

Due to additional validation, I have also ensured that the ai must have more than 2 troops on a territory if they wish to attack the enemy player's territory, this has prevented infinite expansion, thus making the game winnable, and providing a level of fairness towards the player. The ai player also does not have access to the phase resources in the same manner as the player, and while the ai's turn is being actioned, the player is unable to interact with any button other than the 'back' button, from which they can change the colour scheme if they wish to do so.

As well as this, additional territories have been added onto Africa and Europe.

How was it tested

For intense testing such as that for the ai attack phase, time.sleep() has been utilised with print statements on the selected territories' name and troopcount in order to record valid moves and ensure that the ai player follows the base rules of the game, followed by the win likelihood (if applicable). The extra information provided by the likelihood in the terminal was useful in pinpointing valid moves that were actually enacted by the ai player, as it could be spotted easier among processes iterated through every half a second.

To check to see if the ai player's moves were inflated by unfair advantages such as extra troop gain outside of the intended paramaters of the game, before and after screenshots were taken of the map, and compared with one another, as well as this, I kept the dice outcomes displayed on the screen, so as to follow the winning results.

How it meets success criteria

In its now finalised state (with the exception of post-development testing), the moves of the ai, neutral and human player are complete, following the appropriate validation. Reasonable limitations have been placed on the ai to prevent uncontrollable/ unmanageable expansion, thus meeting that particular requirement. Due to how the ai processes were coded, the game now meets the requirements of a turn based system, as every time the turn count is divisible by 2 without a modulus, it is the players' turn, otherwise it is the ai's turn.

Criteria fully met:

- **An on-screen clock should be displayed on screen**
- **Territories must be divided equally, randomly and fairly.**
- **I would like to add a button that can skip phases and end turns prematurely.**
- **A simple menu should be used to link together features of my project**
- **The game can be quit at any given time, with the use of a 'Quit' button**
- **A card system should be implemented into the game**
- **The neutral force should be powerful, but not unbeatable**.
- **I would like the game to follow a turn-based structure**
- **It would be suitable to add a notifications box.**
- **The dice function & probability statistics should be in accordance with one another – the dice function is fully implemented, the probability statistics are to be included in stage 3.**
- ***The game should be winnable against the AI.***
- **The AI should follow all pre-set rules of the game**

Criteria not met:

- **Saving the game should be available from logged in users**.
- **Validation should be used for the login system**

Changes in design from original plans

Due to the sheer size of adding the rest of the map, I decided to add every other continent bar Australia and Asia, however, if I end up with enough time post development, i will add these.

The login and save data system had to be forgone due to complexity reasons, as well as necessity, as the validation served the purpose of adding a layer of input validation, however, in order to follow the base rules of the game and its various functions, an extensive level of validation was required anyway. As well as this, the save game function required the use of assets which my chromebook had difficulty processing, thus this had to be excluded from final development.

Overall Summary

The program in its current state is has now finished development, and is ready to be handed over to stakeholders for any further improvement. Any changes that will be made or bug fixes will be listed in post-development testing. To summarise the capabilities of my program, during the player's turn, they can interact with the graphical user interface and make changes accordingly dependant on the current rules of the phases. The neutral player passively deploys upon every turn, and can only defend itself. During the ai player's turn, the player is unable to alter any elements (aside from the back button, to access the settings) until the timers are up, and their turn has been activated.

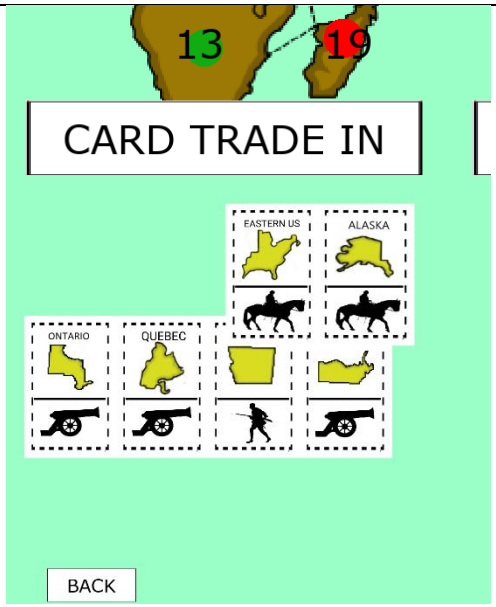
Evaluation

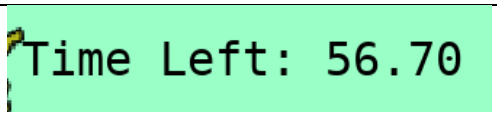
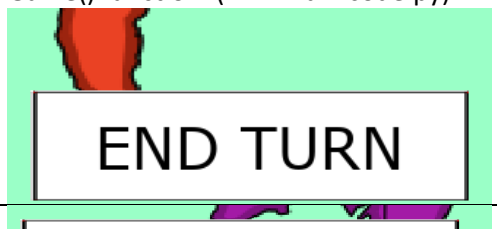
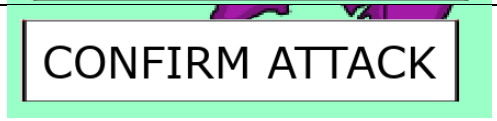
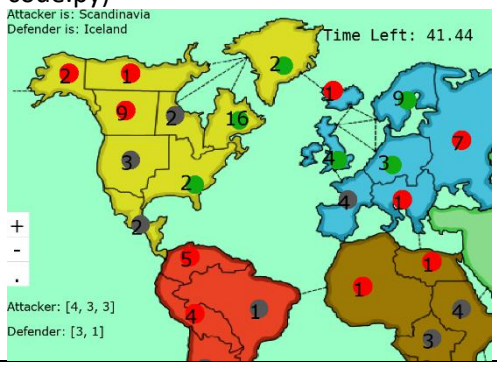
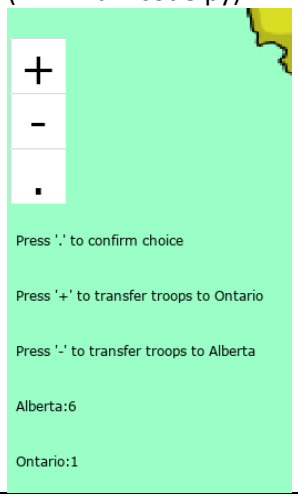
Testing to inform evaluation

Post Development Testing

For any points added to my post development table featured in my Design section, I have created the following tables to identify specific elements encompassed in the features being tested (where applicable – the tests planned for login, which I was unable to incorporate into my solution due to time constraints is addressed in my project limitations and criteria not met) , where every aspect is broken down into an abstracted summary of what it does. (i.e. to summarise card mechanics, I've put 'card trade in'.)

Mechanics and phases – buttons & features

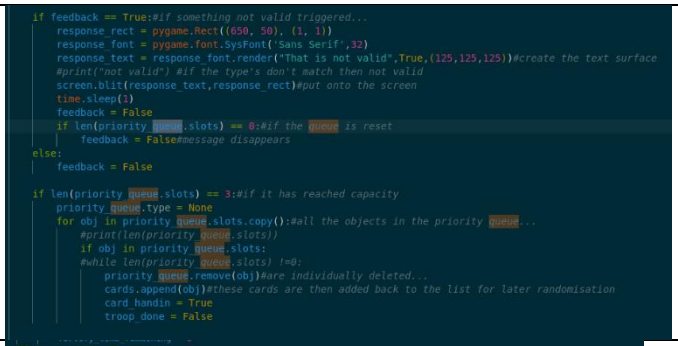
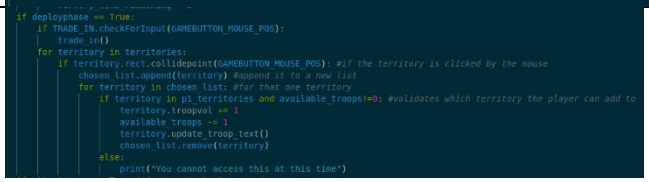
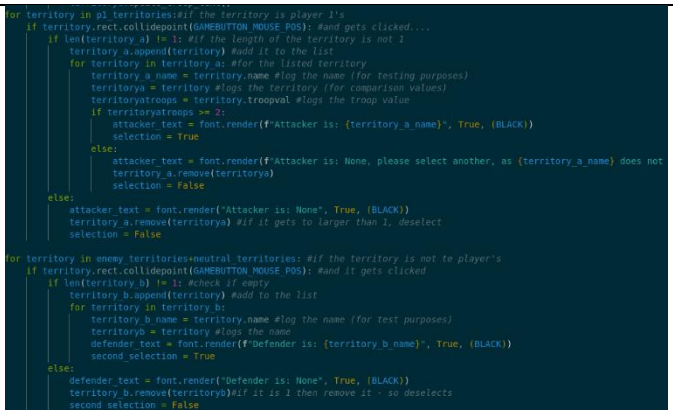
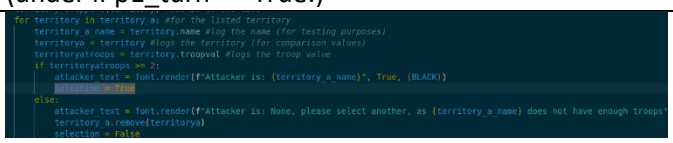
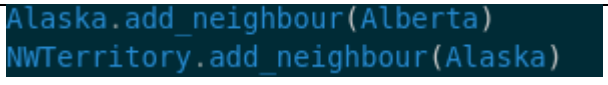
Summary of base mechanic	Result/ reference to coded location	Evaluation of robustness
Card Trade In	 Key features used are called from CardDistribution() and trade_in(), which are interconnected with other parts of the program. (NEA main code.py)	As intended, the user can only interact with the button properly during the deploy phase, when attempted to be used during other phases, the button is non-responsive. When deploy is active, if clicked, it brings the user to a new screen, which loads their cards to an interactable queue. Overall very robust – Card type can be changed when cards removed from priority, auto adds troops when expended, cards disappear and don't reappear preventing infinite exploits. However, one small issue with this is to do with the text output, as the time displayed is temperamental, and should be reformatted for better visibility.

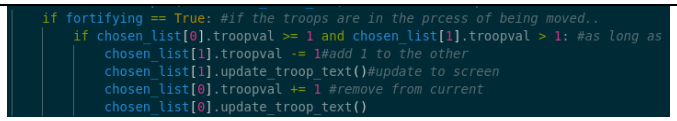
End Turn & Timer	 Key features used are found inside the Game() function. (NEA main code.py) 	As intended, is overall very robust in conditions for user interaction with end turn, works regardless of time set initially. Timer varies dependant on user and phase, which is good for variation. Only (once again small) issue is with button being present during deploy phase when there is no timer – despite this, pressing whilst on deploy does nothing and does not force transition to next phase, thus showing success. However, a success in terms of formatting is that the timer text disappears once the time reaches 0.
Confirm Attack	 All actions for confirm attack found in event loop of Game(). (NEA main code.py) 	Once again, the robustness is relatively good, as only interactable when user has enemy and player territories selected, when pressed it triggers the dice function which updates the text and calculates the dice effect on troops. Like other buttons, the only setback is related to the visibility of the button – as it is present (although not usable) when other phases are active, which could cause confusion for the end user about phase relevance.
Transfer Troops	Located inside of the Game() function. (NEA main code.py) 	Transfer troops shown to be reasonably robust, text blitted to screen when relevant to help user understand the meaning behind the 3 buttons, helping to guide the user. Works as supposed to, updating the troop text on the relevant locations. On the whole it is robust, and it validates with the rules of the game, however, under certain conditions it experiences a bug (more on this is listed in maintenance). Previously mentioned potential setbacks of buttons also apply.
Text Output	Located throughout the file (NEA main code.py), examples of this seen throughout many screenshots of my project, for example the screenshot in the field above.	Text output robust in parts – an example of most robust is during player 1's phases when selecting and deselecting locations, when the dice is rolled and related instructions, however, due to time constraints, when appearing during ai phase, text such as selected territory does not update (although user can still see when dice has been rolled or when a territory visually conquered.)
Setup – Territory Distributi on	Found in territorydistribution() function called at start of game function. Evidence as seen from image surface of every territory.	Very robust – code itself is concise, and distribution mechanics applied once, then not recurringly used until next time program is executed as a whole. Ensures that

		territory distribution is as equal as possible – with the neutral territory gaining any extras.
Setup – Troop Distribution	Found in troopdistribution() function, also called at start of the Game() function. Evidenced as seen from text surface throughout documented screenshots.	Troop distribution – robustness limited by randomness – although the random aspect of the troop distribution is within a certain range (minimum 1, maximum = maximum number of leftover troops), it does, in some cases, produce distributions where a lot of territories have 1 troop (and others have a disproportionately large number), however, with the more territories added to the map, this problem has nullified.

NOTES on table above: Code associated with all these mechanics (with the exception of text output) either call their own function, or are added within for event() loops, and can be seen in the finalized code concluding this project report. The relevant code location is referenced (where/if applicable).

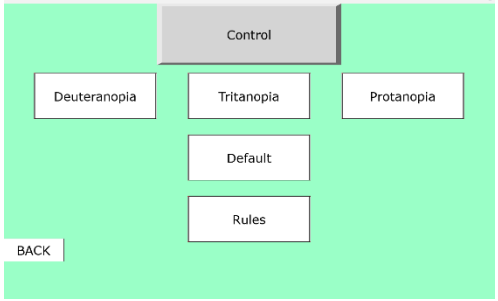
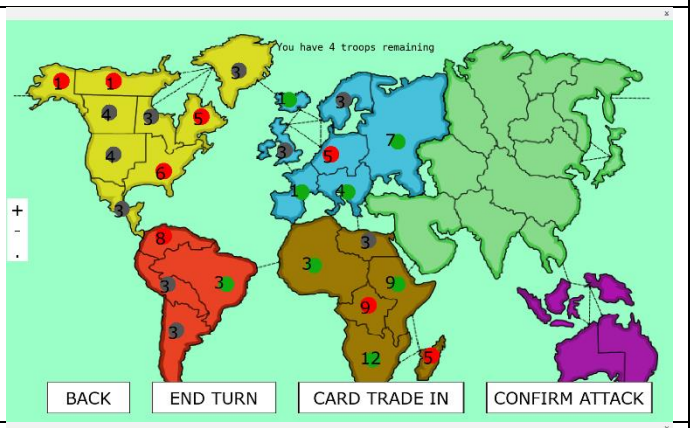
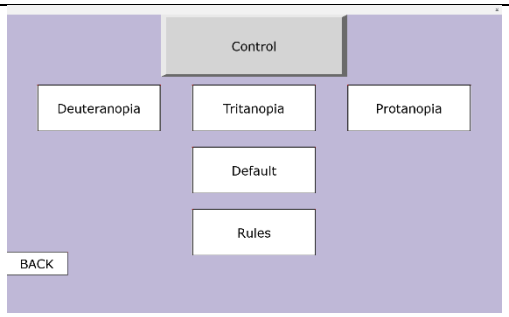
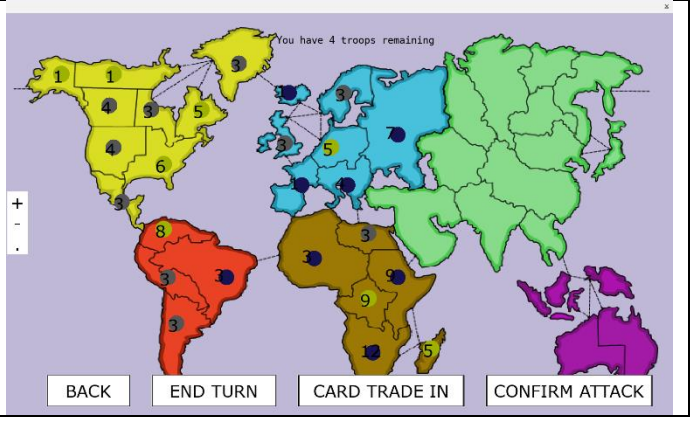
Validation – Rules for the phases (most notable)

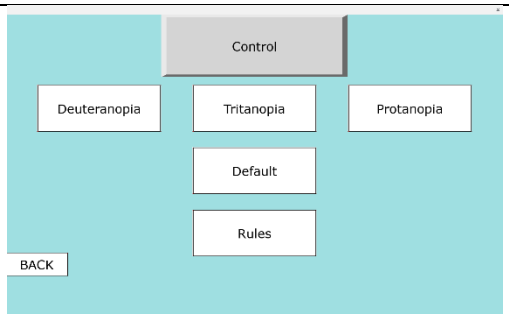
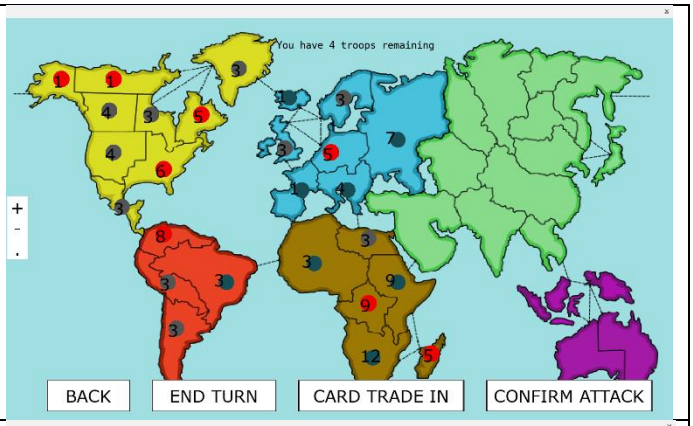
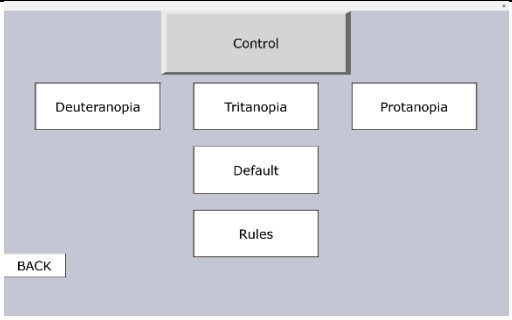
	Sample code of this	Evaluation of robustness
Card selection validation		Fully robust in terms of card type selection/ deselection, ordering of queue, as well as troop addition.
Deploy: Player only placement		Fully robust for both players: always checks to see if the clicked or selected territory conforms to the current turns' player list.
Attack: Player vs enemy	 (under if p1_turn == True:)	Fully robust for both players, inclusive of subject conditions of turn.
Attack: Attacker must have at least 1 troop		Fully robust in project, text also output to user during their turn if attack attempted with 1 troop.
Attack: Neighbouring territories		Fully robust, however, due to the nature of defining neighbours, although .is_neighbour() method helpful in reducing lines of

		code, the individual defining of every possible combination unavoidable.
Fortify: Neighbouring and turn player only		Fully robust, it was useful to index under one list as both territories meet same requirements and only required comparison between their variation.
Fortify/ Transfer mechanics: Cannot exceed range of 1 to territory.troopval -1		Mostly robust, in most cases works as intended, however, when subject to some conditions, a bug is found (more details found in maintenance section)

NOTES on table: Proof of the above table's validation is evidenced throughout many instances of screenshots in this documentation.

Usability Testing – Menus & Colourblindness

	Menu Screens (Settings shown, but opening menu follows the same scheme)	Main Game
Default		
Deuteranopia		

Tritanopia		
Protanopia		

```
if event.type == pygame.MOUSEBUTTONDOWN:
    if SETTINGS_BACK.checkForInput(SETTINGS_MOUSE_POS):
        main_menu()
    if Tritanopia.checkForInput(SETTINGS_MOUSE_POS): #if the tritanopia button is clicked..
        BISQUE = (159,223,225) #change the background colour
        ally_button = pygame.image.load("Images/trit_ally.png")
        ally_button = pygame.transform.scale(ally_button,(60,60)) #change the colour scheme for the ally
        enemy_button = pygame.image.load("Images/trit_enemy.png")
        enemy_button = pygame.transform.scale(enemy_button, (40,40)) #change the colour scheme for ai
    if Deuteranopia.checkForInput(SETTINGS_MOUSE_POS):#if the deuteranopia button is clicked..
        BISQUE = (191,184,215)#change the background colour
        ally_button = pygame.image.load("Images/deutro_test.png")
        ally_button = pygame.transform.scale(ally_button,(60,60))#change the colour scheme for the ally
        enemy_button = pygame.image.load("Images/deutro_enemy.png")
        enemy_button = pygame.transform.scale(enemy_button, (40,40))#change the colour scheme for ai
    if Protanopia.checkForInput(SETTINGS_MOUSE_POS):
        BISQUE = (197,198,212)#change the background colour
        ally_button = pygame.image.load("Images/protan_test.png")
        ally_button = pygame.transform.scale(ally_button,(60,60))#change the colour scheme for the ally
        enemy_button = pygame.image.load("Images/protan_enemy.png")
        enemy_button = pygame.transform.scale(enemy_button, (40,40))#change the colour scheme for ai
    if Default.checkForInput(SETTINGS_MOUSE_POS):
        BISQUE = (154,255,199)#change the background colour
        ally_button = pygame.image.load("Images/ally.png")
        ally_button = pygame.transform.scale(ally_button,(60,60))#change the colour scheme for the ally
        enemy_button = pygame.image.load("Images/enemy.png")
        enemy_button = pygame.transform.scale(enemy_button, (40,40))#change the colour scheme for ai
```

On reflection of this, I did attempt to reduce typing out the line multiple times, however, using a method for the class territory did not work. Overall, the colourblind features was received well by the end users, and the main formatting was also changeable at any point, as lines up with the success criteria. The program would have benefitted from additional usability features outside the realm of colourblindness, however, due to time constraints, I was unable to include many additional features. Despite this, however, I was still able to deliver on most of my criteria.

Aspects of my program overall

Code commenting	In areas where the code itself may be unique or difficult to keep track of, especially whilst variables used are extensive, I have used plenty of code comments to explain what is going on. However, with large sections of code where the same thing is happening in repetition, such as the .add_neighbour()

	method being called, does not require sufficient explanation beyond what is already stated once. In these sections, I have marked the collective code as a summary, such as the variable declarations at the start of the program, where describing every line is not applicable. The range of type of code commenting does vary between actively describing on a line-by-line basis, to just summarising a section, mostly due to the nature of my project. In my final project file, all old prototypes that had been not used were discarded from the overall file, including any unused print statements and such.
Indentation	Indentation was useful in keeping track of specific areas of my code, and ensuring that only certain sections are checked at specific times, which came in especially handy for parts which defined the overall structure of the game as attack<fortify<deploy or the ai and player turns. Indentation was also useful in the fact that it allowed me to hide previously completed sections of code, so that I could abstract out details which were irrelevant to the solution at that stage, so that I could easier identify bugs and issues.
Sub programs	Sub programs, whether it be the modular nature of my functions to retrieve specific sets of data or the different sub programs involved during the event and different outcomes of user input validation, were very useful to help to create some reusable components that could be recalled by different parts of the program, which reduced time requirements in rewriting different sections tailored to different areas, however, despite this, I still did face many limitations with this, as further acknowledged in my limitations section.
Naming conventions	My naming conventions, upon reflection, did generally lack consistency, which is an area of which needs further improvement in the future, despite this, for different components of my solution, I would mostly use one form of naming convention, such as the use of underscores instead of spaces, CamelCase or general caps lock (like what is seen with a large proportion of my buttons). This lack of consistency did, unfortunately, make my program harder to remember variable names, which is why for future reference it may be better to just stick to one.

Overall checklist

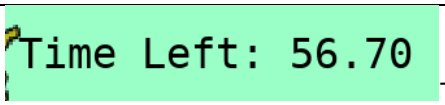
Testing action	***Does it work? (Yes/No)***
Program menu's revert back to previous ones upon selection	*Yes*
The program can be closed at any given time upon selection	*Yes*
Colourblind settings load upon menu interaction	*Yes*
Colourblind settings have the ability to revert back to default	*Yes*
The timer will run whilst other processes are running	*Yes*
All territories interlink and influence one another	*Yes*
Player distribution is in effect upon game loading	*Yes*
The player is able to interact with cards	*Yes*
Cards can be collected throughout the game	*No*
The save game feature is able to be written	*No*
The save game feature can be read from/ loaded	*No*
The player is able to login	*No*
The correct dice is used in comparisons when attacking with variable numbers of troops	*Yes*
The win statistics/ probability are clearly shown to the player	*No*

The timers/ phases load into one another	*Yes*
All rules of the phases are followed	*Yes*
The game shows the winner	*Yes*
The AI interacts with territories	*Yes*
The neutral force is included in setup/ deployment	*Yes*

For items in this checklist that do not work, more details on those are provided in my limitations and/or maintenance section(s) of this project, otherwise, the evidence is referenced multiple times in this project, most notably in my various tests in development.

Evaluation of Solution

Solution evaluation with success criteria

Criteria	Fully met/Partially met/ Not met and test evidence (where applicable)	Explain, focusing on details outlined in analysis (where applicable)
An on-screen clock should be displayed on screen	Fully met Test evidence of this can be found in Test 8	 The timer counts down from x (x being number set for that round) to 0 in 1 second increments, to 2 decimal places. When 0 is reached, the next phase is launched, and if it is the AI's turn, then the user is unable to interact with the GUI.
I would like the game to follow a turn-based structure	Fully met Test evidence of this can be found in Tests 9-14, and 18-19	Only one player can take their turn at a time, when the turn is on a specific phase, the text output infers this to the user.
The AI should follow all pre-set rules of the game.	Partially met Test evidence of this can be found in Tests 18-19	The speed is limited, as well as its timer, the decision is made once every 0.5 seconds, however, a combination appears to be found once every 5 or so seconds (on average), meaning that it doesn't instantly dominate the map. It also follows the same deploy rules and mechanics as the player. However, due to running out of time for the implementation of the ai's fortify phase, it means that in technicality, not all game rules are being followed.
Territories must be divided equally, randomly and fairly.	Fully met Test evidence of this can be found in Test 6	The 'random' module is applied to a list of countries, this is iterating a set number of times, until the counter reaches 0. Each run of the game distributes different player's territories to different starting locations.
I would like to add a button that can skip phases and end turns prematurely.	Fully met Test evidence of this can be found in Test 9	Apart from the deploy phase, if a player has finished their attack or fortifying before the countdown reaches 0, they can finish their turn early. A button is on the bottom right of the screen which sets the timer to 0.

It would be suitable to add a notifications box.	Partially met Test evidence of this can be found in Tests 11-14	The notifications box allows the player to read what is happening and is present while the game is being played. Despite this, there are no customisable options as outlined in Analysis, and there are some bugs involving the text updating, especially surrounding the AI.
The game should be winnable against the AI.	Fully met Test evidence of this can be found in Tests 14-15	Due to having access to the card mechanic and the ability to transfer troops, the player does hold a comparative advantage, despite enacting their moves at a slower rate, however, it does use statistics to decide if a move holds enough probability of winning before taking the risk of attacking.
The neutral force should be powerful, but not unbeatable.	Fully met Test evidence of this can be found in Test 10	The neutral force gains half that of the ai/ player per deploy phase, and can only defend itself, thus naturally having a comparative disadvantage when it comes to dice rolls.
Saving the game should be available from logged in users.	Not Met Test evidence N/A	The logistics involved with MySQL applications would not run on my chromebook, thus it creates a hardware limitation, due to this, I was unable to code or test this in my solution in the event that it worked.
A simple menu should be used to link together features of my project	Fully met Test evidence of this can be found in Tests 1-2, 16	A menu GUI links to settings or the main game, and the back button links all pages to one another.
The game can be quit at any given time, with the use of a 'Quit' button	Fully met Test evidence of this can be found in Test 2	A 'quit' or 'back' button is available on every screen, including the main game screen. The placement of the button is mostly out of the way yet still clearly visible to the user.
A card system should be implemented into the game	Fully met Test evidence of this can be found in Test 10	If the user's turn is being actioned, and the deploy phase is active, upon clicking the trade in button, it loads the cards, these cards can then be traded in if they are the same type.
Validation should be used for the login system	Not Met Test evidence N/A	Although there is no login system, input validation is still used in the program. This was not added due to time constraints.
The dice function & probability statistics should be in accordance with one another	Fully met Test evidence of this can be found in Tests 11-12, 17-18	During the AI phase, probability is used as a threshold to decide whether the dice should be rolled in attacking the enemy territory. If it is, then the AI will attack until that territories' allegiance is changed to theirs.

Further development

In terms of future development, the notifications box issue can be resolved if I resorted to using the same font surface design, and swapped out this variable on command, a similar principle would apply to the text colour or background colour, and this would be implemented through the use of buttons, very similar to how the colourblind features are implemented. For the pre-set rules, simply using a random number within range of what is available and the minimum account for what is left would be assigned to a variable, this variable would then be assigned to the += and -= values on territory.troopval. Once this is done the selected territory will either be removed from its assigned list or set to None. From here it will then probably iterate through the territories again in a similar way to the attack phase, until a valid combination is found. For login and validation, I could either use the terminal to input text to the screen and I would benefit from doing further research on transporting information to an external file, which would also help me to apply this to saving game data, where anything loaded in a field can be read and applied to a variable, or written to the file. This, however, would also require extensive testing and a larger audience to apply it to, as it would allow me to save game data from a variety of users. Additional areas of further development can be extensive surrounding additional usability features, such as changing font sizes, and potentially changing factors such as window size or reformatting aspects of my program so that it can enhance user experience. If I'd have had more time, I also would have added a rules menu, where you can cycle through the different rules and their respective phases (similar to a slideshow).

Success criteria for usability

Usability Feature	Success/Partial Success/ Failure (Overall) to comply with this	Reason for criteria (Incl. Stakeholder feedback post development)
Intuitive map	Success	The game board itself is easy to read and understand, with clear colour & number markings for territories. All stakeholders that were given access to play this version of the file were able to successfully distinguish between different territories.
Input consistency	Success	By having input consistency reduced to mouse clicks, it reduces the cognitive load on the user, as there are no extensive keyboard controls on the user. So when given to my stakeholders, they were quickly to learn and remember how to perform the game's tasks. This frees up mental resources that can be used for other tasks, such as problem-solving or decision-making, with general feedback on this being overall positive.
Ease of use – clearly outlining rules	Failure	Due to time constraints, I was unable to produce a 'rules' menu inside of my product itself, thankfully, my stakeholders were mostly casual gamers, and thus knew how to play the game already, however, this means that for those who are new to the rules will have some difficulty adapting, thus this is a feature that I would have added if I had more time.
Well designed game pieces	Partial Success	The game pieces themselves are mostly easy, however, my solution does have its limitations, as a collision bug and formatting of the pieces of my map meant that the collision was off-centre, this was always meant to be resolved towards the end of my program, however, due to time constraints I

		was unable to do this. Despite this, it still feasibly works and stakeholders could adapt to this.
Turn indicators	Partial Success	The only limitation, it was found with this, was that the text output onto the screen did not explicitly say whose turn it was, and what the phase is, however, for those that know the game, with the text such as the timer, the selection counters, dice, it did imply what phase was in motion, the effect of this varied, but due to the repetitive nature of the text output to the screen, stakeholders with less familiarity were quickly able to deduce what phase was in process.
Clear Victory conditions	Success	This was all fine, with a winning/losing screen appearing accordingly dependant on who won the game, the requirements to win the game was fairly standard: defeat the enemy player. This was understood by all stakeholders who trialled the program.
Game setup assistance	Success	This was especially useful for those who do not know how to set up the board- although conventionally it is done through random distribution of the cards, with the player choosing their starting pieces, this generally takes extensive amount of times, and thus, by doing this for the user when they started up the game saved significant amounts of time, it was found that this was received very well, although there were some suggestions from those with considerable game experience that they would have liked to have set up the map troop distribution.
Accessibility features	Success	With large button options, widespread formatting, it allows users to gain access to accessibility features with ease, however, perhaps this could have been diversified to a larger audience with audio and additional features of personalisation.
Game length	Partial Success	Although the time given was found to be optimal for most of my stakeholders, some struggled to make their moves in time, especially with the transfer phase, where perhaps a button could be added to try to move all but one troops to a location quickly, helpful for larger numbers of troops. A feature to change timer capacity would have helped to diversify the game to a wider audience of differing cognitive ability.
Player aids/ ai inhibition	Success	With features unavailable to the AI such as the card mechanics, players are granted an opportunity to gain a comparative advantage, despite lacking the speed of decision making, to counter this, the timer for the AI player was reduced. This was overall successfully received from stakeholders, however, once again, suggestions were received to create an AI with varying difficulty.

Overall, from providing stakeholders with the program, they were also able to find a few notable limitations in my program, these are documented under limitations, along with additional notes in the following section on improvements to avoid limitations.

Limitations of program

In terms of limitations of my program, general limitations involved with development included:

- Time constraints: Developing and producing a board game takes up a considerable length of time, especially if done as an independent project like this one. Limited time also affects the amount of features or components that can be included, as evaluated and concluded in this overall section.
- Target audience constraints: With a well-defined target audience, affecting the design and marketing of the game. Limitations such as age range, interests, and accessibility can affect the game's appeal and success to the end user, and thus feedback surrounding use of the project can be very varied,
- Game mechanics constraints: Established rules and mechanics, means that amount of innovation or creativity that can be introduced into a new version is somewhat limited, as straying too far from the established mechanics may alienate existing players, while staying too close to the original may not attract new players, or people looking for diversity in mechanics.
- Lack of login & additional personalisation settings: due to the time constraints, in order to thoroughly create a system that makes use of these would take up a significant amount of planning, resources and testing time, with features such as computer capability imposing some level of restrictions on the end-user.
- Scaling back ideas: If we compare success criteria in Analysis to the targets met, a some (often small) changes have had to be made in order to manage and reach the targets provided. To achieve some of these targets, as seen in my failed prototypes, requires attempts at multiple approaches until achieving conditions similar to the desirable goal.
- Fortify for the AI player: Also time constraint related, the game is still progress able to the ai due to its deploy phase, however, this means it is unable to advance attack by at least 1 territory at a time unless if another one of its territories borders a newly-conquered neighbour.

Maintenance issues:

*In light of concluding this project, a few maintenance issues have come to light, how to avoid these issues will be addressed in the section below. Here is a list of some identified bugs that have yet to be fixed (and haven't been fixed yet due to time constraints), it is important to note, however, that these bugs by no means make the game unplayable (aside from perhaps the one marked by *), they only inhibit user experience:*

- *Bonus troops for continent bonus: Despite occupying an entire continent, the player is still not provided the additional troops*
- *Collision reformatting & text centering: The collision system is off centre with the image surface, I believe this may be to due 2 rectangles being created (for the base image, and the selected image), but the wrong one is being used as the surface object for collision.*
- *Load time for setup: Since adding Europe and Africa, the load time of the game takes longer to set up*
- *A random addition of 10 troops per round, despite having not owning an entire continent (after over 8 troops have been added to a player's list*
- *Occasional Zero Error for win probability (occurs every 1 in 30 runs on average) created under exceptional circumstances of near troop deficit, so resulting calculation produces a 0 and thus a theoretical calculation of 0% is impossible*
- *(*) If given a lack of confirmation for transfer, you can edit the troop flow of enemy or neutral territory and give it the player. This only happens 1 in every 20 runs, and will only occur 1 territory at a time before being reset to normal.*
- *Deploy sometimes forgoing the attack phase to fortify -> this occurs once every 9 iterations, after the player deploys their troops, the text for the fortify phase appears immediately after.*

As well as these, there are some (most notable) aspects that have not been fully addressed yet, but were also included in the original plan, these were also subject to time constraints, or infeasibility:

- *Card collection throughout the game: This would have been a very small addition to the game, and allowed collection at a slow rate, better for eventual progression*
- *Save game feature: This is difficult with my current hardware/software capabilities, although it would have been useful for personalisation.*
- *The player is able to login: Due to the integration of this with the main file, and the subsequent lack of knowledge on how to thoroughly mitigate this issue meant that the time required to create this would have been far greater, which is one of the reasons why time was a limiting factor to my final solution.*
- *The win statistics and probability (for the player): Although this was successful for the AI player, to implement for the player was difficult due to my classes and functions being stored on separate files, a management of this issue is detailed in the section below.*

Potential for program improvement & Avoiding limitation

In addressing some of the issues above, here is how they can be managed/ developed/ avoided for future reference:

Addressing the bugs:

- *Bonus troops for continent bonus: Make use of the python function all() to check to see if all elements found inside the continent list can be found inside of that players list, as opposed to trying to find the individual elements in the list(without iteration of traversing this list) - which is the current method being used. So that this would be encompassed in 'if all(territory(x) in p1_territories and territory(y) in p1_territories.....) .*
- *Collision reformatting & text centering: One of the issues is to due with the differential image sizes, formatting and scales, which causes slight variation in appearing on the map. Then amending which rectangle is being collided with would also cause the text to get placed on this surface as opposed to the template rectangle.*
- *Load time for setup: A loading screen that takes a specific amount of time to show before being covered by the main game will help to disguise the temporary freezing of assets as the game sets up.*
- *A random addition of 10 troops per round: This may be to do with the boolean value of the card system, so by enabling the value to false after trading in cards under every circumstance would prevent this.*
- *Occasional Zero Error for win probability: Further input validation and testing for what conditions create this would help mitigate this. By preventing these conditions, it should therefore theoretically never reach that point when the AI cycles through valid attack combinations.*
- *Lack of transfer confirmation and attempt to advance attack: Adding another set of boolean logic gate conditions, so as to ensure that nothing happens whilst territory a and b are selected for transfer, and to ensure that they are not overwritten by the attack confirm part, where you can choose to select or deselect new territories. Similarly, if the phase is finished, these values should also be set to False again, even when that phase comes back again.*
- *Deploy sometimes forgoing the attack phase to fortify: By amending and monitoring the event flow below the event loop, tracing the events of the ai player and its respective timer should pinpoint this specific error, and help produce a solution.*

Addressing scaled back ideas:

- *Card collection throughout: The activation of a new boolean variable for every time a territory is captured by the player, this would then trigger a mechanic that +1's a card from the card list to the player's cards. These could then be recycled back to its previous list once expended.*
- *Save game feature: this can be managed by better research into software and coding languages that are able to transfer and make use of data from different files, along with this, extensive testing would have been useful to keep track of data, for example, a spreadsheet with details such as items in specific lists, identifiable*

features, etc. Also, an abstraction of the data needed to be saved would also save memory requirements, as then it would allow the reusable components to manage the other details.

- *The player is able to login: This issue could have been resolved with an additional screen that loads before the main menu screen, this could then have made use of python libraries such as Tkinter, or been created as an external file to import, a documented trace table would also benefit testing to accompany this, as well as a greater range of knowledge on python libraries.*
- *The win statistics and probability (player): The way this could have been managed could have been through additional testing outside of the file for feasibility, then perhaps integrating at least one class with the main file would have meant that any methods called would be able to interpret the data produced from the win statistics function. Other than that, I could have also added more conditions and included it in the event loop of the main game, so that when both selected were set to true, when the confirm was being hovered by the mouse, it would blit the text to the side, in the format "win probability:{likelihood}% success".*

The finished code

NEA main project.py

```
#NEA main project file
#started 14.11.22 - finished 28.04.23

#imported modules
import random
import pygame
import sys
from classes import *
import time

pygame.init()

#general game variable setup
menu = True
Timer = False
attackphase = True
deployphase = False
fortifyphase = False
finished = False
dist_done = False
p_start_troops = 40
e_start_troops = 40
n_start_troops = 39
troop_minimum = 1
p1_startup = False
timer_active = True
troopval = int()
cards_done = False
feedback = False
card_handin = False
turn_count = 0

#troopcounts for deploy phase
default = 4
```

```
NorthAmericaBonus = 5
SouthAmericaBonus = 3
troop_done = False
n_done = False
first_iteration = 0
p1_turn = True
ai_turn = False
ai_done = False

#variables for attack phase
second_selection = False
selection = False
transfer = True
clicked_territory = None

font_size = 24
font = pygame.font.SysFont('Arial', font_size)
menu_font = pygame.font.SysFont("Calibri",40) #sets the style of the menu font
font_colour = (0,0,0) #sets the font to black
attacker_text = font.render("Attacker is: None", True, (BLACK))
defender_text = font.render("Defender is: None", True, (BLACK))
output = font.render("", True, (BLACK))
attacker_surface = font.render("", True, (0, 0, 0))
defender_surface = font.render("", True, (0, 0, 0))


SCREEN_WIDTH = 1600 #sets the width of the screen
SCREEN_HEIGHT = int(SCREEN_WIDTH * 0.6) #sets the screen height
screen = pygame.display.set_mode(((SCREEN_WIDTH), (SCREEN_HEIGHT)))

timer = pygame.USEREVENT + 1
pygame.time.set_timer(timer,1000) #Fires every 1000 milliseconds/ 1 second
time_remaining = 70
fortify_time_remaining = 30
fortify_timer_active = False
attack_timer_active = True

#fortify text
transfer_between = font.render(f"Troop transfer between:",True,BLACK)
element_zero = font.render(f"None",True,BLACK)
element_one = font.render(f"None",True,BLACK)
fortifying = False

#colour palette declaration - default
RED = (150,6,18)
BLUE = (6,11,150)
BISQUE = (154,255,199)
BLACK = (0,0,0)

#setting a timeframe
```

```
clock = pygame.time.Clock()
FPS = 60

#sets the gui size & caption
screen = pygame.display.set_mode(((SCREEN_WIDTH), (SCREEN_HEIGHT)))
pygame.display.set_caption('Control')

menu_button = pygame.image.load("Images/button.png") #associates the button image with the
 class button
menu_button = pygame.transform.scale(menu_button, (400, 150))

adjustment_button = pygame.transform.scale(menu_button,(50,50))

troop_button = pygame.image.load("Images/neutral.png")
troop_button = pygame.transform.scale(troop_button, (40,40))#scales the image size down of
 the troop button to fit the image

game_button = pygame.image.load("Images/button.png")#associates with the same button image
 as menu_button
game_button = pygame.transform.scale(game_button, (200,75))

elongated_game_buttonv1 = pygame.transform.scale(game_button, (300,75))
elongated_game_buttonv2 = pygame.transform.scale(game_button, (400,75))

title_button = pygame.image.load("Images/Title.svg")
title_button = pygame.transform.scale(title_button,(600,200))

ally_button = pygame.image.load("Images/ally.png")
ally_button = pygame.transform.scale(ally_button,(60,60))

enemy_button = pygame.image.load("Images/enemy.png")
enemy_button = pygame.transform.scale(enemy_button, (40,40))

#map loading
map = pygame.image.load("Images/riskmap.svg")
map = pygame.transform.scale(map,(1600,960))
#loads all the buttons onto the screen, defining their location and text
GameButton = MenuButton(menu_button, 800, 300, "Game")#assigns the MenuButton class to the
 image, position and Text values specific to this instance
SettingsButton = MenuButton(menu_button,800, 500, "Settings")
QuitButton = MenuButton(menu_button,800,700,"Quit")
TitleButton = MenuButton(title_button,800,100,"Control")
#Settings Buttons
ColourBlind = MenuButton(menu_button,800,400,"ColourBlind Mode")
Rules = MenuButton(menu_button,800,400,"Rules")
#loads all adjustment buttons - for transferrence
PLUS = MenuButton(adjustment_button, 10,300,"+")
REMOVE = MenuButton(adjustment_button, 10,400,"-")
END_TURN = GameButtonClass(elongated_game_buttonv1, 500, 900, "END TURN")
```

```
#defining the territories
#North America
Alaska = Territory(troop_button,125,115,"x",troopval,"Alaska")
NWTerritory = Territory(troop_button,125,240,"x",troopval,"North West Territory")
Alberta = Territory(troop_button,200,230,"x",troopval,"Alberta")
Ontario = Territory(troop_button,210,330,"x",troopval,"Ontario")#format y,x
Greenland = Territory(troop_button,100,540,"x",troopval,"Greenland")
WUS = Territory(troop_button,300,240,"x",troopval,"Western US")
Quebec = Territory(troop_button,210,450,"x",troopval,"Quebec")
EUS = Territory(troop_button,340,360,"x",troopval,"Eastern US")
Central_America = Territory(troop_button,430,260,"x",troopval,"Central America")

#South America
Venezuela = Territory(troop_button,495,360,"x",troopval,"Venezuela")
Peru = Territory(troop_button,610,370,"x",troopval,"Peru")
Brazil = Territory(troop_button,600,500,"x",troopval,"Brazil")
Argentina = Territory(troop_button,720,390,"x",troopval,"Argentina")

#Europe
Iceland = Territory(troop_button,160,650,"x",troopval,"Iceland")
GB = Territory(troop_button,290,650,"x",troopval,"Great Britain")
North_Europe = Territory(troop_button,300,760,"x",troopval,"North Europe")
West_Europe = Territory(troop_button,380,680,"x",troopval,"West Europe")
Scandinavia = Territory(troop_button,170,790,"x",troopval,"Scandinavia")
South_Europe = Territory(troop_button,380,790,"x",troopval,"Southern Europe")
East_Europe = Territory(troop_button,260,910,"x",troopval,"East Europe")

#Africa
N_Africa = Territory(troop_button,555,710,"x",troopval,"North Africa")
Egypt = Territory(troop_button,505,850,"x",troopval,"Egypt")
Congo = Territory(troop_button,660,850,"x",troopval,"Congo")
South_Africa = Territory(troop_button,780,850,"x",troopval,"South Africa")
East_Africa = Territory(troop_button,600,910,"x",troopval,"East Africa")
Madagascar = Territory(troop_button,780,1000,"x",troopval,"Madagascar")

#This is for the cards
NW_card_image = pygame.image.load('Images/NWTerritory_Card.png') #creates a surface for th
is type of card
NWTerritory_Card = Cards(NW_card_image,'Cannon','NW')

Ontario_card_image = pygame.image.load('Images/OntarioTerritory_Card.png')
Ontario_Card = Cards(Ontario_card_image,'Cannon','Ontario')

Alberta_card_image = pygame.image.load('Images/AlbertaTerritory_Card.png')
Alberta_Card = Cards(Alberta_card_image,'Soldier','Alberta')

Alaska_card_image = pygame.image.load('Images/AlaskaTerritory_Card.png')
Alaska_Card = Cards(Alaska_card_image,'Cavalry','Alaska')

Quebec_card_image = pygame.image.load('Images/QuebecTerritory_Card.png')
```

```
Quebec_Card = Cards(Quebec_card_image,'Cannon','Quebec')

EUS_card_image = pygame.image.load('Images/EasternUSTerritory_Card.png')
EUS_Card = Cards(EUS_card_image,'Cavalry','EUS')

#neighbouring defining - North America
Alaska.add_neighbour(NWTerritory)
Alaska.add_neighbour(Alberta)
NWTerritory.add_neighbour(Alaska)
NWTerritory.add_neighbour(Alberta)
NWTerritory.add_neighbour(Greenland)
NWTerritory.add_neighbour(Ontario)
Greenland.add_neighbour(NWTerritory)
Greenland.add_neighbour(Ontario)
Greenland.add_neighbour(Quebec)
Alberta.add_neighbour(Alaska)
Alberta.add_neighbour(NWTerritory)
Alberta.add_neighbour(Ontario)
Alberta.add_neighbour(WUS)
Ontario.add_neighbour(NWTerritory)
Ontario.add_neighbour(Greenland)
Ontario.add_neighbour(Alberta)
Ontario.add_neighbour(Quebec)
Ontario.add_neighbour(EUS)
Ontario.add_neighbour(WUS)
Quebec.add_neighbour(Greenland)
Quebec.add_neighbour(Ontario)
Quebec.add_neighbour(EUS)
WUS.add_neighbour(Alberta)
WUS.add_neighbour(Ontario)
WUS.add_neighbour(EUS)
WUS.add_neighbour(Central_America)
EUS.add_neighbour(Quebec)
EUS.add_neighbour(Ontario)
EUS.add_neighbour(WUS)
EUS.add_neighbour(Central_America)
Central_America.add_neighbour(EUS)
Central_America.add_neighbour(WUS)
Central_America.add_neighbour(Venezuela)
Venezuela.add_neighbour(Central_America)
Venezuela.add_neighbour(Peru)
Venezuela.add_neighbour(Brazil)
Peru.add_neighbour(Venezuela)
Peru.add_neighbour(Brazil)
Peru.add_neighbour(Argentina)
Argentina.add_neighbour(Peru)
Argentina.add_neighbour(Brazil)
Brazil.add_neighbour(Peru)
Brazil.add_neighbour(Venezuela)
Brazil.add_neighbour(Argentina)
```

```
Brazil.add_neighbour(N_Africa)
N_Africa.add_neighbour(Egypt)
N_Africa.add_neighbour(Congo)
N_Africa.add_neighbour(West_Europe)
N_Africa.add_neighbour(East_Africa)
Congo.add_neighbour(N_Africa)
Congo.add_neighbour(East_Africa)
Congo.add_neighbour(South_Africa)
South_Africa.add_neighbour(Congo)
South_Africa.add_neighbour(Madagascar)
Madagascar.add_neighbour(South_Africa)
Madagascar.add_neighbour(East_Africa)
East_Africa.add_neighbour(Madagascar)
East_Africa.add_neighbour(Congo)
East_Africa.add_neighbour(Egypt)
East_Africa.add_neighbour(N_Africa)
Egypt.add_neighbour(East_Africa)
Egypt.add_neighbour(N_Africa)
Egypt.add_neighbour(South_Europe)
South_Europe.add_neighbour(Egypt)
South_Europe.add_neighbour(East_Europe)
South_Europe.add_neighbour(North_Europe)
South_Europe.add_neighbour(West_Europe)
West_Europe.add_neighbour(East_Europe)
West_Europe.add_neighbour(GB)
West_Europe.add_neighbour(N_Africa)
West_Europe.add_neighbour(North_Europe)
GB.add_neighbour(Iceland)
GB.add_neighbour(West_Europe)
GB.add_neighbour(Scandinavia)
GB.add_neighbour(North_Europe)
Iceland.add_neighbour(GB)
Iceland.add_neighbour(Scandinavia)
Iceland.add_neighbour(Greenland)
North_Europe.add_neighbour(Scandinavia)
North_Europe.add_neighbour(GB)
North_Europe.add_neighbour(East_Europe)
North_Europe.add_neighbour(South_Europe)
Scandinavia.add_neighbour(GB)
Scandinavia.add_neighbour(Iceland)
Scandinavia.add_neighbour(North_Europe)
Scandinavia.add_neighbour(East_Europe)
East_Europe.add_neighbour(Scandinavia)
East_Europe.add_neighbour(North_Europe)
East_Europe.add_neighbour(South_Europe)

#this function's purpose is to flip between the Game screen and menu screen once this func
tion is activated from the button associated being pressed
def Game():
```

```
    if dist_done == False:
        territorydistribution()
    if p1_startup == False:
        troopdistribution()
    while True:
        global GAMEBUTTON_MOUSE_POS, time_remaining, attackphase, fortifyphase, deployphas
e, timer_active, fortify_time_remaining, fortify_timer_active, DeployTroops, attack_timer_
active
        global timer_rect, available_troops, troop_done, first_iteration, turn_count, terr
itory_a, territory_b, territory_a_name, territory_b_name,  second_selection, selection, at
tacker_text
        global defender_text, output, territorya, territoryb, transfer, clicked_territory,
 element_one, element_zero, fortifying, p1_turn, ai_turn, n_done, ai_done, likelihood
        GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()#location of the mouse pointer is ass
igned to this variable, that is only valid whilst Game() has been activated


        screen.fill(BISQUE)#the screen of the background is filled with bg colour default

        if attack_timer_active == True:
            # Substract time since last frame from remaining time
            time_remaining -= clock.tick(FPS) / 1000
        # Split the remaining time into minutes and seconds. divmod function returns the r
esult of the integer
        # division (minutes) and the modulus of two inputs (seconds).
        # max function ensures timer does not go negative
            minutes, seconds = divmod(max(0, time_remaining), 120)
            timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be u
sed
            timer_text = timer_font.render(f"Time Left: {seconds:.2f}", True, (BLACK))
            timer_rect = pygame.Rect((650, 50), (1, 1))
            screen.blit(timer_text, timer_rect)#put it onto the screen

        elif fortify_timer_active == True:
            fortify_time_remaining -= clock.tick(FPS) / 1000
            minutes, seconds = divmod(max(0, fortify_time_remaining), 120)
            timer_font = pygame.font.SysFont('Consolas',32) #defines the font type to be u
sed
            timer_text = timer_font.render(f"Time Left: {seconds:.2f}", True, (BLACK))
            timer_rect = pygame.Rect((650, 50), (1, 1))
            screen.blit(timer_text, timer_rect)


        elif fortify_timer_active == False and attack_timer_active == False and deployphas
e == True: #if the conditions are in place...
            deploy_font = pygame.font.SysFont('Consolas',24)
            deploy_text = deploy_font.render(f"You have {available_troops} troops remainin
g",True,(BLACK)) #formats the troops available
            screen.blit(deploy_text,timer_rect)#outputs the troops available
```

```
        if attackphase == True and transfer == True:
            screen.blit(attacker_text, (10, 10))
            screen.blit(defender_text, (10,40))
            screen.blit(output,(10,70))
            screen.blit(attacker_surface,(10,600))
            screen.blit(defender_surface,(10,650))


        screen.blit(map,(0,0))#puts the map onto the screen
        GAME_BACK = GameButtonClass(game_button, 200, 900, "BACK")#in format image, y,x,te
xt
        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()

        END_TURN.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        END_TURN.update()

        TRADE_IN = GameButtonClass(elongated_game_buttonv2, 900, 900, "CARD TRADE IN")
        TRADE_IN.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        TRADE_IN.update()

        CONFIRM_SELECTION = GameButtonClass(elongated_game_buttonv2, 1350,900, "CONFIRM AT
TACK")
        CONFIRM_SELECTION.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        CONFIRM_SELECTION.update()

        if ai_turn == True and attackphase == True:#if it is the ai's turn and the attack
phase has commenced
            if time_remaining != 0:#as long as the time is not 0
                random_territory = random.choice(enemy_territories)#picks a random territo
ry
                territorya = random_territory#assigning it to territorya
                if territorya.troopval == 1:
                    territorya = None
                    random_territory = None
                    random_territory = random.choice(enemy_territories)#picks a random ter
ritory
                    territorya = random_territory
                    print(territorya.name, territorya.troopval)
                elif territorya.troopval <= 0:
                    territorya.troopval = 1
                else:
                    print(territorya.name,territorya.troopval)#prints out the name (for te
st purposes) - to keep track of terminal responses
                random_territory = None#resets to None, so will not endlessly iterate
                random_territory = random.choice(p1_territories+neutral_territories) #pick
s a new territory
                territoryb = random_territory#assigns this to territory b
                print(territoryb.name, territoryb.troopval)#print this for testing purpose
s
```

```
                random_territory = None
                if territorya.is_neighbour(territoryb):#if they are neighbours
                    win_probability()#calculate the win probability
                    print(likelihood)
                    if likelihood > 30 and territorya.troopval != 1:#if the win probility
is greater tha 30%
                        #print(likelihood)
                        diceroll()#make the move
                    else:
                        time.sleep(0.05) #time.sleep in place to put limitations on ai
                else:
                    time.sleep(0.05)

        if transfer == False: #setting all the text values...
            transfer_font = pygame.font.SysFont('Arial',12)
            transfer_text = transfer_font.render(f"{territory_a_name}:{territorya.troopval
}",True,BLACK)
            instruction_text = transfer_font.render("Press '.' to confirm choice",True,BLA
CK)
            instruction_text_2 = transfer_font.render(f"Press '+' to transfer troops to {t
erritory_b_name}",True,BLACK)
            instruction_text_3 = transfer_font.render(f"Press '-
' to transfer troops to {territory_a_name}",True,BLACK)
            transfer_text_2 = transfer_font.render(f"{territory_b_name}:{territoryb.troopv
al}",True,BLACK)

        if attackphase == True and transfer == False:
            screen.blit(attacker_text, (10, 10))
            screen.blit(defender_text, (10,40))
            screen.blit(transfer_text,(10,750))
            screen.blit(instruction_text,(10,600))
            screen.blit(instruction_text_2,(10,650))
            screen.blit(instruction_text_3,(10,700))
            screen.blit(transfer_text_2,(10,800))
        PLUS = MenuButton(adjustment_button, 30,450,"+")
        REMOVE = MenuButton(adjustment_button, 30,500,"-")
        TRANSFER_FINALIZE = MenuButton(adjustment_button, 30,550,".")

        REMOVE.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        REMOVE.update()
        PLUS.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        PLUS.update()
        TRANSFER_FINALIZE.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        TRANSFER_FINALIZE.update()

        if fortifyphase and fortifying == True:
            transfer_font = pygame.font.SysFont('Arial',12)
            if len(chosen_list) != 0:
                instruction_text_setb = transfer_font.render("Press '.' to confirm/reset",
True,BLACK)
```

```
                instruction_text_set_b_2 = transfer_font.render(f"Press '+' to transfer tr
oops to {chosen_list[0].name}",True,BLACK)
                instruction_text_set_b_3 = transfer_font.render(f"Press '-
' to transfer troops to {chosen_list[1].name}",True,BLACK)
            else:
                instruction_text_setb = transfer_font.render("Press '.' to confirm/reset",
True,BLACK)
                instruction_text_set_b_2 = transfer_font.render(f"Press '+' to transfer tr
oops to Not selected",True,BLACK)
                instruction_text_set_b_3 = transfer_font.render(f"Press '-
' to transfer troops to Not selected",True,BLACK)

            screen.blit(instruction_text_setb,(10,600))
            screen.blit(instruction_text_set_b_2,(10,650))
            screen.blit(instruction_text_set_b_3,(10,700))


        if fortifyphase == True:
            screen.blit(transfer_between,(10,10))
            screen.blit(element_zero,(10,40))
            screen.blit(element_one,(10,70))
            #screen.blit(not_valid(10,100))

        for territory in territories:
            if territory in p1_territories:
                screen.blit(ally_button, (territory.x_pos, territory.y_pos))
            elif territory in enemy_territories:
                screen.blit(enemy_button, (territory.x_pos, territory.y_pos))
            elif territory in neutral_territories:
                screen.blit(troop_button, (territory.x_pos, territory.y_pos))
            screen.blit(territory.trooptext, (territory.x_pos, territory.y_pos))

        for event in pygame.event.get():
            if len(p1_territories) == 0 or len(enemy_territories) == 0:
                win_screen()
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
                    main_menu()
                elif END_TURN.checkForInput(GAMEBUTTON_MOUSE_POS) and p1_turn == True:
                    if attack_timer_active == True:
                        time_remaining = 0
                    elif fortify_timer_active == True:
                        fortify_time_remaining = 0
                if deployphase == True:
                    if TRADE_IN.checkForInput(GAMEBUTTON_MOUSE_POS):
                        trade_in()
                    for territory in territories:
```

```
                        if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #if the terr
itory is clicked by the mouse
                            chosen_list.append(territory) #append it to a new list
                            for territory in chosen_list: #for that one territory
                                if territory in p1_territories and available_troops!=0: #v
alidates which territory the player can add to
                                    territory.troopval += 1
                                    available_troops -= 1
                                    territory.update_troop_text()
                                    chosen_list.remove(territory)
                                else:
                                    print("You cannot access this at this time")
                if attackphase == True:#if the attack phase in in progress
                    if p1_turn == True:
                        if CONFIRM_SELECTION.checkForInput(GAMEBUTTON_MOUSE_POS): #if the
confirm selection has been clicked
                            if selection == True and second_selection == True and transfer
 == True: #as long as both values have been submitted
                                if territorya.is_neighbour(territoryb): #if the territory
a is neighbours with the selected enemy territory...
                                    output = font.render("Attack Confirmed",True,(BLACK))
#change the text to attack confirmed
                                    diceroll()
                                else:
                                    output = font.render("Error: Territories aren't neighb
ours.",True,(BLACK)) #otherwise tell the user that they cannot do this
                                    #resets the territories selected
                                    territory_a.remove(territorya)
                                    territory_b.remove(territoryb)
                                    #updates the text to notify the user
                                    attacker_text = font.render("Attacker is: None", True,
 (BLACK))
                                    defender_text = font.render("Defender is: None", True,
 (BLACK))
                                    #prevents the cycle from erroring
                                    selection = False
                                    second_selection = False
                        if TRANSFER_FINALIZE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the
user wishes to finalize movement...
                            if transfer == False: #if transfer is active
                                territorya = None#set territory a to none, so that it cann
ot be altered now
                                selection = False #set selection to False, so as to not er
ror the confirm attack button if pressed
                                territoryb = None #set territory b to none
                                second_selection = False #same as selection
                                transfer = True #set to true, so that these buttons can no
 longer be used until next transfer
                        if PLUS.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '+' is presse
d...
```

```
                            if transfer == False: #if the troops are in the prcess of bein
g moved..
                                if territoryb.troopval >= 1 and territorya.troopval > 1: #
as long as at least 1 troop on territory being moved from..
                                    territoryb.troopval += 1#add 1 to the other
                                    territoryb.update_troop_text()#update to screen
                                    territorya.troopval -= 1 #remove from current
                                    territorya.update_troop_text()
                        elif REMOVE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '-
' is pressed...
                            if transfer == False: #if the troops are in the process of bei
ng moved
                                if territorya.troopval >= 1 and territoryb.troopval > 1: #
as long as there is at least 1 troop on the territory it is being moved from...
                                    territoryb.troopval -= 1 #remove one
                                    territoryb.update_troop_text() #update the value
                                    territorya.troopval += 1 #add to the other territory
                                    territorya.update_troop_text()
                        for territory in p1_territories:#if the territory is player 1's
                            if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and get
s clicked....
                                if len(territory_a) != 1: #if the length of the territory
is not 1
                                    territory_a.append(territory) #add it to the list
                                    for territory in territory_a: #for the listed territor
y
                                        territory_a_name = territory.name #log the name (f
or testing purposes)
                                        territorya = territory #logs the territory (for co
mparison values)
                                        territoryatroops = territory.troopval #logs the tr
oop value
                                        if territoryatroops >= 2:
                                            attacker_text = font.render(f"Attacker is: {te
rritory_a_name}", True, (BLACK))
                                            selection = True
                                        else:
                                            attacker_text = font.render(f"Attacker is: Non
e, please select another, as {territory_a_name} does not have enough troops", True, (BLACK
))
                                            territory_a.remove(territorya)
                                            selection = False
                                else:
                                    attacker_text = font.render("Attacker is: None", True,
 (BLACK))
                                    territory_a.remove(territorya) #if it gets to larger t
han 1, deselect
                                    selection = False
```

```
                        for territory in enemy_territories+neutral_territories: #if the te
rritory is not te player's
                            if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS): #and it
gets clicked
                                if len(territory_b) != 1: #check if empty
                                    territory_b.append(territory) #add to the list
                                    for territory in territory_b:
                                        territory_b_name = territory.name #log the name (f
or test purposes)
                                        territoryb = territory #logs the name
                                        defender_text = font.render(f"Defender is: {territ
ory_b_name}", True, (BLACK))
                                        second_selection = True
                                else:
                                    defender_text = font.render("Defender is: None", True,
 (BLACK))
                                    territory_b.remove(territoryb)#if it is 1 then remove
it - so deselects
                                    second_selection = False
                if fortifyphase == True:
                    for territory in p1_territories:
                        if territory.rect.collidepoint(GAMEBUTTON_MOUSE_POS):
                            clicked_territory = territory #the collided territory becomes
selected
                            if len(chosen_list) == 0: #if the list is empty
                                chosen_list.append(clicked_territory) #add the selected te
rritoriy to the list
                                element_zero = font.render(f"{chosen_list[0].name}",True,B
LACK)
                            elif len(chosen_list) == 1: #if the list already has an object
                                chosen_list.append(clicked_territory) #add the next
                                if chosen_list[1] == chosen_list[0]: #if both indexed elem
ents are the same
                                    chosen_list.remove(clicked_territory) #remove from the
 list
                                elif chosen_list[0].is_neighbour(chosen_list[1]): #validat
e if they are neighbours
                                    element_one = font.render(f"{chosen_list[1].name}",Tru
e,BLACK)
                                else: #if they aren't neighbours, remove them
                                    chosen_list.remove(clicked_territory)
                            elif len(chosen_list) == 2:#if the list is already full
                                chosen_list[0] = chosen_list[1] #element 0 is now element
1
                                chosen_list.remove(chosen_list[1]) #remove element 1, so t
hat the length is now 1
                                element_one = font.render(f"None",True,BLACK)
                        if len(chosen_list) == 2:
                            fortifying = True
```

```
                    if TRANSFER_FINALIZE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the user
 wishes to finalize movement...
                        if fortifying == True: #if transfer is active
                            chosen_list.remove(chosen_list[1])#set first selected none, so
 no alteration
                            chosen_list.remove(chosen_list[0]) #set territory b to none
                            fortifying = False
                            element_zero = font.render(f"None",True,BLACK)
                            element_one = font.render(f"None",True,BLACK)
                    if PLUS.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '+' is pressed...
                        if fortifying == True: #if the troops are in the prcess of being m
oved..
                            if chosen_list[0].troopval >= 1 and chosen_list[1].troopval >
1: #as long as at least 1 troop on territory being moved from..
                                chosen_list[1].troopval -= 1#add 1 to the other
                                chosen_list[1].update_troop_text()#update to screen
                                chosen_list[0].troopval += 1 #remove from current
                                chosen_list[0].update_troop_text()
                    elif REMOVE.checkForInput(GAMEBUTTON_MOUSE_POS): #if the '-
' is pressed...
                        if fortifying == True: #if the troops are in the prcess of being m
oved..
                            if chosen_list[1].troopval >= 1 and chosen_list[0].troopval >
1: #as long as there is at least 1 troop on the territory it is being moved from...
                                chosen_list[0].troopval -= 1 #remove one
                                chosen_list[0].update_troop_text() #update the value
                                chosen_list[1].troopval += 1 #add to the other territory
                                chosen_list[1].update_troop_text()

        if attackphase == True and first_iteration<1 and p1_turn == True:
            attack_timer_active = True
            deployphase = False
            #time_remaining = 10
            if time_remaining <= 0:
                fortifyphase = True
                timer_active = False
                attackphase = False
                attack_timer_active = False
                fortify_time_remaining = 30
        if attackphase == True and first_iteration>=1 and p1_turn == True:
            attack_timer_active = True
            deployphase = False
            if time_remaining <= 0:
                fortifyphase = True
                timer_active = False
                attackphase = False
                attack_timer_active = False
                fortify_time_remaining = 30
        if attackphase == True and first_iteration>=1 and ai_turn == True:
            attack_timer_active = True
```

```
            deployphase = False
            if time_remaining <= 0:
                fortifyphase = True
                timer_active = False
                attackphase = False
                attack_timer_active = False
                fortify_time_remaining = 30
        elif fortifyphase == True and first_iteration<1 and p1_turn == True: #continue as
normal
            fortify_timer_active = True
            if fortify_time_remaining <= 0:
                fortifyphase = False
                fortify_timer_active = False
                deployphase = True
        elif fortifyphase == True and first_iteration>=1 and p1_turn == True: #this will a
llow to end the turn
            fortify_timer_active = True
            if fortify_time_remaining <= 0:
                fortifyphase = False
                fortify_timer_active = False
                deployphase = True
                troop_done = False
                n_done = False
                turn_count+=1
        elif fortifyphase == True and first_iteration>=1 and ai_turn == True: #this will a
llow to end the turn
            fortify_timer_active = True
            if fortify_time_remaining <= 0:
                fortifyphase = False
                fortify_timer_active = False
                deployphase = True
                troop_done = False
                ai_done = False
                turn_count +=1 #change turns once this is over.
        elif deployphase == True and p1_turn == True:
            DeployTroops()#calls the function to calculate p1's troops
            NeutralDeploy()#calls the function for the neutral deployment
            if available_troops == 0 and first_iteration == 0:
                first_iteration += 1 #the first iteration is used to create a unique set o
f events from the rest of the game
                attackphase = True
                deployphase = False
                time_remaining = 70
            if available_troops == 0 and first_iteration > 0:#this will be followed for th
e rest of the game
                #so that phase system goes deploy < attack < fortify < end turn
                attackphase = True
                deployphase = False
                time_remaining = 70
        elif deployphase == True and ai_turn == True:
```

```
            DeployTroops()#calls function for troop calc
            NeutralDeploy()#calls function for neutral deployment
            AIDeploy()#calls the function to deploy ai's troops based off this calc

        if turn_count % 2 == 0: #if the turn count(automatically set to 0) is divisible by
 2, and the remainder is 0, then it is
            #player 1's turn, if not, it is ai turn
            p1_turn = True
            ai_turn = False
        else: #if there is 1 left over, it is ai's turn
            p1_turn = False
            ai_turn = True

        pygame.display.update()

territories = [
    Alaska, NWTerritory, Alberta, Ontario, Quebec, Greenland, WUS, EUS, Central_America, V
enezuela, Peru, Brazil,
    Argentina, Iceland, GB, North_Europe, West_Europe, Scandinavia, South_Europe, East_Eur
ope, N_Africa, Egypt, Congo
    ,South_Africa, East_Africa, Madagascar]
neutral_territories = [
    Alaska, NWTerritory, Alberta, Ontario, Quebec, Greenland, WUS, EUS, Central_America, V
enezuela, Peru, Brazil,
    Argentina, Iceland, GB, North_Europe, West_Europe, Scandinavia, South_Europe, East_Eur
ope, N_Africa, Egypt, Congo
    ,South_Africa, East_Africa, Madagascar]
p1_territories = []
enemy_territories = []

chosen_list = []
territory_a = []
territory_b = []

troops = []

#defining the north american countries
NAterritories = [Alaska, NWTerritory, Alberta, Ontario, Quebec, Greenland, WUS, EUS, Centr
al_America]
SAterritories = [Venezuela, Peru, Brazil, Argentina, ]
Europe = [Iceland, GB, North_Europe, West_Europe, Scandinavia, South_Europe, East_Europe]
Africa = [N_Africa, Egypt, Congo,South_Africa, East_Africa, Madagascar]

def win_screen():
    if len(p1_territories) > len(enemy_territories):
        winningalias = "Player"
    else:
        winningalias = "AI Player"
    while True:
        screen.fill(BISQUE)
```

```
        winner_font = pygame.font.SysFont('Arial',72)
        win_text = winner_font.render(f"{winningalias} Wins!", True,BLACK)
        win_subtext= winner_font.render("Close screen to exit",True,BLACK)

        screen.blit(win_text,(550,400))
        screen.blit(win_subtext,(450,800))


        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        pygame.display.update()

#creating the territory randomiser
def territorydistribution():
    global dist_done
    dist_done = True
    territorycount = 0
    if len(neutral_territories) > 0:
        while territorycount !=8:
            random_territory = random.choice(neutral_territories) #for the random territor
ies of the ally
            neutral_territories.remove(random_territory)
            p1_territories.append(random_territory)
            random_enemy = random.choice(neutral_territories) #for the random territories
of the enemy
            neutral_territories.remove(random_enemy)
            enemy_territories.append(random_enemy)
            territorycount += 1

def troopdistribution():
    #this will be used to distibute the troops randomly
    global p1_startup
    p1_startup = True
    global p_start_troops, e_start_troops, n_start_troops #Defines the variables for the n
umber of starting troops per player

    p1_territory_troop_vals = [0] * len(p1_territories) #Creates a list of the troopvals f
oe each territory owned by p1
    enemy_territory_troop_vals = [0] * len(enemy_territories)

    while sum(p1_territory_troop_vals) != p_start_troops: #loop until total troopval is eq
ual to starting number (p1)
        for idx in range(len(p1_territory_troop_vals)): #loops through each p1 territory,
randomly assinging troopvals
            p1_territory_troop_vals[idx] = random.randint(1, p_start_troops-
len(p1_territories)+1)
```

```
    for idx, territory in enumerate(p1_territories):#Assign troopval to p1, update text on
 screen
        territory.troopval = p1_territory_troop_vals[idx]
        territory.update_troop_text()

    while sum(enemy_territory_troop_vals) != e_start_troops:#Loop until total troopval is
equal to starting number (enemy)
        for idx in range(len(enemy_territory_troop_vals)):#Loops through each enemy territ
ory, randomly assinging troopvals
            enemy_territory_troop_vals[idx] = random.randint(1, e_start_troops-
len(enemy_territories)+1)

    for idx, territory in enumerate(enemy_territories):#Assign troopval to enemy, update t
ext on screen
        territory.troopval = enemy_territory_troop_vals[idx]
        territory.update_troop_text()

    neutral_territory_troop_vals = [n_start_troops // len(neutral_territories)] * len(neut
ral_territories)
    #the line above creates a troop value list for each neutral territory where troops are
 equally distributed

    for idx, territory in enumerate(neutral_territories):#Assign troopval to neutral terri
tories, updating the text on screen
            territory.troopval = neutral_territory_troop_vals[idx]
            territory.update_troop_text()

def Settings():
    global BISQUE, ally_button, enemy_button
    while True:
        SETTINGS_MOUSE_POS = pygame.mouse.get_pos()

        screen.fill(BISQUE) #overlays the screen

        TitleButton = MenuButton(title_button,800,100,"Control") #title button

        TitleButton.update()#ensures that it appears on the screen

        SETTINGS_BACK = GameButtonClass(game_button, 100, 800, "BACK") #so the user can ju
mp back and forth between screens
        SETTINGS_BACK.HighlightByMouseHover(SETTINGS_MOUSE_POS)
        SETTINGS_BACK.update()

        Tritanopia = MenuButton(menu_button,800,300,"Tritanopia") #tritanopia button
        Tritanopia.HighlightByMouseHover(SETTINGS_MOUSE_POS)
        Tritanopia.update()

        Deuteranopia = MenuButton(menu_button,300,300,"Deuteranopia") #deuteranopia button
        Deuteranopia.HighlightByMouseHover(SETTINGS_MOUSE_POS)
```

```
        Deuteranopia.update()

        Protanopia = MenuButton(menu_button,1300,300,"Protanopia") #protanopia button
        Protanopia.HighlightByMouseHover(SETTINGS_MOUSE_POS)
        Protanopia.update()

        Default = MenuButton(menu_button,800,500,"Default") #default colour settings
        Default.HighlightByMouseHover(SETTINGS_MOUSE_POS)
        Default.update()

        Rules = MenuButton(menu_button,800,700,"Rules") #rules button
        Rules.HighlightByMouseHover(SETTINGS_MOUSE_POS)
        Rules.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if SETTINGS_BACK.checkForInput(SETTINGS_MOUSE_POS):
                    main_menu()
                if Tritanopia.checkForInput(SETTINGS_MOUSE_POS): #if the tritanopia button
 is clicked..
                    BISQUE = (159,223,225) #change the background colour
                    ally_button = pygame.image.load("Images/trit_ally.png")
                    ally_button = pygame.transform.scale(ally_button,(60,60)) #change the
colour scheme for the ally
                    enemy_button = pygame.image.load("Images/trit_enemy.png")
                    enemy_button = pygame.transform.scale(enemy_button, (40,40)) #change t
he colour scheme for ai
                if Deuteranopia.checkForInput(SETTINGS_MOUSE_POS):#if the deuteranopia but
ton is clicked..
                    BISQUE = (191,184,215)#change the background colour
                    ally_button = pygame.image.load("Images/deutro_test.png")
                    ally_button = pygame.transform.scale(ally_button,(60,60))#change the c
olour scheme for the ally
                    enemy_button = pygame.image.load("Images/deutro_enemy.png")
                    enemy_button = pygame.transform.scale(enemy_button, (40,40))#change th
e colour scheme for ai
                if Protanopia.checkForInput(SETTINGS_MOUSE_POS):
                    BISQUE = (197,198,212)#change the background colour
                    ally_button = pygame.image.load("Images/protan_test.png")
                    ally_button = pygame.transform.scale(ally_button,(60,60))#change the c
olour scheme for the ally
                    enemy_button = pygame.image.load("Images/protan_enemy.png")
                    enemy_button = pygame.transform.scale(enemy_button, (40,40))#change th
e colour scheme for ai
                if Default.checkForInput(SETTINGS_MOUSE_POS):
                    BISQUE = (154,255,199)#change the background colour
                    ally_button = pygame.image.load("Images/ally.png")
```

```
                    ally_button = pygame.transform.scale(ally_button,(60,60))#change the c
olour scheme for the ally
                    enemy_button = pygame.image.load("Images/enemy.png")
                    enemy_button = pygame.transform.scale(enemy_button, (40,40))#change th
e colour scheme for ai
                if Rules.checkForInput(SETTINGS_MOUSE_POS):
                    pass

        pygame.display.update()

def main_menu():#main menu function
    while menu == True:#when the menu is active
        screen.fill(BISQUE)#set background

        MENU_MOUSE_POS = pygame.mouse.get_pos()#the menu pointer is now the mouse pointer

        GameButton = MenuButton(menu_button, 800, 300, "Game")

        SettingsButton = MenuButton(menu_button, 800, 500, "Settings")

        QuitButton = MenuButton(menu_button, 800, 700, "Exit")
        TitleButton = MenuButton(title_button,800,100,"Control") #sets the title button ou
t onto the screen

        for button in [GameButton, SettingsButton, QuitButton, TitleButton]:
            button.HighlightByMouseHover(MENU_MOUSE_POS)
            button.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GameButton.checkForInput(MENU_MOUSE_POS):
                    Game()
                if SettingsButton.checkForInput(MENU_MOUSE_POS):
                    Settings()
                if QuitButton.checkForInput(MENU_MOUSE_POS):
                    pygame.quit()
                    sys.exit()

        pygame.display.update()

def DeployTroops():
    global default, NorthAmericaBonus, available_troops,deployphase,attackphase,attack_tim
er_active,card_handin,troop_done,ai_available_troops
    #for territories in NAterritories:
    if p1_turn == True:
        if troop_done == False: #if it not unactivated
            if card_handin == False: #and there is no additionals
```

```
                if  p1_territories in NAterritories:#if the 1st player's territories canno
t be found in the NA list as a whole
                    available_troops = default + NorthAmericaBonus #make the total 8
                    troop_done = True#make done
                    return available_troops
                if p1_territories in SAterritories:
                    available_troops = default + SouthAmericaBonus
                else:
                    NorthAmericaBonus = 0
                    SouthAmericaBonus = 0
                    available_troops = default + NorthAmericaBonus + SouthAmericaBonus#mak
e total 3
                    troop_done = True
                    return available_troops
            else:
                available_troops = available_troops + 10 #add 10 to the total once.
                troop_done = True
                return available_troops
    elif ai_turn == True:
        if troop_done == False: #if it not unactivated
            if  enemy_territories in NAterritories:#if the 1st player's territories cannot
 be found in the NA list as a whole
                ai_available_troops = default + NorthAmericaBonus #make the total 8
                troop_done = True#make done
                return ai_available_troops
            if enemy_territories in SAterritories:
                ai_available_troops = default + SouthAmericaBonus
                return ai_available_troops
            else:
                NorthAmericaBonus = 0
                SouthAmericaBonus = 0
                ai_available_troops = default + NorthAmericaBonus + SouthAmericaBonus#make
 total 3
                troop_done = True
                return ai_available_troops

available_troops = DeployTroops()
ai_available_troops = DeployTroops()

def trade_in():
    global GAMEBUTTON_MOUSE_POS, main_queue,feedback,available_troops,troop_done, card_han
din, first_obj
    if cards_done == False:
        CardDistribution()
    while True:
        GAMEBUTTON_MOUSE_POS = pygame.mouse.get_pos()

        screen.fill (BISQUE)
        main_queue.draw(screen)
        priority_queue.draw(screen)
```

```
        GAME_BACK = GameButtonClass(game_button, 200, 900, "BACK")#in format image, y,x,te
xt


        GAME_BACK.HighlightByMouseHover(GAMEBUTTON_MOUSE_POS)
        GAME_BACK.update()

        if feedback == True:#if something not valid triggered...
            response_rect = pygame.Rect((650, 50), (1, 1))
            response_font = pygame.font.SysFont('Sans Serif',32)
            response_text = response_font.render("That is not valid",True,(125,125,125))#c
reate the text surface
            screen.blit(response_text,response_rect)#put onto the screen
            time.sleep(1)
            feedback = False
            if len(priority_queue.slots) == 0:#if the queue is reset
                feedback = False#message disappears
        else:
            feedback = False

        if len(priority_queue.slots) == 3:#if it has reached capacity
            priority_queue.type = None
            for obj in priority_queue.slots.copy():#all the objects in the priority queue.
..
                if obj in priority_queue.slots:
                    priority_queue.remove(obj)#are individually deleted...
                    cards.append(obj)#these cards are then added back to the list for late
r randomisation
                    card_handin = True
                    troop_done = False


        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if GAME_BACK.checkForInput(GAMEBUTTON_MOUSE_POS):
                    Game()
                clicked_obj = None
                # Check if an object in the main queue was clicked
                for obj in main_queue.slots:
                    if obj.check_for_input(GAMEBUTTON_MOUSE_POS):
                        clicked_obj = obj
                        break
                # Check if an object in the priority queue was clicked
                for obj in priority_queue.slots:
                    if obj.check_for_input(GAMEBUTTON_MOUSE_POS):
                        clicked_obj = obj
                        break
```

```
                if clicked_obj:
                    if clicked_obj in main_queue.slots:
                        if len (priority_queue.slots)!=3 and len(priority_queue.slots)>=1:
 #if the length ranges from 1-3..
                            if clicked_obj.type == first_obj.type:#if the type is the same
                                main_queue.remove(clicked_obj) #remove from the main queue
                                priority_queue.add(clicked_obj)#add to priority
                            else:
                                feedback = True
                        #this is the setup statement, as it always starts with 0 in.
                        elif len(priority_queue.slots) == 0:#if there is nothing in the pr
iority queue, reset
                            priority_queue.type = None
                            first_obj = clicked_obj#starts off the priority list type
                            main_queue.remove(clicked_obj)
                            priority_queue.add(clicked_obj)
                            priority_queue.type = first_obj.type
                    elif clicked_obj in priority_queue.slots:#can remove anything from pri
ority queue
                        priority_queue.remove(clicked_obj)
                        main_queue.add(clicked_obj)




            pygame.display.flip()
        pygame.display.update()

main_queue = Queue(num_slots_main, queue_position) #defines the queue with 15 slots, and t
heir positions
priority_queue = Queue(num_slots_priority, priority_queue_position) #3 slots and their loc
ation

def CardDistribution():
    global cards
    cards = [NWTerritory_Card,Ontario_Card,Alberta_Card,Alaska_Card, Quebec_Card,EUS_Card]
    p1cards = []
    global cards_done, main_queue
    cards_done = True
    cardcount = 0
    if len(cards) > 0:
        while cardcount !=6: #set this to 1, then every time the turn begins again,
            #set the cardcount back down to 0, so that it can enact this loop again
            random_card = random.choice(cards) #for the random cards of the ally
            cards.remove(random_card)
            p1cards.append(random_card)
            main_queue.add(random_card)
            cardcount += 1
```

```
def NeutralDeploy(): #adds half of what the player recieves, randomly adding it
    global available_troops, n_done
    neutral_available_troops = available_troops // 2
    while neutral_available_troops > 0 and n_done == False:#while the requirements arent m
et...
        random_territory = random.choice(neutral_territories) #pick a random territory
        chosen_list.append(random_territory)#add it to the current edited territory
        random_territory.troopval += 1 #add to the troopvalue
        neutral_available_troops -= 1#decrement the available troops by 1
        random_territory.update_troop_text() #update the text on screen
        chosen_list.remove(random_territory)#remove it from the list
        random_territory = None#set to none, so it wont duplicate itself in the list
        if neutral_available_troops == 0:#if it reaches 0, disable the while loop
            n_done = True

def AIDeploy(): #adds half of what the player recieves, randomly adding it
    global ai_available_troops, ai_done,attackphase,deployphase,time_remaining
    while ai_available_troops > 0 and ai_done == False:#while the requirements arent met..
.
        random_territory = random.choice(enemy_territories) #pick a random territory
        chosen_list.append(random_territory)#add it to the current edited territory
        random_territory.troopval += 1 #add to the troopvalue
        ai_available_troops -= 1#decrement the available troops by 1
        random_territory.update_troop_text() #update the text on screen
        chosen_list.remove(random_territory)#remove it from the list
        random_territory = None#set to none, so it wont duplicate itself in the list
        if ai_available_troops == 0:#if it reaches 0, disable the while loop
            attackphase = True
            deployphase = False
            time_remaining = 35 #shortens the time - makes it fair for the player
            ai_done = True

def diceroll(): #function for the start of the troop decrements
    global attacker_surface, defender_surface, attacker_text, defender_text, territoryb, s
election, second_selection, territorya, transfer
    # Determine how many dice to roll for the attacker and defender

    min = 1

    if territorya.troopval == 2: #if the player has only 2 troops available
        attacker_dice_count = 1 #they only get one attack dice
    elif territorya.troopval == 3: #if there are 3 troops, give them 2 dice
        attacker_dice_count = 2
    else:
        attacker_dice_count = 3#otherwise they get 3 dice

    if territoryb.troopval == 1:#if the defender has only 1 troop, they are entitled to 1
dice
        defender_dice_count = 1
    else:
```

```
        defender_dice_count = 2#whereas anything more than 1 allows them to have 2

    #picks a random value from 1 to 6, for every time each playr have a dice
    attacker_rolls = sorted([random.randint(1, 6) for i in range(attacker_dice_count)],rev
erse=True)
    #reverse = True ensures that it is sorted into descending order so that its first elem
ent is the highest
    defender_rolls = sorted([random.randint(1, 6) for j in range(defender_dice_count)],rev
erse=True)

    # Render the results as text
    attacker_dice = f"Attacker: {attacker_rolls}"
    defender_dice = f"Defender: {defender_rolls}"

    attacker_surface = font.render(attacker_dice, True, (0, 0, 0))
    defender_surface = font.render(defender_dice, True, (0, 0, 0))

    if attacker_rolls[0]>defender_rolls[0]:#if the attacker rolls more than the defender
        territoryb.troopval -=1 #attacker wins, so defender loses a troop
        territoryb.update_troop_text()
    elif attacker_rolls[0] == defender_rolls[0]: #if they are equal, only the attacker los
es a troop
        territorya.troopval -= 1
        territorya.update_troop_text()
    else:
        territorya.troopval -= 1 #if the defender is higher, attacker loses a troop
        territorya.update_troop_text()

    if attacker_dice_count >= 2 and defender_dice_count == 2:
        if attacker_rolls[1]>defender_rolls[1]:#if the attacker's 2nd highest is greater t
han defender's 2nd highest
            territoryb.troopval -=1 #attacker wins, so defender loses a troop
            territoryb.update_troop_text()
        elif attacker_rolls[1] == defender_rolls[1]: #if they are equal, only the attacker
 loses a troop
            territorya.troopval -= 1
            territorya.update_troop_text()
        else:
            territorya.troopval -= 1 #if the defender is higher, attacker loses a troop
            territorya.update_troop_text()

    if p1_turn == True:
        if territorya.troopval == 1: #if the attacker only has 1 troop left, do not let th
em attack
            territory_a.remove(territorya) #thus remove it from the selected
            attacker_text = font.render("Attacker is: None", True, (BLACK)) #update the di
splay to show this
            territorya = None
            selection = False
```

```
            defender_text = font.render("Defender is: None", True, (BLACK)) #update the te
xt on the display
            territory_b.remove(territoryb) #deselect the territory so thar it can no longe
r be attacked by player
            territoryb = None
            second_selection = False#so that territories can be reselected
        elif territoryb.troopval == 0:#if the defender runs out of troops
            territoryb.ChangeAllegience() #change the allegience
            territoryb.troopval += 1 #ensures that there is never 0 on that territory - wi
ll later be changed on the input mechanics
            territoryb.update_troop_text()
            p1_territories.append(territoryb) #ensures that now the player owns this
            if territoryb in enemy_territories:
                enemy_territories.remove(territoryb) #ensures that the enemy no longer own
s this
                territory_b.remove(territoryb) #deselect the territory so thar it can no l
onger be attacked by player
                defender_text = font.render("Defender is: None", True, (BLACK)) #update th
e text on the display
                territory_a.remove(territorya)
                attacker_text = font.render("Attacker is: None", True, (BLACK))
                transfer = False
            else:
                neutral_territories.remove(territoryb)
                territory_b.remove(territoryb) #deselect the territory so thar it can no l
onger be attacked by player
                defender_text = font.render("Defender is: None", True, (BLACK)) #update th
e text on the display
                territory_a.remove(territorya)
                attacker_text = font.render("Attacker is: None", True, (BLACK))
                transfer = False
    elif ai_turn == True:
        if territorya.troopval == 1: #if the attacker only has 1 troop left, do not let th
em attack
            attacker_text = font.render("Attacker is: None", True, (BLACK)) #update the di
splay to show this
            territorya = None
            selection = False
            defender_text = font.render("Defender is: None", True, (BLACK)) #update the te
xt on the display
            territoryb = None
            second_selection = False#so that territories can be reselected
        elif territoryb.troopval == 0:#if the defender runs out of troops
            territoryb.ChangeAllegience() #change the allegience
            territoryb.troopval += 1 #ensures that there is never 0 on that territory - wi
ll later be changed on the input mechanics
            territoryb.update_troop_text()
            enemy_territories.append(territoryb)
            if territoryb in p1_territories:
                p1_territories.remove(territoryb)
```

```
                defender_text = font.render("Defender is: None", True, (BLACK)) #update th
e text on the display
                attacker_text = font.render("Attacker is: None", True, (BLACK))
            else:
                neutral_territories.remove(territoryb)
                defender_text = font.render("Defender is: None", True, (BLACK)) #update th
e text on the display
                attacker_text = font.render("Attacker is: None", True, (BLACK))

def win_probability():
    global territorya, territoryb, likelihood
    #Reusing the dice mechanics for the dice roll
    if territorya.troopval == 3:
        attacker_dice_count = 2
    elif territorya.troopval == 2:
        attacker_dice_count = 1
    else:
        attacker_dice_count = territorya.troopval - 1

    if territoryb.troopval == 1:
        defender_dice_count = 1
    else:
        defender_dice_count = 2

    # Roll the dice for each side - a simulation
    attacker_rolls = sorted([random.choice(range(1, 7)) for x in range(attacker_dice_count
)], reverse=True)
    defender_rolls = sorted([random.choice(range(1, 7)) for y in range(defender_dice_count
)], reverse=True)

    attacker_troops = territorya.troopval #creates a copy - preventing erroring
    defender_troops = territoryb.troopval #copy

    # Compare the dice rolls and remove the appropriate number of troops (the simulated re
sults)
    for i in range(min(attacker_dice_count, defender_dice_count)):
        print(attacker_rolls,defender_rolls)
        if attacker_rolls[i] > defender_rolls[i]:#if attacker's result is highest
            attacker_troops -= 1#remove from defender
        else:
            defender_troops -= 1 #otherwise -1 from attacker

    # Return the win probability for the attacker
    likelihood = (territorya.troopval / ((territorya.troopval) + territoryb.troopval) * 10
0)
    return likelihood

while menu == True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
```

```
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:#this bit ensures that the buttons themsel
ves respond to the given input
            if GameButton.checkForInput(pygame.mouse.get_pos()):#GameButton will check if
it is clicked, if so, will output checkforinput value test
                Game()
            if SettingsButton.checkForInput(pygame.mouse.get_pos()):
                Settings()
            if QuitButton.checkForInput(pygame.mouse.get_pos()):
                pygame.quit()
                sys.exit()

    screen.fill(BISQUE)

    #ensures all update with highlight animation
    GameButton.update()
    GameButton.HighlightByMouseHover(pygame.mouse.get_pos())

    SettingsButton.update()
    SettingsButton.HighlightByMouseHover(pygame.mouse.get_pos())

    QuitButton.update()
    QuitButton.HighlightByMouseHover(pygame.mouse.get_pos())

    TitleButton.update()

    pygame.display.update() #updates with feedback
```

Classes.py

```
#classes module
import pygame
import sys


pygame.init()

font_size = 64
font = pygame.font.SysFont(None, font_size)
menu_font = pygame.font.SysFont("Calibri",40) #sets the style of the menu font
font_colour = (0,0,0) #sets the font to black

SCREEN_WIDTH = 1600 #sets the width of the screen
SCREEN_HEIGHT = int(SCREEN_WIDTH * 0.6) #sets the screen height
screen = pygame.display.set_mode(((SCREEN_WIDTH), (SCREEN_HEIGHT)))

p1turn = True#for now is used for turn distinction
```

```
#colour palette declaration - default
RED = (150,6,18)
BLUE = (6,11,150)
BISQUE = (154,255,199)
BLACK = (0,0,0)

#Card mechanics variables
slot_size = 5 #the size of the individual slots
num_slots_main = 15 #number of slots in the main queue
num_slots_priority = 3 #number of slots in the priority queue
queue_position = (50, 300) #the main position of the queue
priority_queue_position = (500, 50)
capacity_indicator_position = (1100, 550) #position of the recording stuffs
capacity_font_size = 24
capacity_font = pygame.font.SysFont('Arial', capacity_font_size)


class MenuButton():#creates the menu buttons
    def __init__(self, image, x_pos, y_pos, text_input):
        self.image = image
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))#co-
ordinates to draw the location of the detection rect
        self.text_input = text_input #allows me to input the text
        self.text = menu_font.render(self.text_input, True, "Black")
        self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))

    def update(self):
        screen.blit(self.image, self.rect)
        screen.blit(self.text, self.text_rect)

    def checkForInput(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(
self.rect.top, self.rect.bottom):
            return True

    def HighlightByMouseHover(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(
self.rect.top, self.rect.bottom):
            self.text = menu_font.render(self.text_input, True, RED) #allows the user to k
now whether the cursor is by the text
        else:
            self.text = menu_font.render(self.text_input, True, "black")#resets back to bl
ack

class GameButtonClass(MenuButton):#creates the new button instance for the Game buttons ie
 Back button etc
    def ___init__(self,image,x_pos,y_pos,text_input):
```

```
        super().__init__(image, x_pos, y_pos, text_input)
        self.image = image #image for this
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.circle = self.image.get_rect(center=(self.x_pos,self.y_pos))
        self.text_input = text_input
        self.text = menu_font.render(self.text_input,True,"Black")

    def update(self):
        screen.blit(self.image, self.rect)
        screen.blit(self.text, self.text_rect)

    def checkForInput(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(
self.rect.top, self.rect.bottom):
            return True

    def HighlightByMouseHover(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(
self.rect.top, self.rect.bottom):
            self.text = menu_font.render(self.text_input, True, RED) #allows the user to k
now whether the cursor is by the text
        else:
            self.text = menu_font.render(self.text_input, True, "black")#resets back to bl
ack

class Territory(MenuButton):
    def __init__(self, image, y_pos,x_pos,text_input,troopval,name):
        super().__init__(image, x_pos, y_pos, text_input)
        self.image = image
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.text_input = text_input
        self.text = menu_font.render(self.text_input,True,"Black")
        self.troopval = int(troopval)
        self.trooptext = menu_font.render(str(self.troopval),True,"Black")
        self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))
        self.name = name
        self.Neighbours = []

    def update_troop_text(self):
        self.trooptext = menu_font.render(str(self.troopval),True,"Black")

    def update(self):
        screen.blit(self.image,self.rect)
        screen.blit(self.text, self.text_rect)

    def checkForInput(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(
self.rect.top, self.rect.bottom):
```

```
            return True

    def HighlightByMouseHover(self, position):
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(
self.rect.top, self.rect.bottom):
            self.trooptext = menu_font.render(self.troopval, True, RED) #allows the user t
o know whether the cursor is by the text
        else:
            self.text = menu_font.render(self.troopval, True, "black")#resets back to blac
k

    def ChangeAllegience(self):
        if self.image == 'ally.png' and self.troopval <= 0: #if this is the image it occup
ies but the troops have been expended
            self.image = 'enemy.png' #change to enemy
            self.troopval = 1 #increment to 1 so it avoids the 0 error
        elif self.image == 'neutral.png' and self.troopval <= 0:
            if p1turn == True:
                self.image = 'ally.png'
                self.troopval = 1
            elif not p1turn:
                self.image = 'enemy.png'
                self.troopval = 1
        if self.image == 'protan_test.png' and self.troopval <= 0: #if this is the image i
t occupies but the troops have been expended
            self.image = 'protan_enemy.png' #change to enemy
            self.troopval = 1 #then add 1 so that it doesn't loop in diff colours
        elif self.image == 'neutral.png' and self.troopval <= 0:
            if p1turn == True:
                self.image = 'protan_test.png'
                self.troopval = 1
            elif not p1turn:
                self.image = 'protan_enemy.png'
                self.troopval = 1
        if self.image == 'deutro_test.png' and self.troopval <= 0: #if this is the image i
t occupies but the troops have been expended
            self.image = 'deutro_enemy.png' #change to enemy
            self.troopval = 1 #then add 1 so that it doesn't loop in diff colours
        elif self.image == 'neutral.png' and self.troopval <= 0:
            if p1turn == True:
                self.image = 'deutro_test.png'
                self.troopval = 1
            elif not p1turn:
                self.image = 'deutro_enemy.png'
                self.troopval = 1
        if self.image == 'trit_ally.png' and self.troopval <= 0: #if this is the image it
occupies but the troops have been expended
            self.image = 'trit_enemy.png' #change to enemy
            self.troopval = 1 #then add 1 so that it doesn't loop in diff colours
        elif self.image == 'neutral.png' and self.troopval <= 0:
```

```
            if p1turn == True:
                self.image = 'trit_ally.png'
                self.troopval = 1
            elif not p1turn:
                self.image = 'trit_enemy.png'
                self.troopval = 1

    def add_neighbour(self,territory):
        self.Neighbours.append(territory)

    def is_neighbour(self,territory):
        return territory in self.Neighbours



class Cards():
    def __init__(self, image, type, titlename):
        self.image = image #the image
        self.rect = self.image.get_rect() #the clickbox
        self.type = type
        self.titlename = titlename #this is to identify if the title matches self.territor
yname later on in development

    def draw(self, screen, x, y): #draws the object onto the screen
        self.rect.x = x
        self.rect.y = y
        #puts the image onto the screen
        screen.blit(self.image, self.rect)

    def check_for_input(self, GAMEBUTTON_MOUSE_POS):  #checks to see if object clicked by
mouse
        return self.rect.collidepoint(GAMEBUTTON_MOUSE_POS)

class Queue:
    def __init__(self, num_slots, position):
        self.num_slots = num_slots #defines the number of available slots
        self.position = position
        self.slots = [] #defines the slots themselves as an empty list
        self.capacity_indicator = capacity_font.render(f"{len(self.slots)}/{self.num_slots
}", True, (0,0,0))
        #creates a surface, for testing purposes, so that i may see if the cards fulfil it
s capacity limits

    def add(self, obj): #defines the adding mechanics between queues
        if len(self.slots) < self.num_slots: #checks to see if the slot number is less tha
n the number of available slots
            self.slots.append(obj) #if there are sufficient slots available, append the ob
ject
            self.capacity_indicator = capacity_font.render(f"{len(self.slots)}/{self.num_s
lots}", True, (0,0,0)) #records the capacity
```

```
            return True
        else:
            return False #if it exceeds capacity, do nothing

    def remove(self, obj): #defines the removal mechanics between queues
        if obj in self.slots: #if the object is to be found in the slot position
            self.slots.remove(obj) #remove the object
            self.capacity_indicator = capacity_font.render(f"{len(self.slots)}/{self.num_s
lots}", True, (0,0,0))

    def draw(self, screen):
        for i, obj in enumerate(self.slots):
            x = self.position[0] + (i % 5) * (slot_size + 200) #draw each card 200 x co-
ordinates away from one another
            y = self.position[1] + (i // 5) * (slot_size + 300) #for each new line in the
queues, add 200 y co-ords of space
            obj.draw(screen, x, y) #draw the objects onto the screen in their assigned pla
ces
        screen.blit(self.capacity_indicator, capacity_indicator_position) #put the objects
 onto the screen
```