

[name]

Candidate No: xxxx

Centre No:xxxxx

Non Exam Assessment Programming Project

“Throughout History – An Idle Game”

[name]

Candidate No: xxxx

Centre No:xxxxx

1. Analysis of the problem	7
Problem Identification	7
Problem decomposition.....	8
Divide and conquer	9
Abstraction.....	9
Stakeholders	9
Justification of Computational Methods	9
Interview	10
Creation.....	10
Analysis of survey.....	11
Stakeholder Requirements	18
Research of existing problems	19
Progress bars and resource visualisation.....	19
Portals and Run Resets	21
Offline Progress System	21
Save/Load system	22
Combat.....	23
Upgrades	24
Log system	26
Other notes found out about the games	27
Findings	28
Limitations.....	28
Second Interview	29
Questions	29
Sammy's Response.....	30
Alex's Response.....	30
Christian's Response	31
Stakeholder Requirements	31
Hardware and Software Requirements	33
Success Criteria	34
Abstraction Diagram	37
2. Design.....	38
Problem Decomposition	38
Resource Collection decomposition diagram	40
Logs decomposition diagram	41
Wars decomposition diagram.....	42

List of wars	43
Save/Load decomposition diagram	43
File Structures	44
Graphical User Interface Design	45
Main	45
Resource collection [A]	46
Upgrades [A]	48
Save/Load [B]	49
Logs [D]	50
Combat [C]	50
Inputs Table [TC]	51
Sub-program design.....	52
Class Links	53
GlobalData	53
ResouceCollection [A]	55
Upgrades [A]	64
Combat [C]	75
V1 16/09/19	75
V2 11/11/19	84
V3 18/11/19	85
Save/Load [B]	86
Logs [D]	90
Variables Table.....	92
Testing Table	97
Alpha Test Table.....	97
Beta Test Table.....	103
3. Development.....	107
Main Setup 03/10/19.....	107
GlobalData Setup 03/10/19	108
Resource Collection GUI [A].....	108
V1 05/10/19	108
V2 18/10/19	109
V3 23/10/19	110
V4 24/10/19	110
Resource Collection Code [A].....	111
V1 09/10/19	111

V2 17/09/19	120
Menu Strip [E]	121
V1 12/09/19	121
V2 21/10/19	122
V3 02/11/19	123
Upgrades GUI [A]	124
V1 15/09/19	124
V2 24/10/19	125
Upgrades Code [A]	126
Storage	126
V1 22/10/19	126
Housing	137
V1 23/10/19	137
Workers.....	145
V1 24/10/19	145
V2 25/10/19	157
Combat.....	158
V1 25/10/19	158
Research.....	164
V1 26/10/19	164
Review 1.....	170
Logs GUI [D]	174
V1 27/10/19	174
Logs Code [D]	176
V1 01/11/19	176
V2 21/12/19	179
Review 2.....	180
Combat GUI [C]	184
V1 02/11/19	184
V2 27/11/19	186
Combat Code [C]	187
V1 07/11/19-18/11/19	187
V2 27/11/19	209
Review 3.....	210
Save/Load GUI [B]	214
V1 19/11/19	214

V2 15/12/19	218
Save/Load Code [B]	219
Buttons	219
V1 21/11/19	219
V2 15/12/19	220
Display Files.....	221
V1 23/11/19	221
V2 15/12/19	221
Delete Files.....	222
V1 26/11/19	222
V2 15/12/19	225
Autosaving	227
V1 29/11/19	227
V2 15/12/19	227
V3 20/12/19	228
New File structure	230
V1 02/12/19	230
Saving to file.....	232
V1 05/12/19	232
V2 15/12/19	238
Loading from file	241
V1 09/12/19	241
V2 14/12/19	246
Stakeholder Feedback.....	249
Balancing [E].....	250
Review 4	251
4. Evaluation	255
Walkthrough of Solution.....	255
Beta Testing	260
Usability Tests	260
Stakeholder Feedback.....	260
Overall Feedback.....	263
Destructive tests	264
Solution Success.....	279
Success Criteria	279
Stakeholder Requirements	295

Description of Final Product	298
Limitations.....	298
Maintenance and Development of Solution.....	299
Final Code.....	300
GlobalData.cs	300
MainForm.cs	302
ResourceCollection.cs	305
Upgrades.cs.....	310
Combat.cs	318
Logs.cs	328
FileHandling.cs	329

1. Analysis of the problem

Problem Identification

In this section, the problem addressed is defined as an incremental (idle) game. An idle game is associated with the genre of “video games whose gameplay consists of the player performing simple actions repeatedly to gain currency”. This can be used to obtain items or abilities that increase the rate at which currency accumulates. It requires less frequent interaction than normal games do. Typically, an idle game can be set up, and then eventually can get a spot where the game automatically/semi-automatically continues on its own. The player can then come back multiple minutes/hours later and continue the game at a spot further than before.

The idle game I would like to create is based off real-world history which progresses through time, introducing new game mechanics, content and storyline. At first, there are basic resources to collect, and they get more advanced as time progresses, starting with wood and stone, ending with modern day materials like uranium. There is a story line which is the world history that it moves through. In order to progress through the story line, the players are able to take part in wars that actually happened, and gain/lose progress from winning/losing them. The story line will be linear because then the game does not have to generate thousands of game progress possibilities where anything could happen/change from the real world timeline. Along with this comes a combat system that allows wars to be fought, won and lost.

A very common feature of idle games is the progress bar – visualising how far towards a certain goal the user is, e.g. how much damage has been dealt to an opposing unit. Idle games should also be able to run in the background, so have a save/load games system which includes an offline progress system so that progress can be done in the background. An upgrades system allows the progress through the game faster, gives the player a sense of purpose, and increases the number of options the user has to progress. A log system may also be useful to allow the user to easily look through specific messages that give them verbal clues of how far along they are into the game, e.g. how much loot they got from winning a war. If a player leaves the game running overnight, the storyline sections could be isolated so they can read through it in one go without being interrupted by other parts.

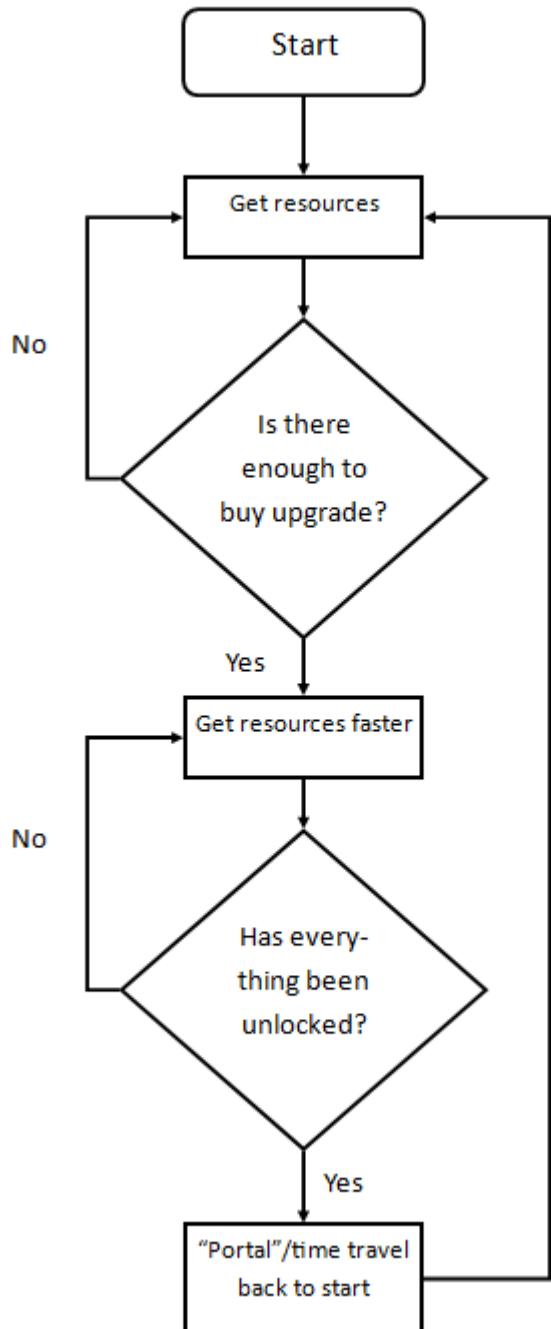


Figure 1 (above): The structure of most idle games

Problem decomposition

The problem can be decomposed into a set of much smaller steps. The steps are:

1. Resource collection and visualisation
2. Storyline
3. Combat/wars
4. Save/load system

Once all of the basic parts of the steps are completed, and it is done fast enough so that there is little/no lag, the rest of the game can be built upon the foundations much faster.

Divide and conquer

Each of these steps will need to be completed separately, and I will try to keep them as separate as possible to solve the problems on their own. I can then combine them into a modular program which makes use of the divide and conquer problem solving method.

Abstraction

In my game, I don't want to have to worry about certain details, like the style of the progress bar, or what the combat units look like. With resource visualisation, I don't need to worry about:

1. The styles of the progress bars
2. Showing the exact number of a resource, e.g. 100,000,000 stone, which would need the GUI to stretch to provide more space, or decrease the size of the text, overcomplicating the system. Instead, I could have a system that detects when a "milestone" is reached, and add the appropriate symbol at the end. E.g. 1,000,000 becomes 1M, and 134,450 becomes 134.45K
3. How the resources are collected

I also won't need to worry about what the terrain looks like, or where on the "world map" the wars will need to be. That's something I can look past and work on the core parts, like the actual battle mechanics.

Stakeholders

The main features are to be developed through the investigation and analysis off existing idle games; the requirement analysis through interviews and questionnaires from potential player with an age range of 8-18 years old. The group of people will include a year 7, a year 8 and so on (from my tutor group) all the way up to year 12 where I will have many stakeholders. I will include other friends whom I know have long histories of playing idle games to interview for ideas. I could give this to history teachers to play to check the accuracy of the history timeline in the game, and also some GCSE or A-Level history students to allow them to have fun whilst also learning a little bit about past wars and their outcomes.

Justification of Computational Methods

An incremental game cannot be played physically because every tick, a new value has changed, and it would be very tedious to play without the computer doing it for the player. Another feature of an idle game is that the player can leave for a long time and come back to see that automatic progress has been made. This progress can only be made by the computer running the program whilst the player is gone. Different save games can also be loaded/saved on a computer filesystem, whereas that isn't practical or even possible using physical pen and paper.

When the game is being played, the players can use the internet to help pick the right side of wars, so that they know which side won in real world history, thus can win the wars themselves. Looking

up these facts online can help them with their learning of the world's past, and actually experiencing the war themselves in-game will most likely consolidate the facts into their memory.

Interview

Creation

I will interview a range of students who do a level computing, a level history and some others between ages 13-18 that go to my school. These are the questions I intend to ask:

1. Where in world time should I start?
2. When the player gets to "modern" times, they can "portal" that allows them to save certain tech/items and go back to the beginning
 - a. Strongly Disagree
 - b. Disagree
 - c. Neutral
 - d. Agree
 - e. Strongly Agree
3. From your chosen time, how many basic resources should I start with, and what would you call them?
4. Should I implement a log system that allows the player to read story lines, war status or loot gained?
 - a. Yes
 - b. No
5. Should I use progress bars, numbers, or both to visualise item count, rate of gain or max amount stored?
 - a. Progress Bars
 - b. Numbers
 - c. Both
6. When the game is closed, it is saved, but should I implement an offline progress system?
 - a. Yes
 - b. No
7. Should I implement an auto-save system?
 - a. Yes
 - b. No
8. Which should I implement, a turn based or a real time based combat system?
 - a. Turn based
 - b. Real time
9. Should wars have "difficulty levels" that make the user work harder to defeat?
 - a. Yes
 - b. No
10. When a player wins a war on the "wrong" side, how would the game get back to that one timeline?
11. What should the game look like?
 - a. Different forms that pop up or change after a set milestone is reached
 - b. One form, but when milestones are reached, new parts pop up
12. Add anything about the game that you would like to ask/mention

I have asked these questions in order to narrow down the finer details of the game and what will be in it, and what won't. Some questions are about the times/timeline (1, 2), some are about resources and visualisation (3, 4, 5), some questions are about a saving system (6, 7), wars/combat (8, 9, 10), and the last couple for what the game itself should look like. I use a bias system "Strongly Agree" to "Strongly Disagree" on question 2 because when informally asking many people on it, most of them had complex views on it, rather than the "Yes" or "No" which does not allow them to so easily make this opinion clearer.

I made questions 3 and 10 open ended "written" answers so that the stakeholders can give more details on what they think should be in that part of the game. Especially with question 10, because it is a very complicated problem and needs to be addressed fully, as many different games could be made off each solution put forward.

I needed to add question 11 because of the nature of idle games (new "base" parts popping up every so often), I want to know whether the stakeholders want the parts to pop up in different areas of the screen overtime (in different forms), or fill in the blank space on one form/show new menu tabs when parts are unlocked. Again, there is a lot to be done with either option so the stakeholders need their input on that problem.

Analysis of survey

Here were the outcomes of the stakeholder survey. There were 9 responses all together. 6 were from my A-level Computer Science class, 1 has played idle games for the past few years, 1 is a well-known game developer, and the last is a student doing A-level history.

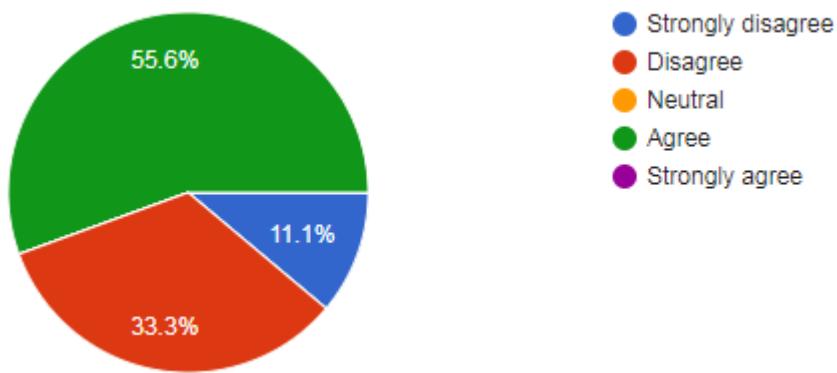
1. Where in world time should I start?

- a. 4000BC (x2)
- b. Ancient Egypt
- c. Roman times (x3)
- d. Medieval – 1066
- e. When the first amoeba was – 1767
- f. Modern times

Start Time	Justification
Roman times	First of all, it seems that many people would like the game to start around the Roman times, when there are some basic developments in technology and combat, like automatic farming, aqueducts, barracks etc. This allows the game to start off with action like the Trojan War for example, allowing the player to be quickly introduced to the war system of the game.
4000BC	When I followed up with these people about why they wanted it to start around this time, both said roughly the following (they are completely separate and do not know each other): "The game can start off at this time with the most basic of resources – wood, stone, a little food. But because there was so little advancements in technology, the game should run quite fast through the years until around the Roman times (200-100BC) when it should slow down because all of a sudden there was a lot of advances in human civilisation and technological advances. The game should stay slow through around that time period until the middle ages when time should speed up a bit again because of a lack of developments/major wars, and then slow down again around the rise of the British empire, when big technological advances are made again." So what they

	are saying, is that when there was a lot of activity in world history, game time should slow down, and when there was little, game time should speed up. This should therefore balance out the amount of time that users spend in the more important and “well recognised” times of world history.
Other	I will ignore the “modern times” and the “when the first amoeba was discovered” because I know I cannot have the game start in either of those times as there will not be nearly enough content of the player to get through. It will also require them to be introduced to a lot of stuff at once, thus introducing a very steep learning curve which would not be good.

2. When the player gets to “modern” times, they can “portal” that allows them to save certain tech/items and go back to the beginning



The “portal” question was split on what people wanted. Just over half agreed with the idea and want it to be in the game. When I asked someone who selected this option, I found that people wanted to be able to play the game over but with a bit more stuff to start off with, thus working their way through the world timeline in a different way. They also asked if there could be “challenges” that the user will need to overcome during their “run” (a run is going through the world history once, and then once they portal, their run will reset). When they portal again, the next run will become easier as the challenge would have unlocked better perks like faster food production or similar.

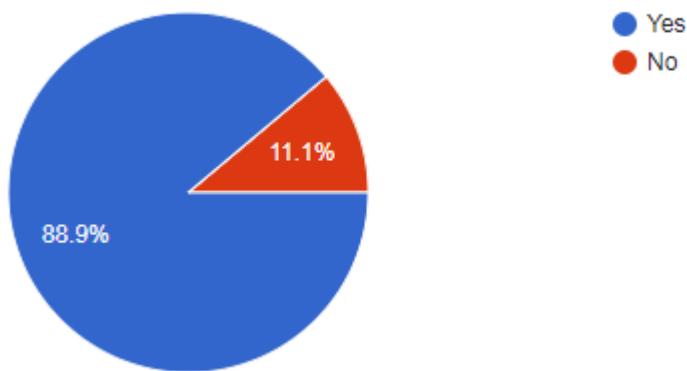
When I asked someone who had put “strongly disagree” as their answer, they said that they did not want the ability to reset the game as it would “ruin the game’s balance” and that it would be very hard to rebalance for every new run.

3. From your chosen time, how many basic resources should I start with, and what would you call them?



The resources question came up with a lot of suggestions that I was not expecting, like happiness, population, and currency. The population idea is very good because it allows the game to keep track of how many people are available for various jobs like soldier, farmer, lumberjack etc. This then means that the player will have to also balance their population as well as other parts of the game, to keep their empire strong. Happiness is also a good suggestion because it can be used as a multiplier for resource gathering and combat. If people are not happy, they will not fight very well, or have no motivation to farm for more food than they themselves need. This can also be used as a determining factor for deals and going into wars etc. I do not agree with having currency in the game however, because it would bring unnecessary complications with the currency essentially being the “middle-man” of buying new combat units for example.

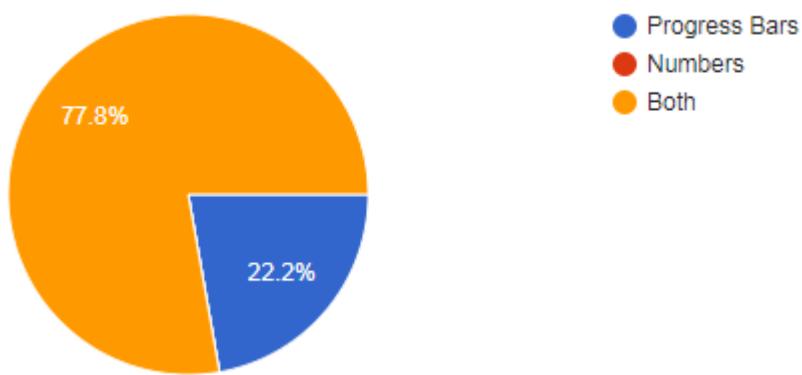
4. Should I implement a log system that allows the player to read story lines, war status or loot gained?



It is clear that almost all of my stakeholders want to have a log system in the game. When I asked one of them why they wanted it, they said that they didn't know of another way that would inform them of the various game decisions that the player makes, or how else the game story would be presented, without being annoying or in the way. When I asked the person who said they didn't

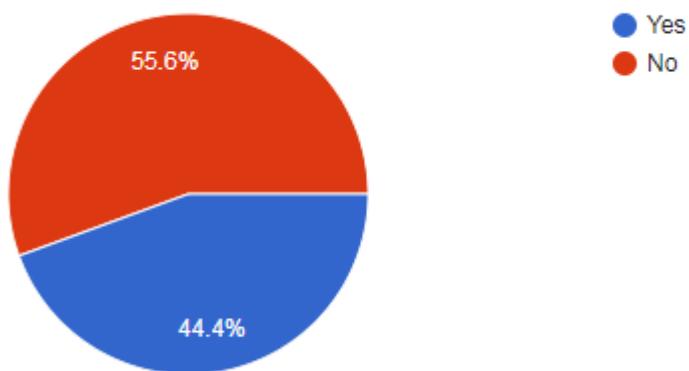
want this system, they said that they didn't want the screen to be "filled up" with these logs that the game would be "spewing out".

5. Should I use progress bars, numbers, or both to visualise item count, rate of gain or max amount stored?



Question 5 surprised me because I thought everyone would have voted both, rather than just progress bars or just numbers. Although there was a majority who voted both, I still wondered why some people wanted just progress bars. When I asked one of the 22.2% of people who said "progress bars", they said that they didn't want the resources part to be filled up with various numbers that don't make much sense and flood the player with "too much information". However, they said that maybe they would put both if the game only had numbers that displayed the rate of resource gain. This is something that I will think about when designing the game, although I will most likely still display rate of resource gain, number of resources, and the storage limit for that resource, as well as the progress bar to visualise this.

6. When the game is closed, it is saved, but should I implement an offline progress system?

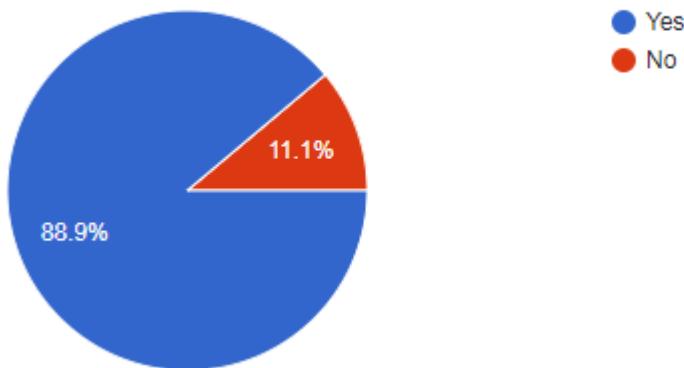


For question 6, the "no" option just about wins, by 1 vote. This means that some of my stakeholders do not want an offline progress system, whilst some do. However, when I spoke to someone who voted "no", they had misunderstood what I meant by "offline progress system". My idea of the system is that when the user wants to stop player (and close the game), the game will save, and also

print the system time into the save file. When the game is loaded back up, the time difference is calculated between save and load, and “progress” is suddenly made up in all of that time. It would only collect the number of resources it would have collected in that time, rather than that and playing in wars or other events.

The big problem that arises with this is should the game also run along its time history, and if so, how fast? What if the player quits the game for say, a year in real life, how long does the time in-game go along? One possible solution would be to allow the offline progress to “run” through time until the resources full up, and then it stops. This would mean that not much time would pass for the player to miss, but there would also be some “progress” when the player isn’t playing, so that they can go back to faster advancements until they use up their full resources.

7. Should I implement an auto-save system?



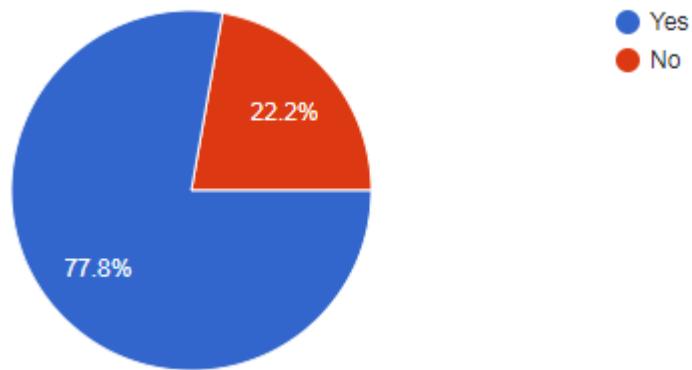
A big part of idle games is a save & load system. Of course, if it takes years of real time to get through all of the content of an idle game, the players will not want to lose their game progress every time they quit the game. Therefore, a system could be implemented where the player can save a game when they want to quit, and load a game when they want to play. However, what if their computer suddenly lost power? The Operating System needs to prioritise the safe shutting down of the important data that cannot be lost, rather than the games running on it. Therefore, the game should have a system where it auto-saves to a file every x minutes, so that in the event of a sudden shutdown, there is a backup so not much progress is lost. It is obvious that the majority of my stakeholders want this. The person that said “no”

8. Which should I implement, a turn based or a real-time based combat system?

- a. 100% Turn based

It very blatantly obvious that all of my stakeholders want a turn based combat system rather than a real-time based combat system. From asking a few of my stakeholders, I learnt that they all put this because they either were most familiar with the turn-based system and thus want to stick with it, or they find it easier to use rather than a real-time based system.

9. Should wars have “difficulty levels” that make the user work harder to defeat?



When I made this question, I was unsure of the direction it was going to take. I thought that most people would put no, when actually most people actually like the idea. I thought that most had wanted a “simple” war system where you could choose which side to put your resources and troops into, and after a bit of fighting, one side comes out better than the other, and depending on which side you pick, your resources get a boost or are lost. However, some people said that they want a challenge of getting through the wars, by having to “upgrade” their combat abilities whilst at war so they can defeat the more powerful enemies. If they don’t upgrade fast or effectively enough, they will get beaten at the “later” combat and thus may contribute to the loss of the war.

10. When a player wins a war on the “wrong” side, how would the game get back to that timeline?

I do not know, perhaps the new government collapses or is overthrown

They should be able to be on the wrong side

Carry on regardless

That's a hard question and there's no easy solution, you'd have to work it out on a case-by-case basis.

Make the government get overthrown or turned over by rebels to eventually turn world history back to the right way

Yes, but have different bonuses/penalties stem from it that last for the whole game until a time warp perhaps.

No. Any “war” type events are loosely based on our time but not exactly (E.G. At 1945 you get dragged into a war with every country and have to pick one side, you would then get a boost in your side wins)

Have it be impossible to ‘be’ on the wrong side

Just keep going but have some flavour text change depending on who wins

This is probably the biggest problem for the whole game that has many solutions that could take it in any direction. First of all, I want the game to follow a singular timeline, the one that world history has taken place on. This is because generating different timelines based on war outcomes will be very, very hard, and would be a whole project in itself. Therefore, I can rule out the “carry on regardless” answer, the “they should be able to be on the wrong side” and the “work it out on a case-by-case basis” answers.

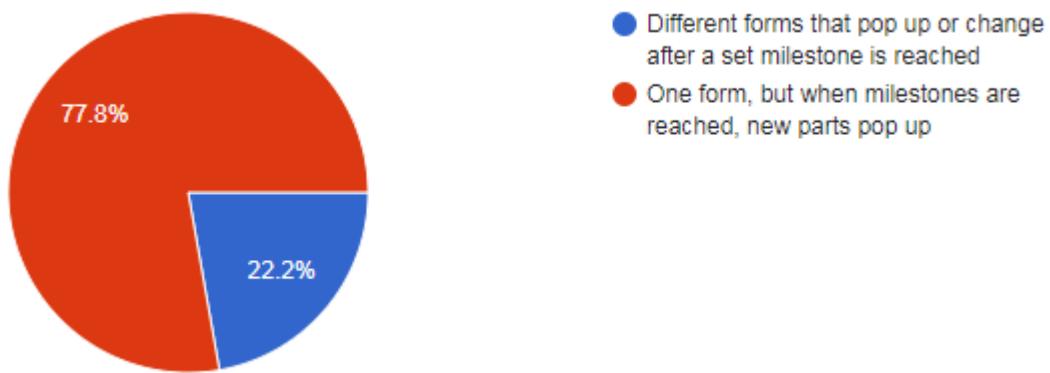
Some of the answers like overthrowing the new governments are good, but in some cases won't make much sense because when a war was won in real world history, the new governments never really got overthrown right after winning every single time. So I can probably rule out these as well, although they are still a possible solution if no other solution is good enough.

Yes, but have different bonuses/penalties stem from it that last for the whole game until a time warp perhaps.

No. Any "war" type events are loosely based on our time but not exactly (E.G. At 1945 you get dragged into a war with every country and have to pick one side, you would then get a boost in your side wins)

These two answers above most likely represent the best solution. What they are saying is that your empire is "neutral" and when major world wars come up, the player will have to pick a side to fight on, depending on what the best outcomes are for winning/losing on that side. The outcomes for being on the winning side is a resource gathering or combat bonus until portal, and the outcomes for being on the losing side is a penalty to these things. For example, if it were 1945, the player's empire gets dragged into World War II, and the player picks the side of Germany. They will fight, but it is likely that you will lose the war, thus gain a penalty for x amount of time. Time will continue as if nothing has happened. If you win the war on Germany's side, because the wars are loosely based on real world history, nothing will come out of it other than the player gaining the boost. This means that I will not need to worry about generating hundreds or thousands of timelines based off outcomes of each war, nor do I need to worry about governments being "overthrown" and returning back to the single timeline.

11. What should the game look like?



I included this question because I wanted to get a tiny bit of answers on the whole GUI layout of the game, since it is a large part of it. I have seen some idle games with very contrasting UI layouts and I want mine to be different. It is clear that the majority of my stakeholders want the game to fill up the whole screen, but there be blank parts that fill up as milestones are reached. E.g. Combat is unlocked and the entire bottom right of the screen fills up with the combat panel. However, I will need more input from the stakeholders about how this should be presented, either in one screen, or within tabs or a menu strip.

12. Add anything about the game that you would like to ask/mention

a. Will there be a tutorial character that teachers the player?

- b. There should be options to do “shady” deals that have positives and negatives E.G.
Start the Atlantic Slave Trade +Man power +Food +Metal BUT –Happiness –Innovation
- c. You should implement the Christero war
https://en.m.wikipedia.org/wiki/Cristero_War (look at the belligerent)

I find answer b interesting because it should be easy to implement in the game, but also be an interesting mechanic that might hook players in more, by having to make more decisions with how they want their empire to go. This would also mean that each run could be played in different ways and thus people will play it more.

Stakeholder Requirements

Based on the results of the interview, the stakeholders want:

- 1. The game will start between 4000BC to Roman times
- 2. Players will be able to “portal” once they reach modern times
- 3. Basic resources will be:
 - a. Population
 - b. Metal
 - c. Food
 - d. Research/Science/Tech/Innovation (all same thing)
 - e. Wood
 - f. Stone
- 4. A log system that displays story lines, game decisions, loot gained and other related things
- 5. Progress bars and numbers to display resources data
- 6. There will be an option to have the offline progress system running or not
- 7. There will be an auto-save system
- 8. Combat will be turn-based
- 9. Wars will get harder as they progress, forcing the player to spend on upgrading combat
- 10. Wars are loosely based on real world history, and the player picks a side to join. If they win, they get a bonus, if they lose, they get a penalty
- 11. The game will run on one big windows form but different parts will pop up when milestones are reached
- 12. There will not be a tutorial, however some early game mechanics will be introduced
- 13. There will be “shady” deals that have positives and negatives towards resource collection, population and combat.

Research of existing problems

I will now research existing idle games, so that I can discover any new ideas or limitations to my stakeholder requirements.

Progress bars and resource visualisation

In the well-known game AdVenture Capitalist, the progress bars and visualisation of how much money the player has is unique to many other idle games I have seen.



Figure 2 (Left): A snippet from the idle game AdVenture Capitalist

Above shows a list of some of the items the game is automatically running, to gain more money. This money is then used to buy more of these items, which then fetch more money per run and the next level's price is also a bit raised. This loop continues on until the player has to wait for more money to be gained over time (or pay real money to get virtual money faster).

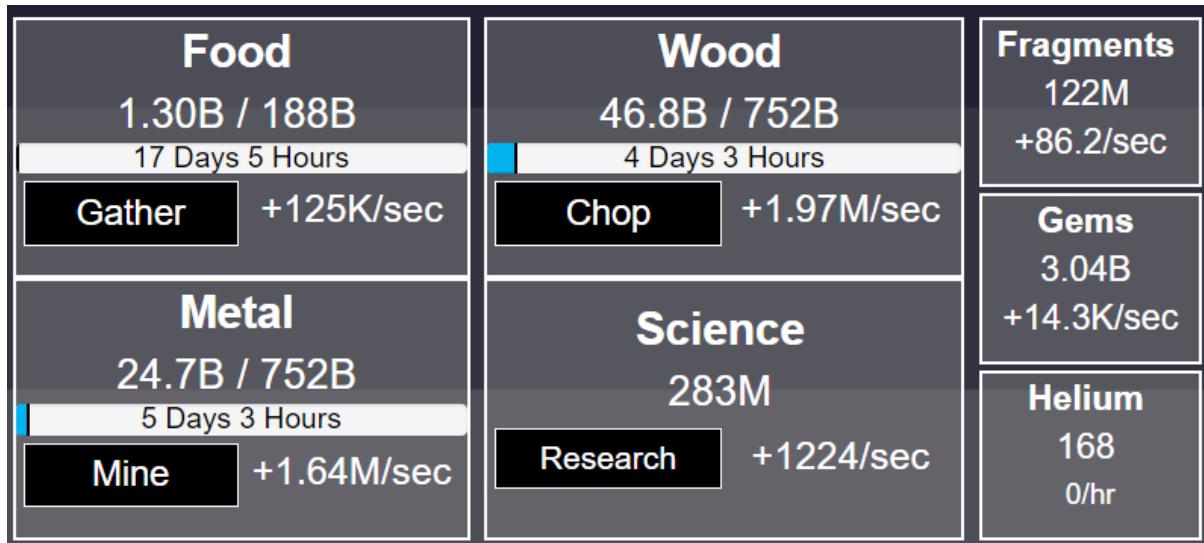
Figure 3 shows the progress in time towards the end point when the cash is given to you. Here, the "9,000.00" represents the \$9000 the player will earn once the time is complete. The little grey box underneath the progress bar is the amount of time left until the timer runs out. This is another way of telling the player how much time is left until the next cash comes in. However, the progress bar is there above because it is much better at visually representing the time, although not the exact amount left. Also I note that in this game, the progress is in time, rather than amount collected. This

is because in this game, there is no “limit” of this specific resource, thus the progress bar does not need to be based off that.



Figure 3 (Left): A close up of one of the progress bars in AdVenture Capitalist

Figure 4 (Below): The resource collection panel of the idle game Trimp



Trimp is a web-based idle game that is all about getting to the highest “zone” possible. Each zone brings new challenges combat-wise, and to get past these, the player must upgrade their combat abilities, which means they must find ways to collect more resources faster.

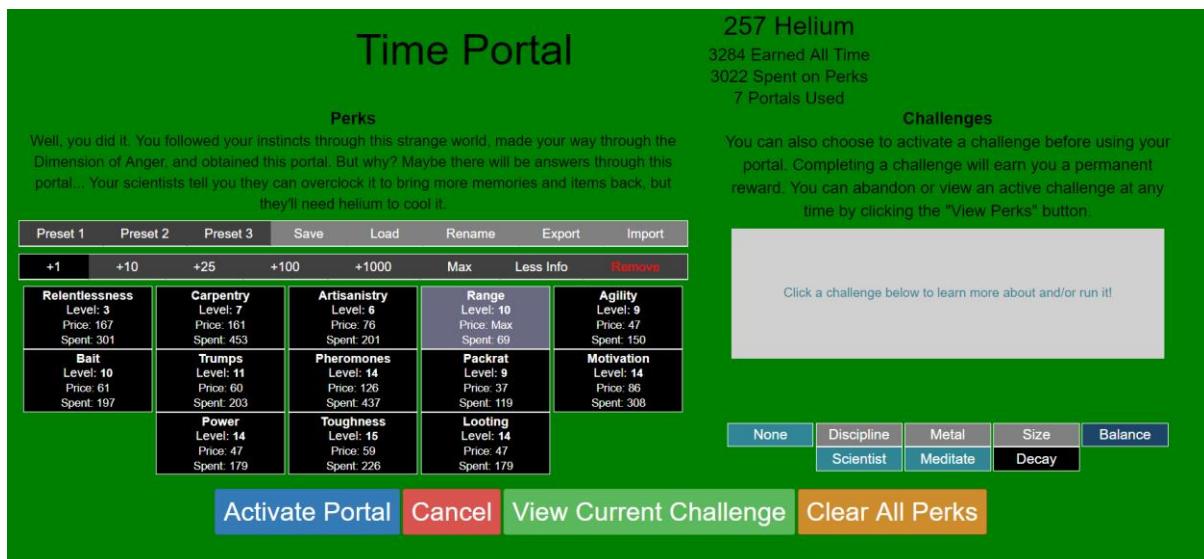
Figure 4 shows the resource collection part of the game. The progress bars represent each resource in relation to the max amount of resources that the storages allow. However, the text on the progress bars show time. This time is actually how long it will take, at that rate of collection, to reach max storage. The amount of each resource is stated above, and the maximum amount. The rate of resource gain is also stated.

Portals and Run Resets



Figure 5 (above) shows AdVenture Capitalist, where the player can “claim” angels to restart with a boost in profits. They can also be used to buy new upgrades which increase profit gain even faster. This teaches the player to sacrifice short term progress for long term progress, as over time, they will get further faster, thus gaining more angels, thus increasing the speed the game is played faster. It is a key part of idle games, to keep players hooked on, without having to add an increasing amount of content.

Figure 6 (below) shows Trimps, where the player can go back in time to restart but with some “Perks”. These perks cost Helium, which is a rare resource and is hard to get. However, the perks are very useful for getting to the stage where the player can portal again.



Offline Progress System

Most idle games have offline progress systems that makes progress with certain aspects of the game, usually with resource collection – the most automated part of the game. In Trimps, for example, when the player comes back to the game, a message box pops up telling the user how much resources were collected while they were gone (see Figure 7 below). However, even though there is an auto combat feature (that I will cover later), the combat progress does not continue, because some story and major advances could be made when the user is offline, and that would be bad because they could miss a major story event.

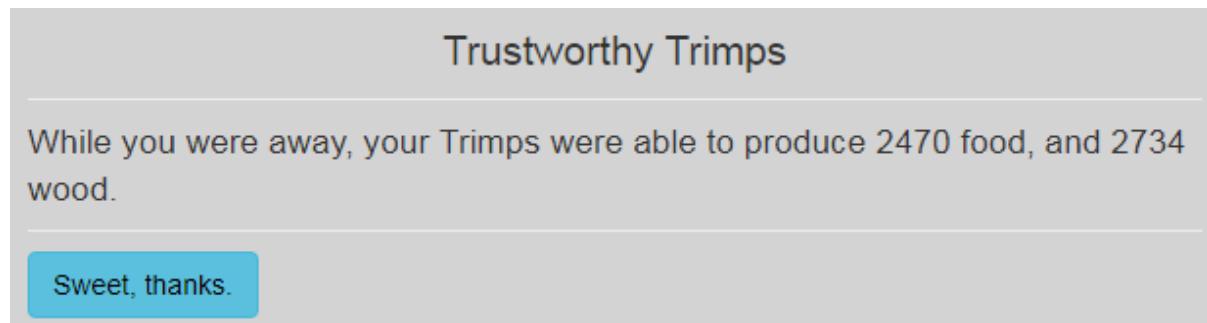


Figure 7 (Above)

With AdVenture Capitalist, the offline progress system works the same as in Trimps, because when the user comes back to play, they are met with a box that tells them how much they made in the last x hours, x minutes they were gone for.



Save/Load system

Every idle game I have seen or played has a save/load system. This is because it can take days, weeks or even years to finish all the content in an idle game, and losing all of that progress will most likely cause the player to quit. Therefore, implementing this system is a key feature to the success of my game, as not having one would see a big drop in the player base number.

First of all, there is the save system. Trimps has this feature automatically implemented so that the player never has to hit save (although they can manually), because it is all done automatically. Whilst the game is being played, every x minutes (time interval can be changed by player in settings), the game is automatically saved. When the game is closed, the game is also saved before it closes. Even though Trimps runs inside the web browser and all saves are cached into the browser, it would still work the same if the game saves to a file in the computer system. AdVenture Capitalist does not have an integrated save or load button – the player cannot save games manually, as everything is done per PC automatically (when the game is closed).

Again, in the load system in Trimps, the game is automatically loaded from the browser cache (or the Trimps servers if the player has an account), or the user can import games manually from their PC files. In AdVenture Capitalist, players cannot load games manually – the game only loads from the PC files automatically.

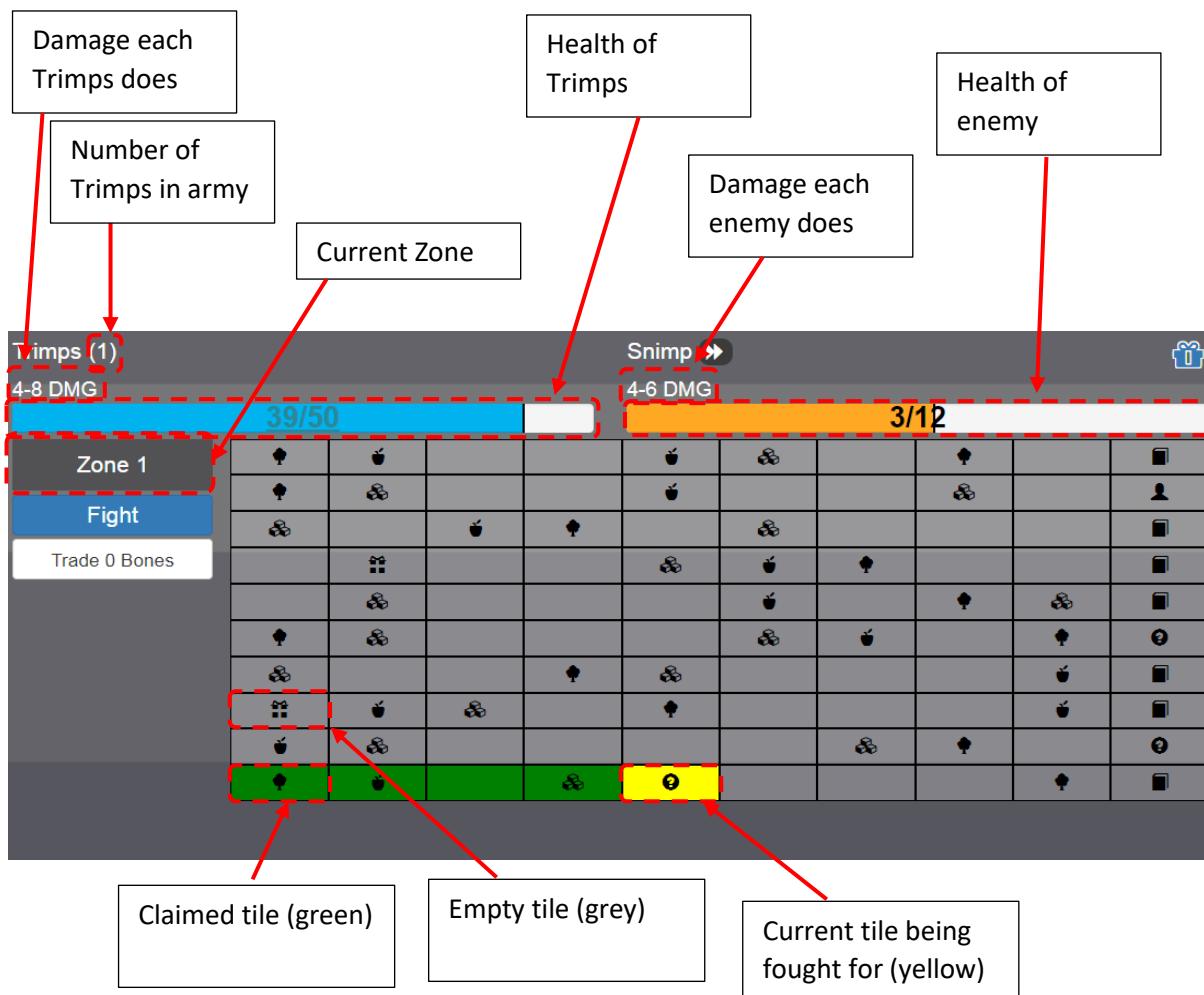
Combat

Many idle games do not feature combat at all. AdVenture Capitalist does not have any combat, because it focuses on the upgrades/money making side of the game, rather than fighting with some other large cooperation or similar. Trimps on the other hand has a complex combat system that is a large part of the game.

The table below shows the varying features of the combat in Trimps:

Combat System	Explanation	Needed?
Block	When the enemy attacks, the troops can block a certain amount of damage. E.g. the block of the troops is 100, the enemy does 150 damage on the hit. Therefore, the troops block 100 of the damage, so take only 50 damage. This can be upgraded by buying more trainers (a work type), or by buying gyms (a housing type).	No
Attack	This is how much damage a side does to the other, e.g. 150 damage takes away 150 health (assuming block is 0). Although in Trimps, the damage is random between 2 values for each the Trimps and enemies. This is upgraded by buying new weapons.	Yes
Health	The health is self-explanatory – how much damage can be done to the troop/enemy until it dies (assuming 1 damage takes 1 health).	Yes
Square grid	This grid shows how far the Trimps have gotten in the “Zone”, and what type of loot is in each square. Every tile in the grid either has loot on it, or is blank. When a battle is won, and a tile is claimed, the tile turns green and if there is loot on it, the loot is claimed.	No
Turn based	The army of Trimps hit first, then the enemy, and so on until one of them dies. If the enemy dies, the tile is claimed and they move onto the next tile. If the Trimps die, the player can click the “fight” button to send more Trimps if they have enough, or when the upgrade is available, the Trimps will automatically go to battle once there are enough.	Yes
Zones	The zones are like levels of difficulty, and completing them unlocks a map difficulty for Trimps to train in, so that they can get upgrades to make the next Zone easier. So the player usually plays a zone -> plays a map for upgrades -> plays next zone. Enemies also get harder as the Trimps progress through the zone, but not by much.	No
Maps	The maps, as said above, are for the upgrading of the Trimps' block, health and damage, to make the next zones much easier (and faster) to run through. Map difficulty tiers run by the current zone you are in, and completing a zone unlocks that map difficulty tier. E.g. if the player is in zone 3, they will have access to maps with difficulty levels of zone 1 & zone 2. Maps are bought using “explorer fragments”, a resource collected by completing zones. They can be made easier, and harder, by changing the difficulty sliders and loot levels, but making them	No

	easier increase the number of explorer fragments the player needs to collect.	
--	---	--



Upgrades

The upgrades system in Trims is very easy to use, and user friendly. There are different tabs for types of upgrades, e.g. housing, workers, combat etc. There is also an "All" tab which allows players to see all the available upgrades on one page. The way it works is that when a box showing the upgrade, and the cost of the upgrade, is black, the player has enough resources to buy it. When it is

grey, the player does not have enough resources to buy it. Additionally, the player can select how many of each upgrade they want per click, also known as buying in bulk. They can select +1 (upgrade), +10, +25, +50, +100, +max, and then +custom (that the user can input manually).

All	Buildings	Jobs	Upgrades	Equipment
+1	+10	+25	+100	+1000
Custom Max				
Buildings				
Trap 0	Barn 8	Shed 9	Forge 9	
Hut 6	House 3	Gym 10		
Jobs		35 workspaces	Fire	
Farmer 40	Lumberjack 75	Miner 64	Scientist 20	
Trainer 11				

All	Buildings	Jobs	Upgrades	Equipment
+1	+10	+25	+100	+1000
Custom Max				
Jobs				
35 workspaces			Fire	
Farmer 40	Lumberjack 75	Miner 64	Scientist 20	
Trainer 11				

There are, however, downsides to this system. If the player accidentally buy hundreds of upgrades (+100 bulk buy on), they cannot "roll back" thus lose potentially many hours of resource collection time, which can be quite frustrating.

In AdVenture Capitalist, the upgrades system is within its own menu, which appears on the left side of the screen. The as soon as the player has enough money, a button will become orange for the next available upgrade, and it can be bought. Whilst the other upgrades are visible, so the player knows what they are working towards, they are ordered by cost of upgrade. Each item can only be bought once, although there are duplicate upgrades, they appear in the ordered list of cheapest to most expensive.



Once the player reaches late game, starting over is very quick (with more "angels" that I discussed earlier), and thus they can buy a "quick buy" button. This allows the player to click once to immediately buy every upgrade that can be bought, so they don't have to sit there for ages clicking every buy upgrades button, as there are a lot of them. This system is vastly different to that of Trimps.

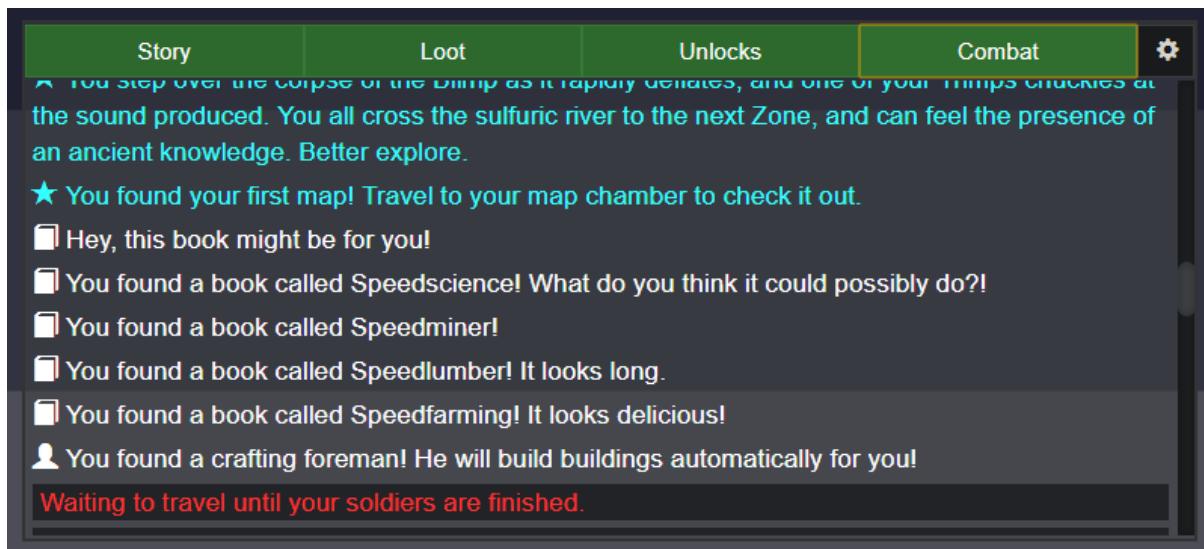
The table below shows the comparisons between the system in Trimps, and the system in AdVenture Capitalist:

Trimp	AdVenture Capitalist
"Bulk buy" - Player can buy an upgrade +1, +10, +25, +50, +100, +max, or +custom number of times. This is applicable for all types of upgrades	"Quick buy" - Player can instantly buy every upgrade that they can afford
"Tabs for different types of upgrades" - The player can switch tabs for each part, out of buildings, jobs, general upgrades, equipment, and then all for everything	

Log system

There isn't a log system on AdVenture Capitalist, so I won't talk about that. In Trimp however, there is an extensive log system that captures story, loot gained (from combat), automatic game saves, and unlocks. It operates using a tab system like that of the upgrades system - one for story, one for loot, and one for unlocks and the last for combat. Any of these can be toggled on or off, allowing the player to only view story and combat logs if they want, or loot and combat. This toggle visualisation changes the colour of the buttons, which makes it easy for the players to see the logs that are being shown, and those that aren't. Different types of logs also have a symbol next to them, and also may have a different font colour, e.g. the story logs have a star and then blue text. This may just be aesthetical, but it makes sense when there are logs generating every second and old logs can be lost quickly. There is also the ability to scroll up and see old logs, although they get reset when the game

is closed and reset. Although it is only minor, I think the ability to clear the logs should be given, although a player could always do that by just toggling all of them off.



Other notes found out about the games

- None of the games have tutorials, or any “easy” starts tell the player what to do. However, there are tooltips when various buttons are hovered over in Trimps. This is something I could have in my game
- Both games have progress bars for visualisation. I looked at some other popular idle games, and all but one use progress bars to show resource count/progression. The one that I saw (Kittens Game) did not have progress bars instead used a simple [number of resources / maximum amount that can be stored] with a rate of increase/decrease counter, and if applicable, a tooltip showing what the factors are for the rate of that resource

Kittens Game by bloodrizer					
					Bonf
catnip	24.09K	/15K	(+0.22/sec)	[Production: +35/sec	
wood	0	/600	(-0.14/sec)	Weather: -15%	
minerals	385.56	/750	(+0.39/sec)	(:3) Village: +10/sec	
iron	35.99	/150	(+0.14/sec)	(:3) Demand: -39.52/sec	
catpower	130.70	/475	(+0.30/sec)		
science	471.68	/4000	(+0.44/sec)		
kittens	10	/10			
furs	44.83		(-0.50/sec)		
ivory	0		(-0.35/sec)		
<u>Send hunters (1 time)</u>					

Findings

My research showed me that even the most basic and intuitive idle games can still be enjoyable to play. Trimps and the Kittens Game are highly popular and both (Kittens Game especially) have very basic GUIs. I can thus take this into account with my game so I won't need to worry about the GUI being too complicated. As seen in Trimps, the player can get around the entire GUI from the main game screen within mostly 1, or sometimes 2 clicks. This makes it easy and intuitive to use.

My research also shows me how I can layout and easily visualise my resource collection, by using progress bars or rate counters. There are also 2 different types of resource collection, but I will have to ask my stakeholders to know which they want to be in the game.

From Trimps, I learnt about the "block" combat mechanic, which is highly useful and can be a factor that is easily upgradable and still have a balanced combat. All I would need to do for wars is increase the enemies' health, attack, and block, so thus the player will have to keep up with their troops, or else they will lose wars.

The log system from Trimps is something that I can take a lot of ideas from, since I find it highly useful to the game, to see what has been happening, especially with any storyline advances. The options to toggle different types of logs is very useful as well, because combat logs fill up a lot of the logs and prevents me from being able to see the other logs without having to spend a long time scrolling up.

Limitations

The issue with bulk buying upgrades, like the system in Trimps, is that if the player accidentally bulk buys 50 of an upgrade, they will lose a lot of their hard-earned resource. Additionally, there won't be any easy way of implementing a refund system, because problems like "What if resources are full, and a refund is made? Then the resources will be lost" and similar.

Additionally, because during the Roman times there were a lot of wars and science/infrastructure advances, I most likely won't end up pushing the player past the end of the Roman era for the game prototype. Instead, I will try to use object-orientated programming that will allow me to build a foundation for any advancements in time that comes with new content, and allows for easy upgradeability.

The portals and run resets requires the player to get quite far into the game, with plenty of progression and late-game technology, which may require a lot of bulk content programming, which may take too much time and is not needed to show off the game prototype. Therefore, I doubt I will include portals and run resets in my game, since the game prototype will never get to the size where I can safely justify going back to the start with certain upgrades/tech.

Second Interview

I will now conduct a secondary interview with my 3 primary stakeholders, Christian, Alex, and Sammy. Based off what I have learned from other games in my research, I will create a more rigorous set of questions that will give me more detail about what the stakeholders want to be in my game.

Questions

I've been doing some research into idle games, and I've identified some new ideas that could be in the game, but also some limitations. I'm going to ask a few questions about what I've found, and your responses to those.

1. For resource visualisation, do you want the game to use progress bars, or a simple [number of resources/max can be stored]? Do you also want the rate of resource gain/loss, how long it takes for the resource to fill to max at the current rate? Or do you find that too much information?
2. There are two ways for resources to be collected; a resource fills up after a certain amount of time, and a certain number of resources are collected (there is no max storage), and a resource fills up by a rate of increase, and a maximum is achieved when it reaches its storage limit (so you cannot have an infinite number of resources). Which do you want in the game?
3. In regards to the “start” date of the game, the earlier the game starts, the more content will have to be added to the game, thus the longer it will take to make. Also, when not much happens over long periods of time (in world history), there are limitations with increasing and decreasing the speed of the game’s progress. Based off this, should the game start at 4000BC (and have the game run fast but learn lots of stuff), or in the Roman Times (when there was decent infrastructure and lots of war action)?
4. All the idle games like Trimps that I've seen start with very few resources, and slowly build up as the player progresses through the game, or does research/science/tech and discovers new resource types. I also wish to keep close to reality, but with more “general” terms of resource, e.g. instead of bronze, copper, zinc, just have metal. Knowing this, and the limitation of having too many resources, which 4 or 5 of the following basic resources should I start with? Metal, stone, food, wood, population, research?
5. What do you want the log system to feature? This could be story line, combat logs, loot gained, year/date, auto-saves, or any other. Do you want them to be toggle-able? Should there be a visual indication for this? Should there be an option to view all the logs on one page? Should players be able to clear their logs?
6. Seeing as every other idle game I have played/seen has an offline progress system, do you want my game to also have one? If so, what should progress whilst the player is offline? Just resources? Time?
7. How often should the game auto-save?
8. How do you want the combat system to look? Like the one in Trimps? Instead of “zones”, each grid set could represent a war, and yours/your enemies’ progress through it? Maybe have enemies start on one end as red, and allies start on the other end as green?
9. Should I include the “block” mechanic in combat (from Trimps)?
10. Do you think the wars should get harder within the wars, or harder per war?
11. What should the bonuses/penalties be for winning/losing wars?
12. How should the early game content be introduced (in a tutorial-like fashion)? Using message boxes? The log system?

13. For the sake of realism, do you think there should be “shady” deals that may complicate the game further?

Sammy's Response

1. Progress bars would be nice. Perhaps a little number beside it (or by hovering over it or clicking it) to denote the other stats
2. Resources should be capped so you're not encouraged to just do nothing, perhaps you can upgrade your max too for a sense of progression
3. I'd be fine with a shorter period around the Roman times but with a lot of depth/options
4. Wood & Stone is a definite, gold for purchasing, perhaps start off with “low quality” iron which you can upgrade with better forging techniques to make it tougher and more effective. Population from armies also is a definite
5. Perhaps each log message should have a category (e.g. DEBUG, STORY, and INFO etc.) and you can toggle which categories are visible, similar to the Minecraft log system. Also the option to export all the logs to a text file would be nice
6. Resources go up, but I'd say a little bit slower than if you were actually there
7. Often. Perhaps even every major decision
8. I think it should just be a very simple sort of thing, with your troops starting on the green and enemies on the red, perhaps some currently neutral white ground in between
9. Yes
10. Per war
11. The bonus for winning is loot & resources, and if you lose you lose some resources
12. Assuming early game, you have a few options so should be explained as you go on. For testing purposes, please add a way to “skip” all tutorial features
13. No

Alex's Response

1. Progress bars, rate of resource gain/loss, not time it takes for resource to fill up
2. Resources fill up by a rate of increase, and a maximum is achieved when resources fill up
3. Beginning of Roman Era (27 BC)
4. Food, metal, stone, wood, science
5. Combat logs, loot and auto-saves (year should be on another indicator). Should be a minimisable window if possible, all logs on one page is too much info (have tabs that separate them). Clearing logs would also be useful
6. Yes, there should be an offline progress system, but only resource collection progresses
7. Every 2 minutes and before closing
8. Grid of battles where each one is fought by both sides. Most battles won wins war
9. Yes
10. Harder per war
11. Production boost/penalty for the current age
12. Log system
13. No

Christian's Response

1. Resource visualisation is always good, maybe hide in a menu so it isn't too cluttered
2. Limited so you have to upgrade rate of collection and storage
3. Roman times, already plenty of progression to be had
4. Stone, metal allow, population (troops), wood and food. They can also be upgraded to an advanced resource which is more efficient. Research just requires materials for "prototyping"
5. Combat logs, story line logs to show the current year and loot gained
6. Definitely. Have an upgrade called motivation which is a multiplier starting on 0 which multiplies the production of population when their leader is present
7. Whenever the program is closed. It won't be open for more than a few minutes if it is an idle game anyway
8. Combat like Trimps
9. Yes
10. Easier within the war as resources get low. Harder between wars
11. Depends on how much you lost by e.g. men left
12. Toggle-able message boxes on new features, such as to not annoy veteran players
13. No

Stakeholder Requirements

Feature	Proposed Solution
Resource collection	<p>There will be 5 resources:</p> <ul style="list-style-type: none"> • Population • Food • Wood • Stone • Metal <p>These will be the 5 resources throughout the whole game, but will be able to be upgraded as the game progresses. Research will be a separate part which requires resources to make for "prototyping" which thus allows the player to buy upgrades, using science plus varying resources depending on the upgrade type.</p> <p>The population will also use food, so the rate of food increase will also be decreased by this (forcing the player to buy more food workers)</p> <p>The player can also manually increase resource rate (but only one resource at a time) by toggling a button for a resource to increase the rate by 10%.</p> <p>The resource visualisation will be using progress bars, with the rate of increase and the number of resources underneath it. It will use a resource collection system like that of Trimps' and Kittens Game's – capped and resources fill up over time. There will also be a milestone system that changes each main unit from a large string of zeros to a "K" or "M" (for thousand or million). E.g. 4,850 will become 4.85K.</p>
Upgrades	<p>There will be 5 types of upgrades:</p> <ul style="list-style-type: none"> • Storage – this will cost wood, and increases the amount of resource space • Workers – this will cost food, and increases the rate of resource gain • Housing – this will cost stone, and increases the number of workers and troops the player can have

	<ul style="list-style-type: none"> • Combat equipment – this will cost metal, which will be used to increase the health, attack and block of the player's troops • Science/tech – this will cost science, and unlocks various new housing types, combat equipment and upgrades worker speeds and maximum storage capacity
Starting era	Roman empire, because there was already a decent amount of tech, and a lot of progress of tech. There were also quite a number of major wars which my game can follow
Logs	<p>The game will have logs which track and help the player progress through the game. The logs will have 5 different sections:</p> <ul style="list-style-type: none"> • Combat logs – records which wars have been won or lost, and other war-based statistics • Story line – shows the current year, and the other story based lines • Tutorial logs – If the player wishes to, they can show the tutorial logs which explain the game step by step, when various progression milestones are made • Auto-save logs – Shows a little message when the game is auto-saved • Loot gained – When wars are won or lost, this log will show what loot is gained, and what bonuses and penalties winning/losing wars will have on the player's resources <p>All of the logs can be toggled on/off, so the player can only show the tutorial logs, or story line and combat logs etc. There will also be an option to clear each section of logs separately. Logs can also be saved to a text file</p>
Offline progress system	The offline progress system will only progress resource collection, until the storage is full
Saving	The auto-saves will happen every 2 minutes, and a message in the auto-saves section of the logs will show. The game can also be saved manually by the player, and they will have to load the games manually from file when they load the game back up. All auto-saves will save data into a text file
Combat	<p>There will be a grid, and when a war is started, the allies will start at the bottom on green tiles (owned land) and the enemies will start at the top on red tiles. The side that wins the most battle after a set amount of time wins the war. The set amount of time is the length of time the war lasted for in real world history.</p> <p>Combat will consist of 3 main mechanics:</p> <ul style="list-style-type: none"> • Attack – The amount of damage the unit does to the enemy, regardless of how much it blocks • Health – The health of the unit, separate to block • Block – How much attack the enemy can “absorb” before taking damage. E.g. enemy does 30 attack, but player has 25 block, so only takes 5 damage <p>The combat is turn based. So the player attacks first, and then the enemy attacks. There will be an upgrade to automatically attack (when player is online) so that they don't have to press a button to send more troops every time they lose a battle</p>

Hardware and Software Requirements

Software Requirements	Justification
Windows 7 or later versions	Since the game will be made using the Microsoft development SDK for C#, it utilises the internal operating system DLLs. If the player tries to play the game on another operating system, the program won't have access to these DLLs thus won't run.
.NET Framework 4.4 or later version	The framework that C#.net uses

Hardware Requirements	Type	Justification
Mouse	Input	To move the pointer/cursor around the screen and click on elements in the game
Keyboard	Input	To activate various key-binds or type name into game etc.
Monitor	Output	To see the game
<ul style="list-style-type: none"> • CPU: Intel Core 2 Duo E8400 2.5 GHz or AMD Athlon 64 X2 6000+, 3.0GHz or higher • RAM: 500MB or higher • GPU: None required • 500MB HDD or SDD or higher 	Processing	If the player cannot meet this spec level, they won't be able to run the game without it crashing or lagging out

Success Criteria

I have different sections because I want to be able to go back to my success criteria easily by referencing the section code. Some of my criteria will have the “*Optional*” label because I can come back to do them later if I have time.

Criteria	How to evidence criteria being met	Section Code
Section A: Resources/upgrades system		
Resources go up over time	2 screenshots – first one taken before, second one taken after to show progress with resources	A1
Resource rate increased through upgrade	Screenshot of increased rate	A2
Resources reach “milestone” where the number shortens	Screenshot of 4,000 resource shown has 4k resource	A3 <i>Optional</i>
Button to manually increase rate of resource collection increased, button turns brown, all other buttons turn grey and toggle off, that resource rate increases by 10%	2 screenshots – first one taken before with one button pressed, second one taken after with another button pressed to show increased rate and only one rate bonus allowed at a time	A4 <i>Optional</i>
[Resource] storage upgrade is bought	Screenshots to show max [resource] capacity increased	A5
Worker for [resource] upgrade is bought, not enough housing space	Screenshot to show “Not enough housing space” in tutorial logs	A6
Worker for [resource] upgrade is bought, enough housing space	Screenshot to show increased [resource] rate	A7
Housing upgrade is bought	Screenshot to show housing number increased	A8
Science upgrade is bought, but not enough science points available	Screenshot to show “Not enough science to buy this” in tutorial logs	A9
Science upgrade for combat is bought, new combat upgrade button appears in combat tab	Screenshot to show new upgrade button	A10
Different upgrade types tabs selected	Screenshot to show only worker upgrades shown in workers tab	A11 <i>Optional</i>
Combat upgrade to increase block bought	Screenshot to show troops’ increased block	A12
Enough resources are available to buy upgrade	Screenshot to show upgrade button turning to “clickable” state	A13 <i>Optional</i>
Section B: Save/load system		
Game is auto-saved every 2 minutes	3 screenshots – first one taken when auto-save happens, second one taken 2 minutes later when second auto-save happens, third taken of auto-save file changed (see time stamp in file)	B1

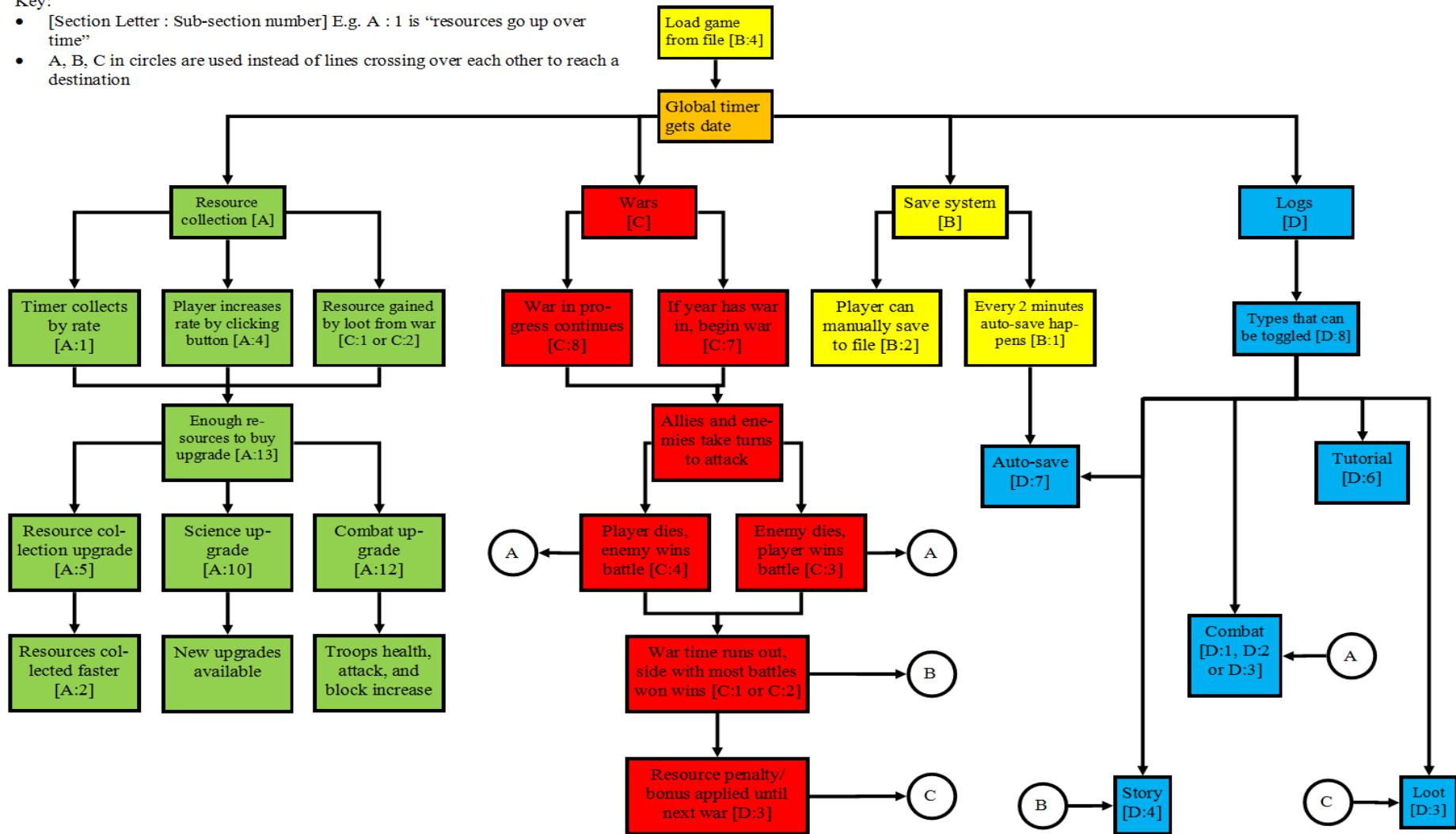
Player manually presses the save game button	Screenshot of new save file created	B2
Game closed	Screenshot of auto-save file changed	B3
Game loaded up, player selects save file to load, message box tells player how many resources were made when they were offline	Screenshot of before game closed, screenshot of after game reopened and save file loaded, screenshot of message box	B4
Section C: Combat/war system		
War is won by player	Screenshot of resources rate gain, and screenshot of grid reset	C1
War is lost by player	Screenshot of resources rate loss, and screenshot of grid reset	C2
Battle is won by player	Screenshot of player gaining green tile from enemy's red tile	C3
Battle is lost by player	Screenshot of enemy gaining red tile from player's green tile	C4
Player's army with 170 health and 50 block gets hit by 100 attack, so health goes down to 120	Screenshot before and after turn to show health/block/attack works	C5
Number of troops increased, total army health increases by (current upgrade of health) * number of new troops	Screenshot to show the stats of army increased	C6
New war started when right year reached, enemies have higher health/attack/block than in previous war	Screenshot to show stats in old war, and then stats in new	C7
Game loaded up, war in progress continues as before	Screenshot showing war before game close, after game close	C8
Section D: Logs system		
A battle is lost or won, show the message in the combat logs	Screenshot of combat logs	D1
A war is lost or won, show the statistics in the combat logs	Screenshot of combat logs	D2 <i>Optional</i>
A war is lost or won, logs show resource penalty/bonus until next war	Screenshot of loot logs	D3
A tech upgrade is made, and story progresses, show message in logs	Screenshot of story logs	D4
A new year begins, print new year in logs	Screenshot of story logs	D5
Player battles for first time, buys first upgrade or reaches first resource milestone, print	Screenshot of tutorial logs	D6

various tips or explanations in tutorial logs		
Game is auto-saved or manually saved by user, show message in auto-saves logs	Screenshot of auto-save logs	D7
A log type button is clicked to toggle it off, button changes from green to red and those logs stop showing	Screenshot of log toggle buttons and logs to show right logs toggled off	D8
The clear logs button is clicked, all logs toggled on cleared	Screenshot of nothing in logs	D9
Save logs to file button is clicked, all logs saved to a text file	Screenshot of file it saved to	D10 <i>Optional</i>
Section E: Usability		
The game balanced so that it can't be progressed really quickly (amount upgrades cost, how much the affect various parts of the game etc.)	Screenshots of player being able to keep up with enemies' combat stats, to show balanced progression through time	E1 <i>Optional</i>
Intuitive and easy to use menu	All of the game menus can be traversed within 2 clicks of the main game screen. Screenshots showing each menu	E2
Game does not have low framerate	Screenshot of CPU load reduction when game closed	E3
Big buttons, some colour coding to show toggles and being able to be clicked	Screenshots of buttons changing colour/position when various buttons pressed	E4 <i>Optional</i>

Abstraction Diagram

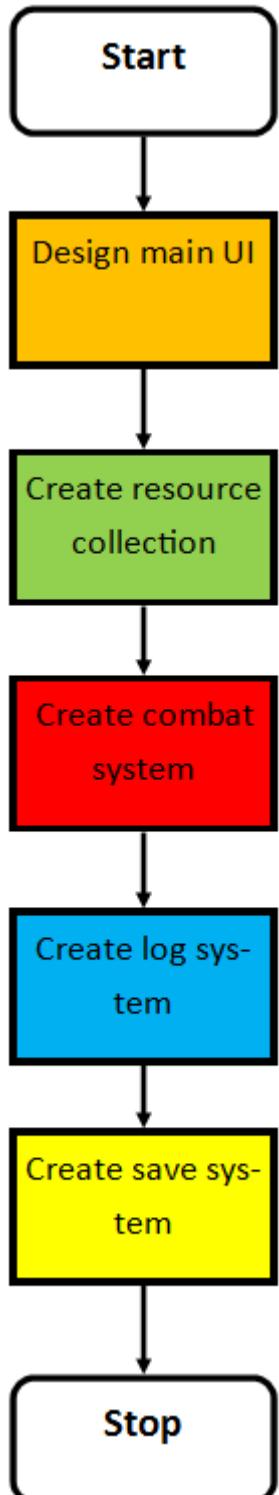
Key:

- [Section Letter : Sub-section number] E.g. A : 1 is “resources go up over time”
- A, B, C in circles are used instead of lines crossing over each other to reach a destination

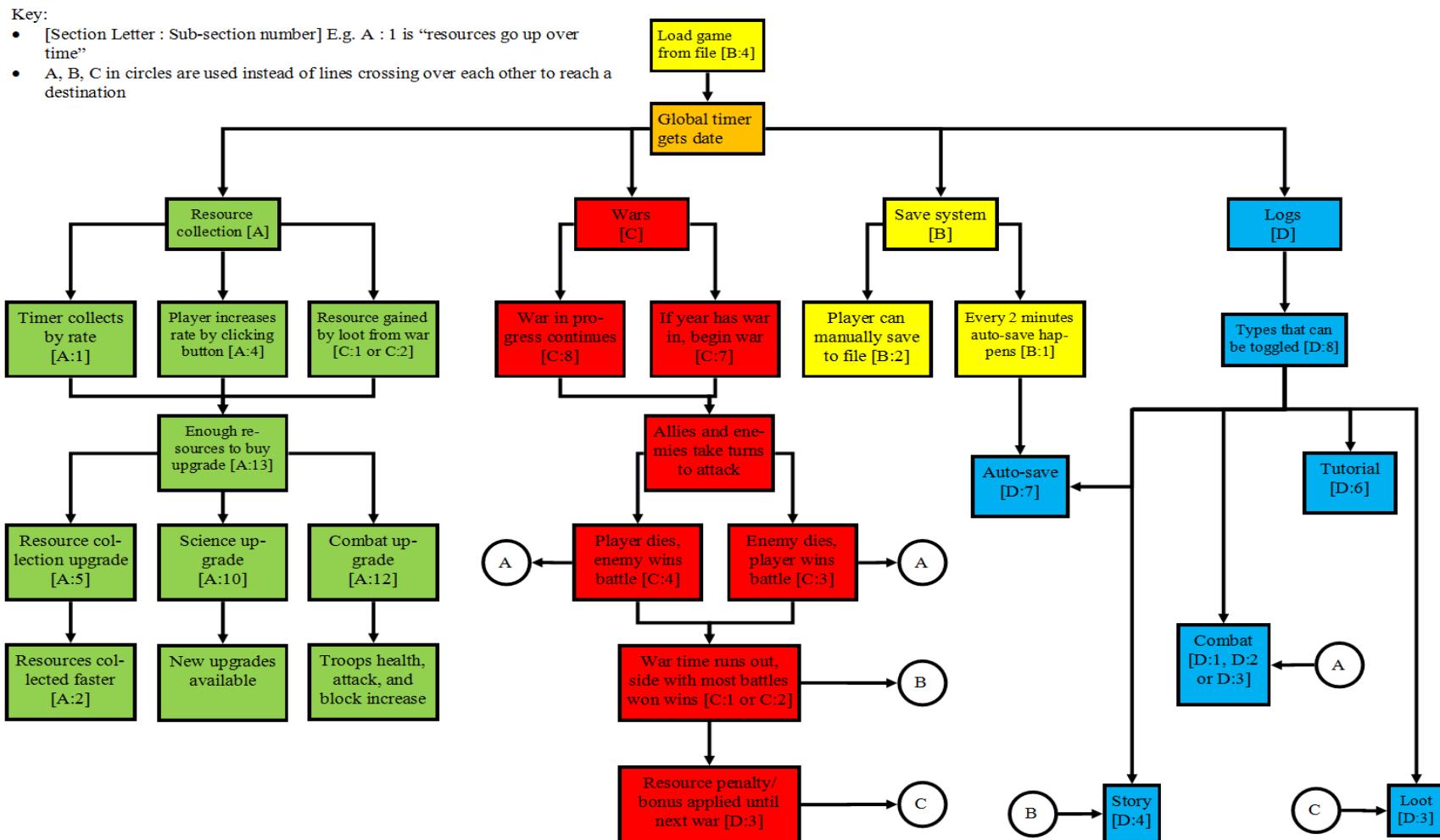


2. Design

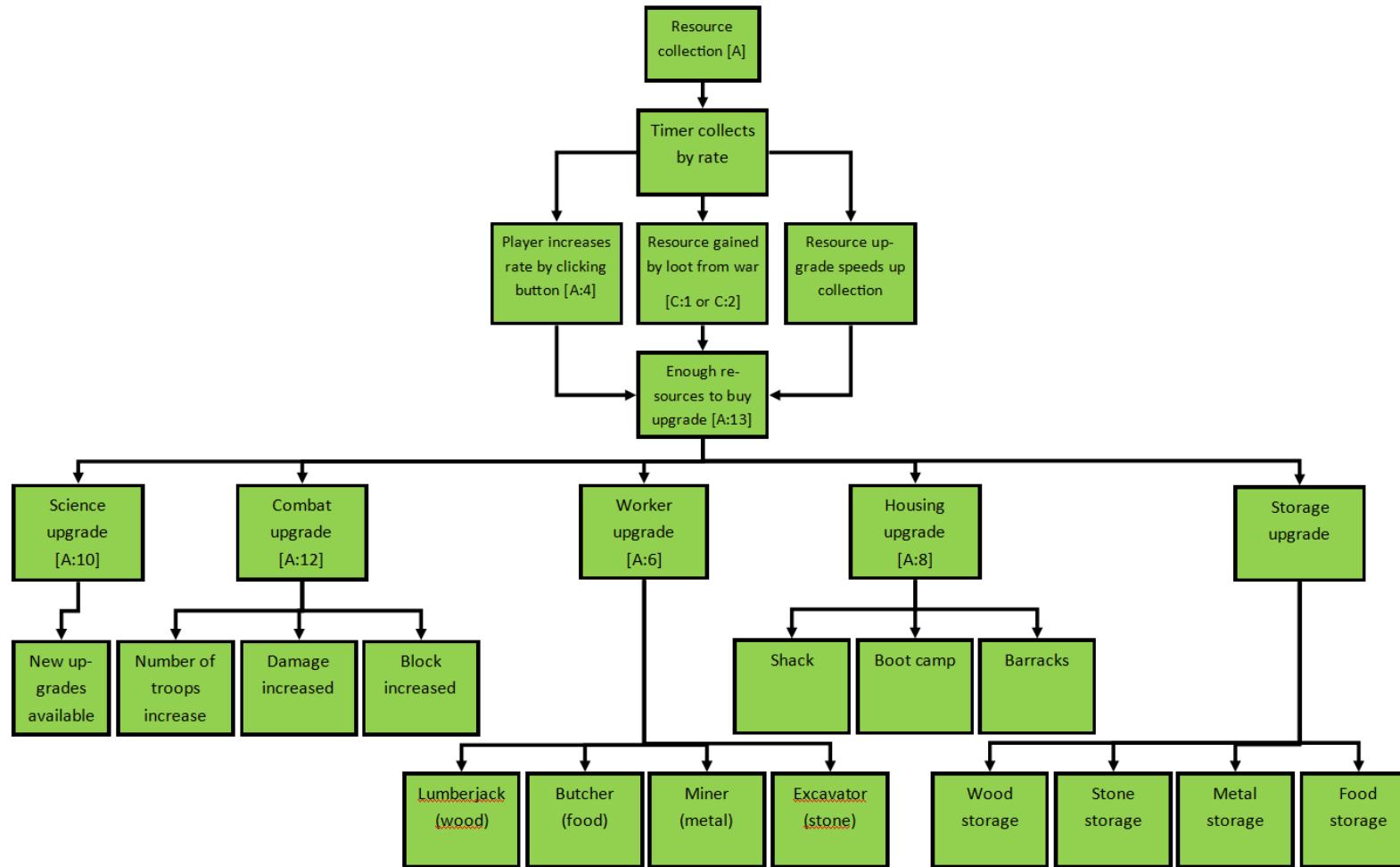
Problem Decomposition



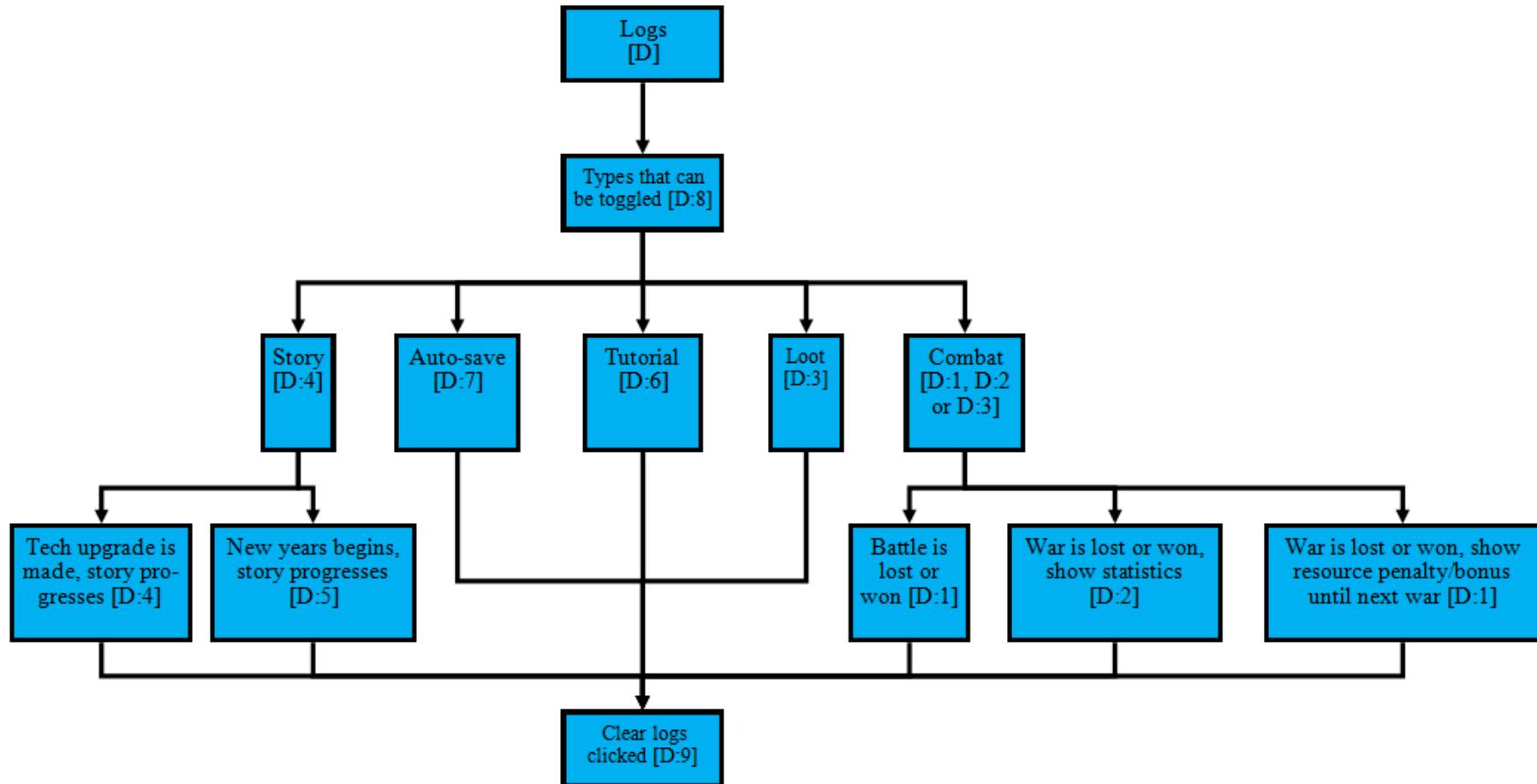
Here is the abstraction diagram. The colour codes match with the program creation structure flowchart as shown above. Most boxes have a section and/or subsection number, which represent which part they are in the success criteria.



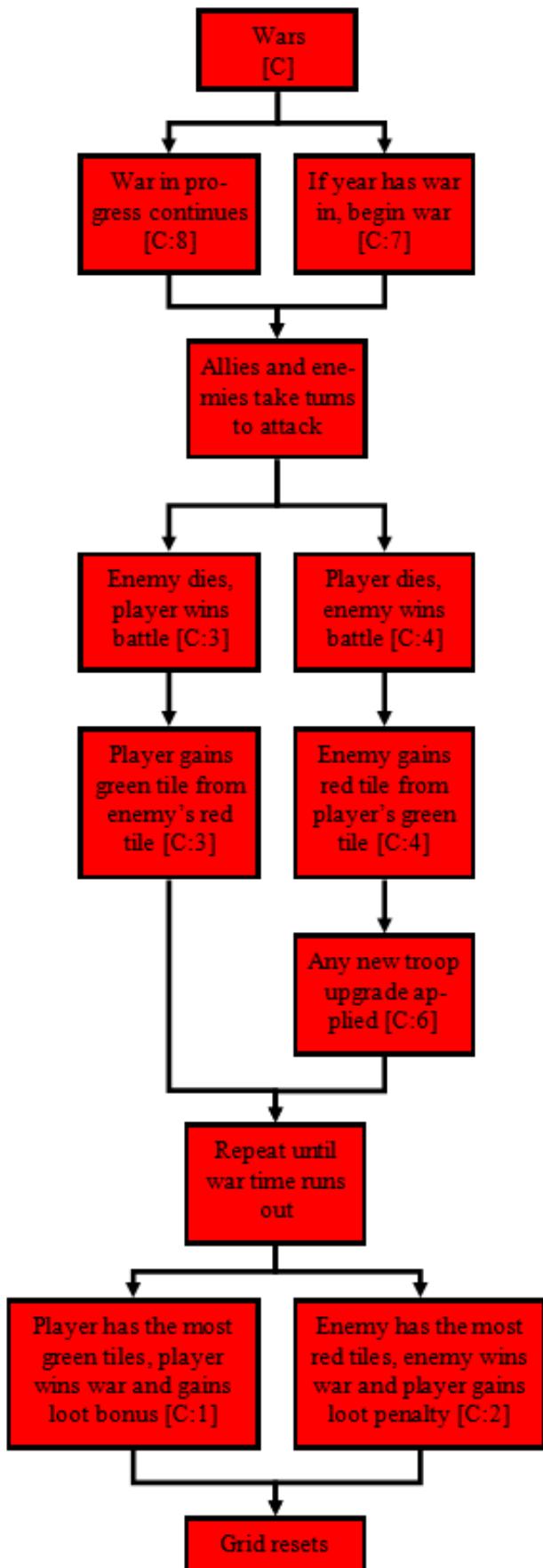
Resource Collection decomposition diagram



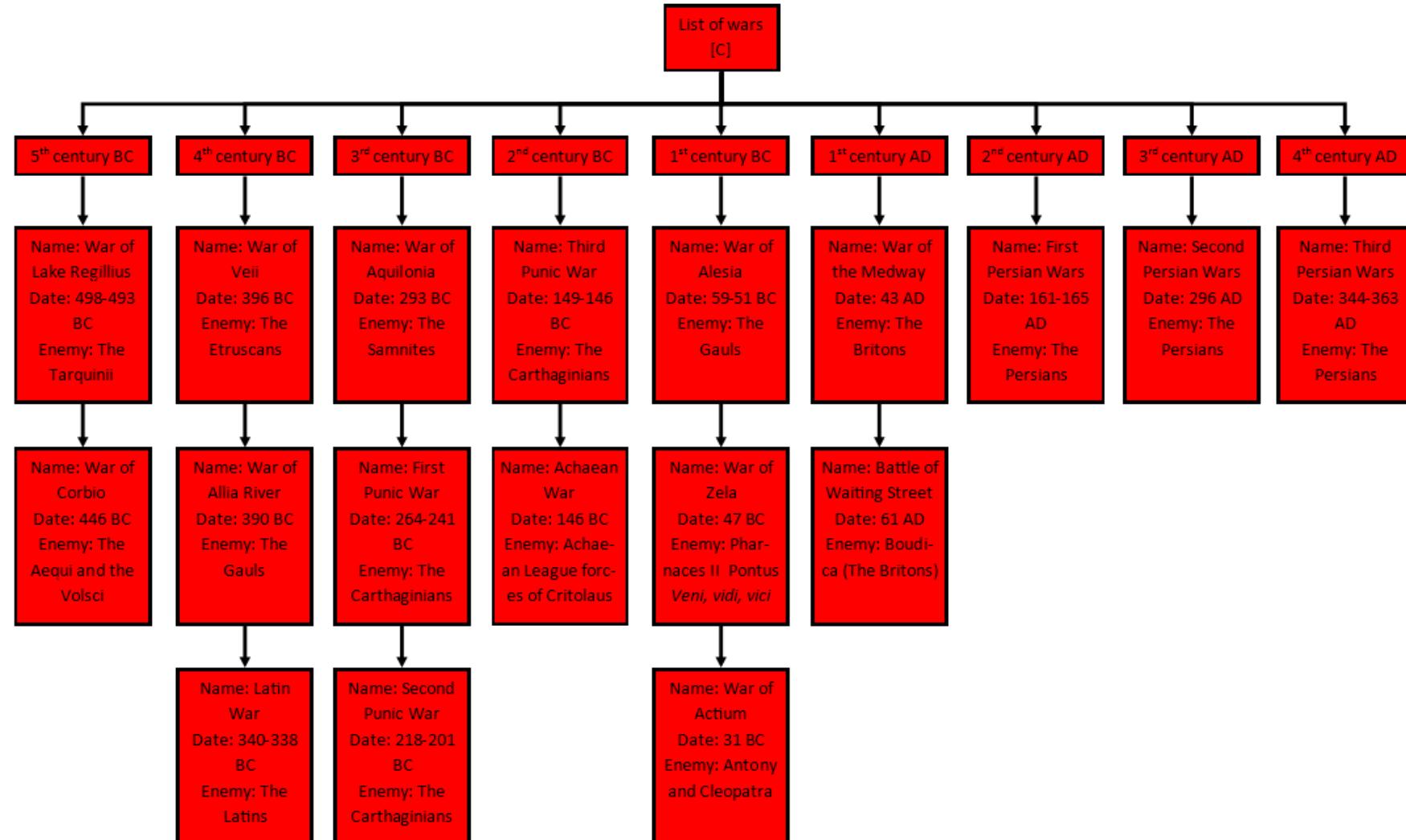
Logs decomposition diagram



Wars decomposition diagram



List of wars

[Save / Load decomposition diagram](#)

File Structures

Each line in the below table represents a line in the save file. Each item will be separated by a # because then the program can go through the file and check for each #, and thus pull out/put in each relevant part of the file.

Wood amount	Stone amount	Food amount	Metal amount
Wood rate	Stone rate	Food rate	Metal rate
Wood capacity	Stone capacity	Food capacity	Metal capacity
Wood gather multiplier	Stone gather multiplier	Food gather multiplier	Metal gather multiplier
Housing amount	Housing remaining		
Troop block	Troop attack	Troop health	Troop amount
Number of tiles held by player	Number of tiles held by enemy		
Wood storage upgrade cost	Stone storage upgrade cost	Food storage upgrade cost	Metal storage upgrade cost
Wood workers upgrade cost	Stone workers upgrade cost	Food workers upgrade cost	Metal workers upgrade cost
Science upgrade cost			
Shack housing cost	Boot camp housing cost	Barracks housing cost	
Shack housing increase	Boot camp housing increase	Barracks housing increase	
System time			

The rate will be calculated by taking the base rate number (in milliseconds) and multiplying it by 1000 to simulate the 1 second interval. The reasons I will multiply it by 1000 afterwards rather than having the base rate number in 1000s in the first place are:

1. The file size will be reduced by a few bytes (not the main issue though)
2. The rate is more readable because the rate will always be a small number even when upgrades increase it a lot – but if it is already in the 1000s, then it will be much harder to show the really long number in the menu when the player views the rate. E.g. starting on 1, it starts as 1/s, but if it starts on 1000, it starts at 1000/s. Imagine an upgrade that increases rate 1000x. Then the rate will be shown as 1,000,000/s, which could be very annoying to try and automatically fit into the window

Of course all of these variables will just be numbers separated by the hashtags – e.g. first line will be 10#30#20#15

There are a lot of variables here, but all will be necessary. When loading a file, each variable will just have to be read into the global data variables (a class where all of the variables that are needed in multiple classes that can be accessed and changed to/from). When saving to a file, all of the parts will be saved from the global data variables at their current point of saving. The game will most likely have to be “paused” where the timer is disabled so that when the game is saving variables to a file, the values are all saved from the exact same point of time.

The system time is needed to be added because the offline progress system requires the time of save to determine how much time has passed before working out how many new resources have been collected since the program was closed. E.g. the rate for wood is 1/second, there is 10 in storage when game is closed (thus saved as it saves before close). The game is closed for 1 minute. When the save game is loaded, the “offline progress” should be rate * seconds offline. Therefore in

this case, because there are 10 already in storage, it will be $10 + (1 * 60)$, so the amount in wood storage now is 70.

Population is a “resource”, but it is controlled by other parts. The population amount is the same as housing amount (number of people in housing), the population only increases by buying more workers or troops, and the population capacity is the same as (housing remaining - housing amount). The workers’ amounts are worked out using the rate; if a worker increases rate by 2 and the base rate is 1, then number of workers = rate - base rate. The same does not however apply to the troop amounts; the army health, block and attack are worked out through knowing the number of troops there are. E.g.

1. Army health = troop health * troop amount
2. Army block = troop block * troop amount
3. Army attack = troop attack * troop amount

Graphical User Interface Design

Main

The GUI will consist of 5 menu strip tabs within the same form:

1. Resource collection [Section A]
2. Upgrades [Section A]
3. Combat [Section C]
4. Logs [Section D]
5. Save/Load [Section B]

These will each have their own UI elements in to navigate the game. There should not be more than 2 or 3 clicks between getting to or from one point of the menu to another. This will be achieved by using a menu strip. Menu strips are user interface elements that appear as a task bar of buttons at the topic of the menu form, which on clicked leads to different parts of the game. This leads into two slightly different ideas:

1. There are sub-buttons that list when the user clicks the main button on the menu strip. Therefore, the user could click the main button (e.g. logs) and then the list of sub-buttons appears (e.g. auto-saves, story, logs, combat, and tutorial). The user then clicks one of these buttons, and the appropriate panel for that part appears. This reduces the number of clicks to get around the entire menu, but there is less on screen and thus more clicks overall are needed to play the game

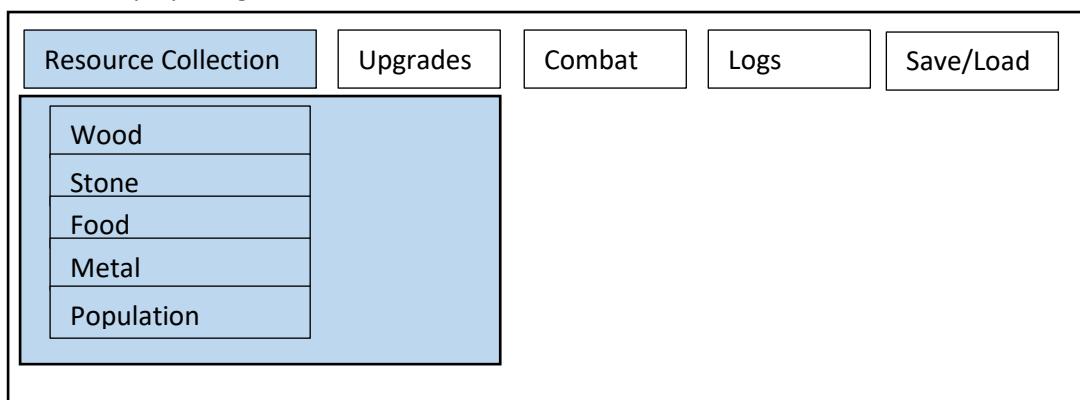


Figure X (above) shows the dropdown of sub buttons of the resource collection tab

2. There is just the first layer of buttons on the menu strip, which when clicked, just shows the panels from the start. This takes more clicks to get to, however fewer clicks are needed to view the same number of parts of the game.



Figure X (above) shows the menu strip without the sub buttons

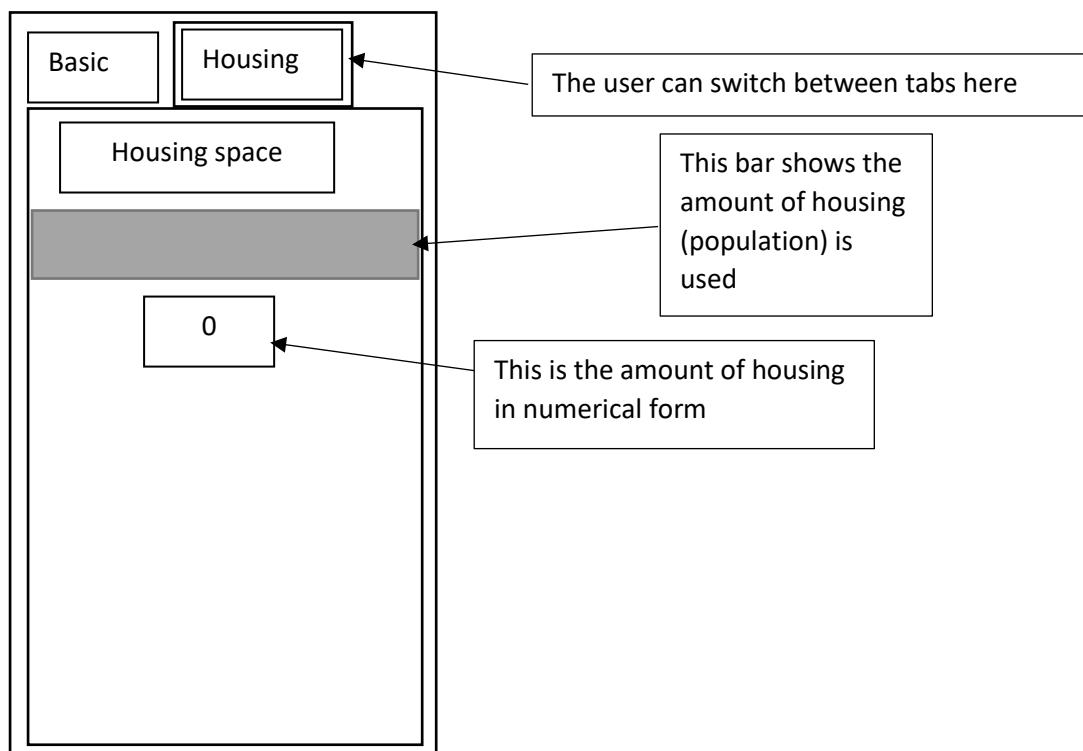
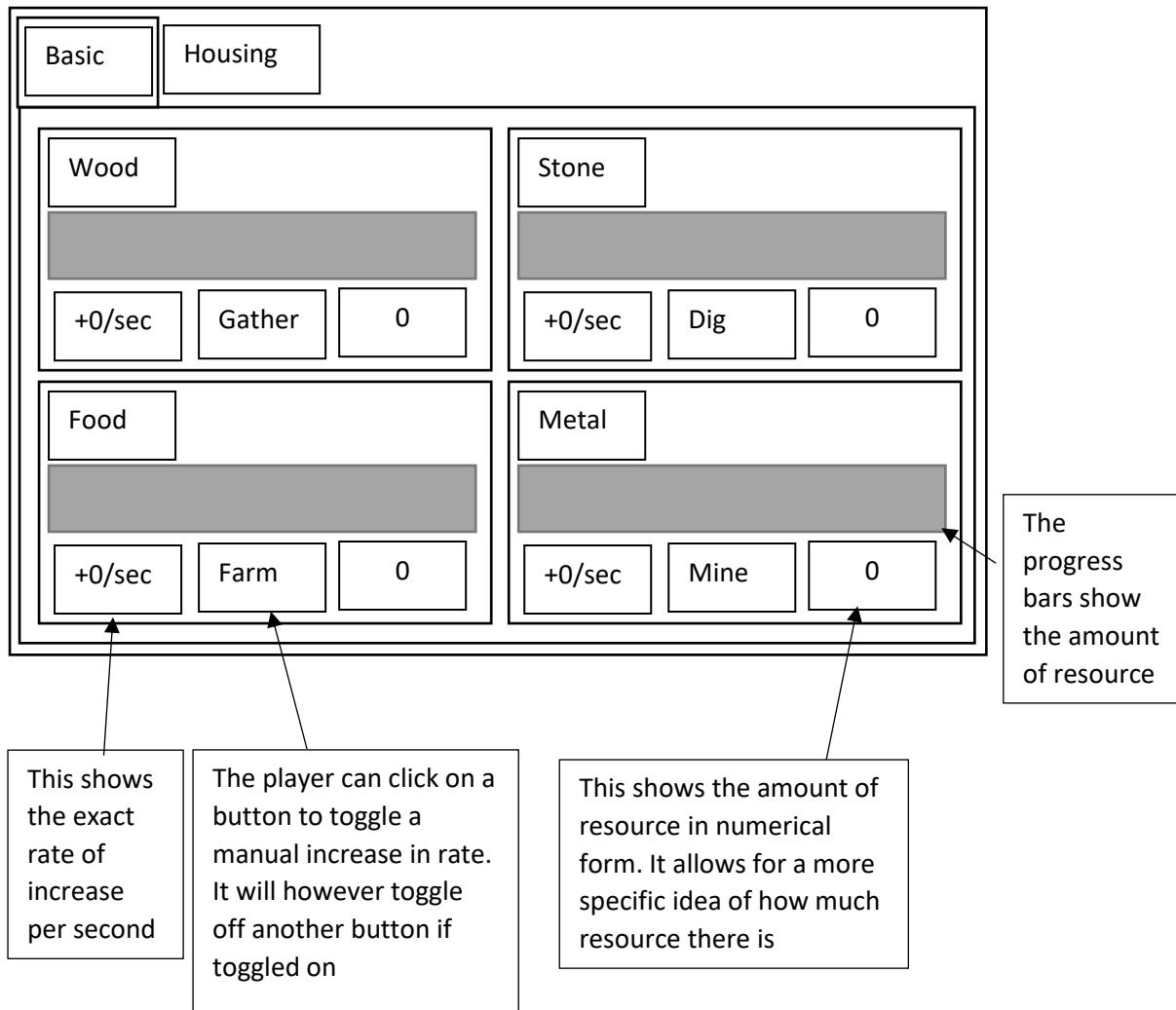
After some talk and showing examples with my stakeholders, they have chosen to go with option 2, because they want to view more on one screen.

Resource collection [A]

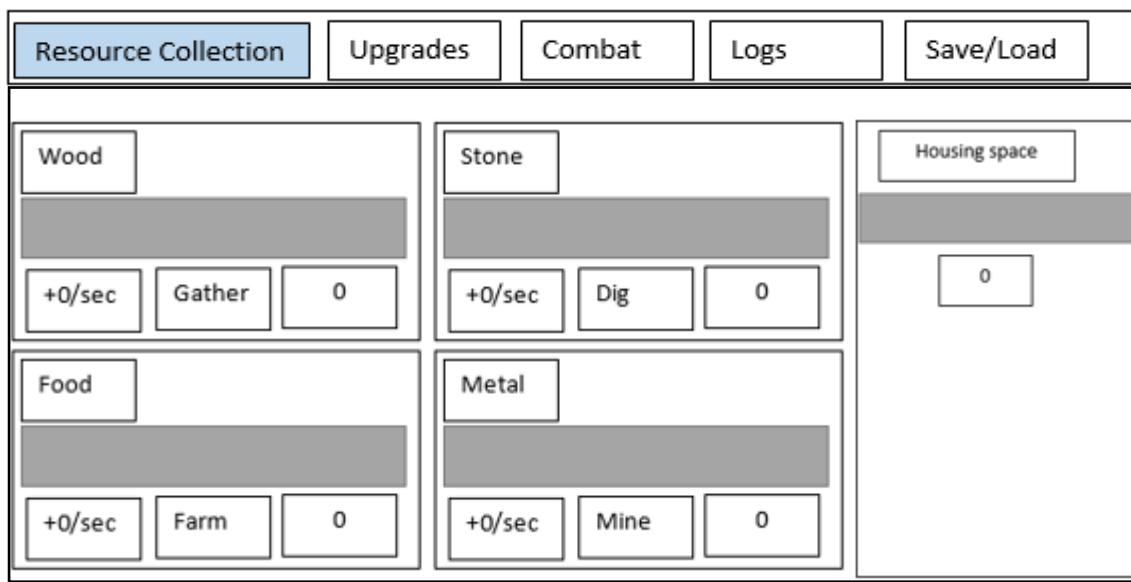
I spoke to my stakeholders and they want to have two tabs inside the resource collection menu:

1. A tab that shows wood, stone, food and metal
2. A tab that shows housing (population)

When designing, I already had something in my mind. The Trimp's layout is very intuitive and after talking to my stakeholders, I learnt that they also find it useful and they think my design should be based around it.



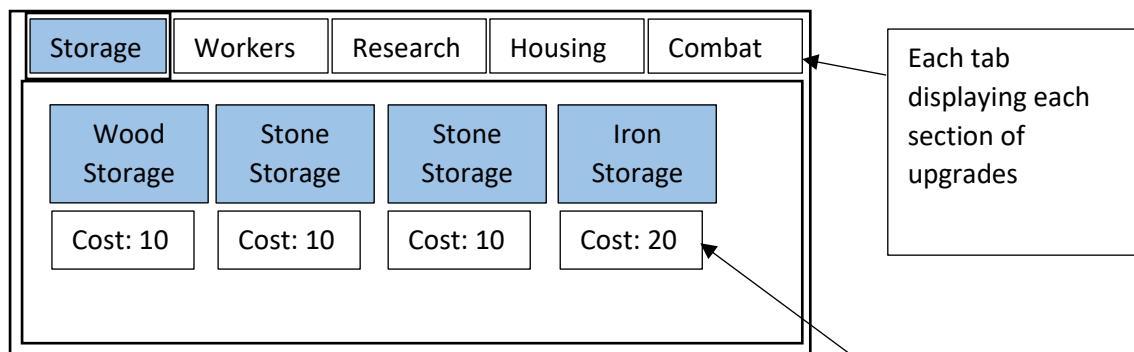
When I showed these designs to the stakeholders, all but one said they found it intuitive and useful to read. When I asked the one why they do not like it, they said, “I think the housing space should be on the same tab as the other resources.” The upside to having housing space in a different tab is so that more “special” types of resources can be added later if the game is expanded. However, on closer inspection, I realised that the player will have to switch tab frequently just to view how much population they have and there is still a lot of space on the sides, so I iterated the design to only have one tab. I checked with my other stakeholders and they realised how much they would be clicking between tabs so changed their minds as well. Here is the new design:



Upgrades [A]

Because I was unsure of what to do for my upgrades collection interface, I asked my stakeholders what they wanted. After some discussions, we came up with 2 ideas:

1. Have all of the upgrades in one menu, arranged in a table with storage at the top, followed by workers, research, housing and then combat. There were however 2 downsides with this:
 - a. When milestones are reached, new upgrades will appear which means I will have to code a system to realign the upgrade buttons once they appear or disappear
 - b. There is not a very clear approach to the ordering of the upgrades and the player may get confused
2. Split the upgrades up into different tabs for each section – storage, workers, research, combat & housing. My stakeholders also want an “All” tab which displays all of the upgrades in one panel but that is a low priority task.



Save/Load [B]

The saves menu should be as simple as possible. I want to have 4 buttons:

1. Save current – Saves the current game to a file
2. Load game – Brings up a file explorer to load from an existing save file
3. Toggle auto-save – Allows the user to turn on or off the auto-saving every 2 minutes
4. Delete current – Deletes current save file. This will bring up a separate form in the middle of the screen asking if the user really wants to do this

I will also display the current time (in UTC) because it is useful information, and is going into the save file for the offline progress system anyways. When the player clicks the autosave option button, a label to the right of it temporarily shows either “Autosave interval: 5 minutes” or “Autosave off”, depending on which way it is toggled. The load game button on clicked will show a new form which will allow the player to select one of the files stored in the file location of the save files. When the file is selected, on the player clicks the submit button, the form will close and the new save is loaded (the current game in progress is of course autosaved).

Save Current	Delete Current
Load Game	
Autosave Option	Autosave interval: 5 minutes
The current time (UTC) is 18:00	

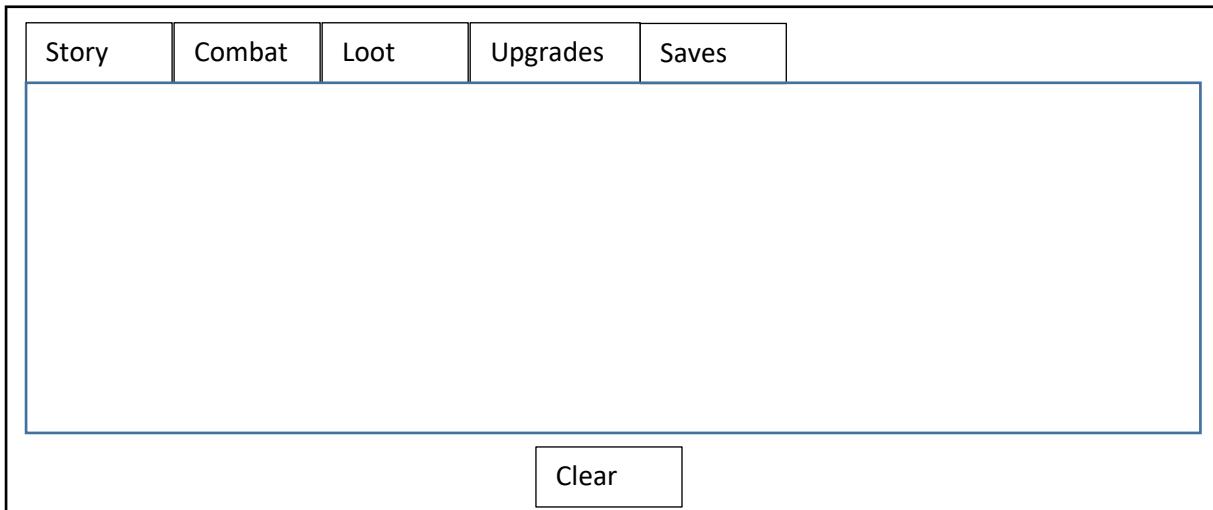
newGame1.txt
myGame.txt
thebestsave.txt
anewworld.txt
<input type="button" value="Submit"/>

Logs [D]

The logs will have the 5 tabs:

1. Story
2. Combat
3. Loot
4. Upgrades
5. Saves

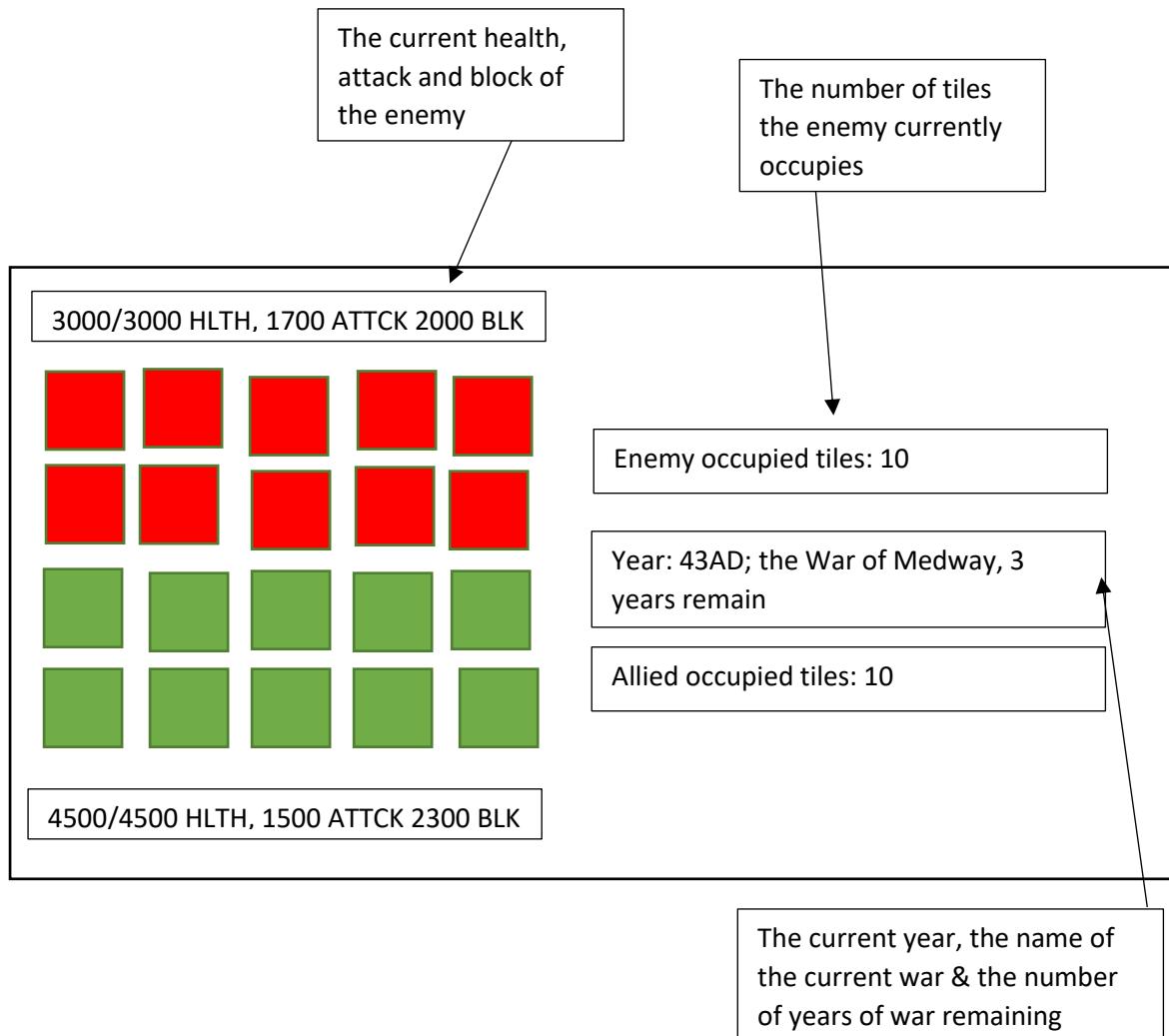
As stated in the success criteria, I will have a clear button that on press, will only clear the logs for each independent tab (D9).

**Combat [C]**

My stakeholders want a grid, and when a war is started, the allies will start at the bottom on green tiles (owned land) and the enemies will start at the top on red tiles. The side that wins the most battle after a set amount of time wins the war. The set amount of time is the length of time the war lasted for in real world history.

Criteria of menu:

1. 10 green tiles at the bottom of the grid (starting point)
2. 10 red tiles at the top of the grid (starting point)
3. The current year and the year the war ends
4. The current health, attack and block of the allied and enemy troops
5. Other basic info like how many tiles player's army/enemy have



Inputs Table [TC]

Input name	Type	Explanation
Change menu	Menu strip	Buttons on the top of the screen to choose which menu the player wants to go to
Increase resource rate	Toggle button	A button is toggled on, toggles off every other button to make sure only one is on at a time. Increases rate
Change upgrade type	Tab control	Player selects upgrade type tab to get to the correct upgrade buttons
Upgrade item	Button	Player clicks button to buy an upgrade once
Save current game	Button	Player clicks button to save the current game to file
Load game	Button	Player clicks button to load a game from a file
Autosave option	Toggle button	Player toggles button to change Autosave from 5 minutes to off or vice versa
Delete current	Button	Player clicks button to delete current game
Confirm deletion	Button	Player clicks button to confirm current game deletion
Cancel deletion	Button	Player clicks button to cancel current game deletion
Change logs type	Tab control	Player selects which logs they want to see by selecting the correct tab
Clear logs	Button	Player clicks button to clear the logs from the selected tab

Sub-program design

I have chosen to use the Functional programming method, as I want to avoid shared state, mutable data and side-effects of calling functions with the wrong sets of parameters. I will have 7 files for each part:

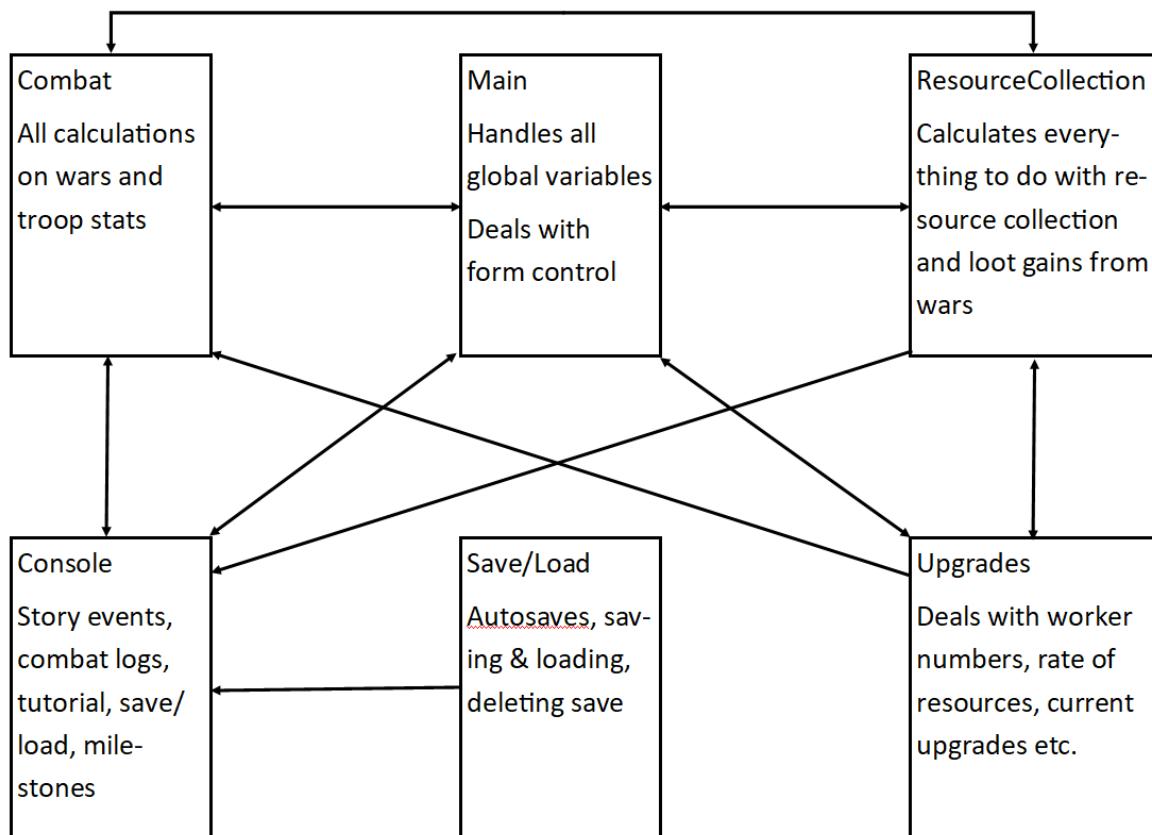
1. MainForm.cs
2. GlobalData.cs
3. ResourceCollection.cs
4. Upgrades.cs
5. Combat.cs
6. Logs.cs
7. FileHandling.cs

However, these are not 7 different classes. There will actually only be 2; MainForm.cs and globalData.cs – the reason for this is that my stakeholders want everything to be in one form only. Therefore, my solution is to use partial classes. A partial class (in C#) allows a class to be split over multiple files, but still keep all of its properties. This then allows me to split up each section of the program into these 7 individual files.

1. MainForm: this class will have the main form control code (close button, maximise, minimise, change look of form etc.)
2. GlobalData: this class will hold every global variable that needs to be accessed by the MainForm class
3. ResourceCollection: this will be for calculating everything to do with the four resource collection parts. This includes taking in the button presses when a user wants to manually increase a certain resource, although that is the only input that is required by the user for that part. When a war is won or lost, the combat partial class will send the loot values won or lost from the wars to this which deals with numbers accordingly
4. Upgrades: This handles all inputs, processing and outputs in regards to the upgrades menu, like a new upgrade becoming available, costs changing when an upgrade is bought, and increasing troop health when the troop health upgrade is bought. It also handles all processing to do with changing the amount of housing that is available, and increasing it when a housing upgrade is bought. It does not increase the rate of resource, but it does increase the number of workers for that resource when a worker upgrade is bought. This is responsible for many factors in the game, so is a vital class, only second after main
5. Combat: This is responsible for calculating when wars need to start and end, when an army attacks another (health, damage, block) and changing the colours of the tiles grid
6. Logs: This takes in inputs from the other partial classes in the form of events when a certain milestone is reached, so a story or tutorial event begins. It also takes damage done from each attack & which side won each battle/war from the combat partial class, and when an autosave is completed from the fileHandling partial class. When a war is won or lost, the combat partial class will also send the loot values won or lost from the wars which displays the numbers in the logs. It also deals with the user pressing the clear button to clear the current logs displayed, although it mostly runs passively in the background where the user does not have to interact with it much
7. FileHandling: This handles all of the saving, autosaving, loading and deleting of saves. When loading, it goes through each line of the file, copying the values directly into the global variables stored in main. When saving, the data is copied directly from the global variables and into the file. Of course, when an operation like this occurs, the main timer that the

whole game runs off is turned off because otherwise it could mean that inconsistent things like year number and war might be different, thus breaking the game

Class Links



GlobalData

The structure of the global variables will be key for the readability and function of the code. I will need to have them using as few variables as possible, whilst also keeping them relevant, tidy, and accessible. The idea I have with them is each part will have its own global variable, which will be a list of arrays, holding multiple values. This would then allow for e.g. the resource collection timers to loop through the same various but of different elements, each holding values for the different resources. It would look something like this:

```

LIST<INT[]> resourcesData = NEW LIST<INT[]> //List of arrays
  NEW INT[4] {1000, 1000, 1000, 1000}, //Amount
  NEW INT[4] {1, 1, 1, 1}, //Rate
  NEW INT[4] {1000, 1000, 1000, 1000}, //Capacity
  NEW INT[4] {1, 1, 1, 1} //Gather multiplier
  
```

In this example, [][]0 would be wood, [][]1 would be stone, [][]2 would be food, [][]3 would be metal. [0][] would be amount, [1][] would be rate and so on. Because of this variable design, many different resources can be accessed using one variable, thus one loop when adding e.g. 1000 of each resource as a consequence for winning a war. The reason the variables will be structured as list of arrays is because then if the game gets past prototype stage, more values and complicated mechanics could be added into each variable without having to change how any of the existing parts work, and saves me from having to add multiple variables for the same general things later on.

Talking about future proofing, I will also have variables that store the various names of each resource, in an array, sorted by each type. E.g.

```
STRING[] resourceName = NEW STRING[4] {"Wood", "Stone", "Food", "Metal"}; //Names
```

This will mean that if the game goes far enough, just changing these resource names would automatically change the types for all values, and thus would allow the progression system in game to not take up a stupid number of different variables and parts for each different resource. However currently I am only developing a prototype so I am not required to go any further.

Now I will show the layouts I have come up with for each needed global variables. (Note: Arrays in lists of arrays have a starting index of 1, not 0).

Resources

```
STRING[] resourceName = NEW STRING[4] {"Wood", "Stone", "Food", "Metal"}; //Names  
INT scienceData = 0; //Amount of science there is, used to buy upgrades
```

Housing

```
STRING[] housingNames = NEW INT[3] {"Shack", "Bootcamp", "Barracks"};  
INT[] housingData = NEW INT[3] {5, 10, 15}; //How much space each housing type gives  
INT totalHousing = 0; //Total housing space  
INT housingRemaining = 0; //How many workers can fit in current housing
```

Upgrades

```
LIST<INT[]> upgradesCosts = NEW LIST<INT[]>  
    NEW INT[4] {50, 50, 50, 100}, //Storage Costs  
    NEW INT[4] {10, 10, 10, 20}, //Workers Costs  
    NEW INT[1] {100}, //Science Costs  
    NEW INT[3] {50, 50, 100} //Housing Costs
```

Combat

```
//[[0]Player combat data, [][1]Enemy combat data  
LIST<INT[]> combatData = NEW LIST<INT[]>  
    NEW INT[2] {0, 0}, //Health  
    NEW INT[2] {0, 0}, //Attack  
    NEW INT[2] {0, 0}, //Block
```

I also have my constants class, which, as the name describes, holds all the constants. All constants variables will be capitalised so that I know that they are a constant when programming.

Technically, globalData and constants are both their own classes, but the main class will parent them, as the main class is where everything will branch off. When I want to access or change a global variable, I will have to do globalData.combatData for example, or constants.INTERVAL_TIME.

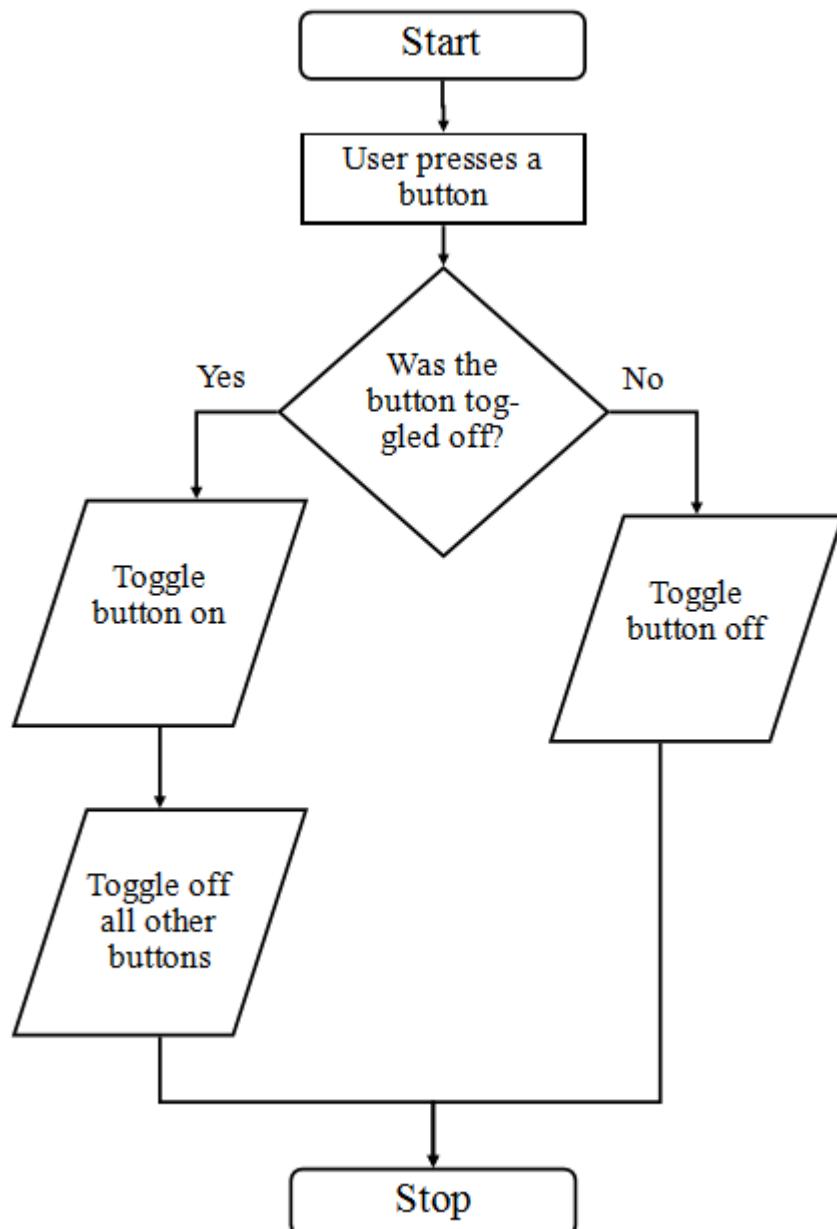
ResourceCollection [A]

The resource collection class will be split up into 3 regions;

1. Buttons: these will call functions that check if other buttons are toggled on and need to be toggled off because only one button can be toggled on at once
2. Timers: there will be a separate timer for each resource type that runs the resource gathering function and the year counting etc, as they require different intervals
3. Functions: these functions include the main resource gathering function that runs every tick (by each resource timer) which calculates how much each resource grows by each second etc. Most of the other functions will be run by this main function as simple error checking (like stopping dividing by 0), and milestone checking (if a resource reaches 1,000, turn the displayed number into 1k). The remaining functions will just be the button toggles checking run by clicking each of the resource buttons, and handling the housing bar.

Buttons

The buttons' code should be very simple – when a button is clicked, run an event off it and run a couple of button toggles checking functions inside them.



```
FUNCTION btnWood()
    toggleButton(0);
    checkOtherButtons(0);
END FUNCTION
...
```

The reason there are 2 functions run here is because the first checks if the button is already toggled on, and if so, toggles it off, and if not, toggles it on. The second then toggles off all other buttons so that only one says on. The following functions should look something like:

```
BOOLEAN[] btnToggled = BOOLEAN[4] {TRUE, TRUE, TRUE, TRUE};

FUNCTION toggleButton(INT arraySlot)

    IF btnToggled[arraySlot] = TRUE THEN

        btnToggled[arraySlot] = FALSE;

    ELSE

        btnToggled[arraySlot] = TRUE;

        checkOtherbuttons(arraySlot);

    END IF

END FUNCTION

FUNCTION checkOtherbuttons(INT arraySlot)

    IF btnToggled[arraySlot] = TRUE THEN

        FOR INT i = 0, i < btnToggled.LENGTH; i++

            IF !btnToggled[i] = (btntoggled[arraySlot]) THEN

                btnToggled[i] = TRUE;

            END IF

        NEXT

    END IF

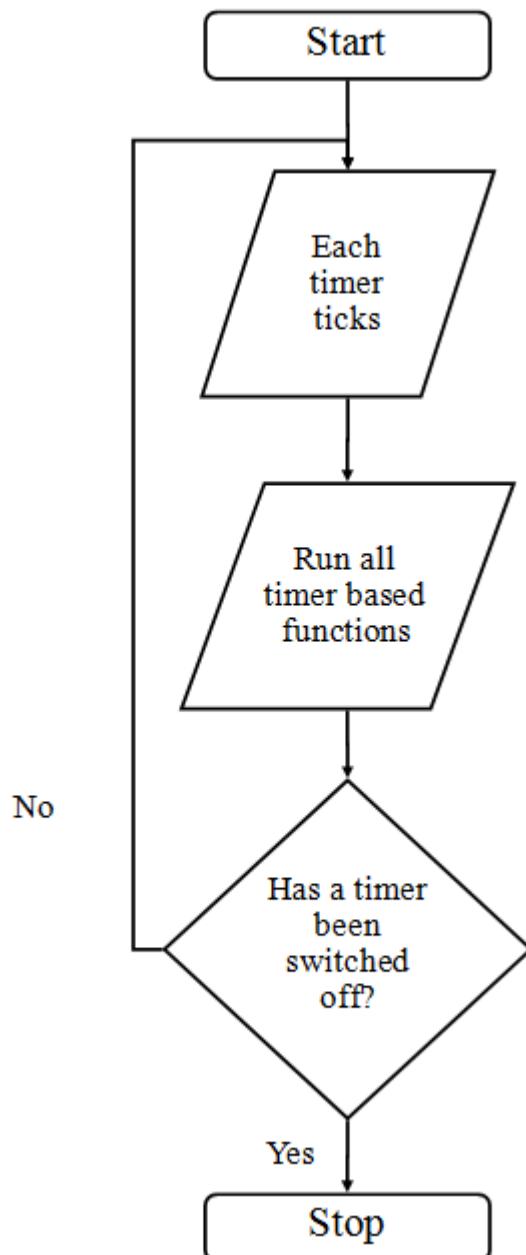
END FUNCTION
```

The parameters always point to the array slots from btnToggled, as that variable stores which button is toggled or not (true being toggled on, false being toggled off). As seen from the buttons code, btnWood has index 0, btnStone has index 1 and so on.

Finally, I will have a small function that runs in my resourceCollection function that makes sure the currently toggled button is brown in colour, to indicate that the user knows the button has been toggled on (and all other buttons are toggled off, as they are not uniquely coloured).

Timers

There will be a separate timer for each resource type + housing in order to run the resourceCollection functions every tick. That way, each resource progress bar will tick up at the same time and all parts will stay uniform.

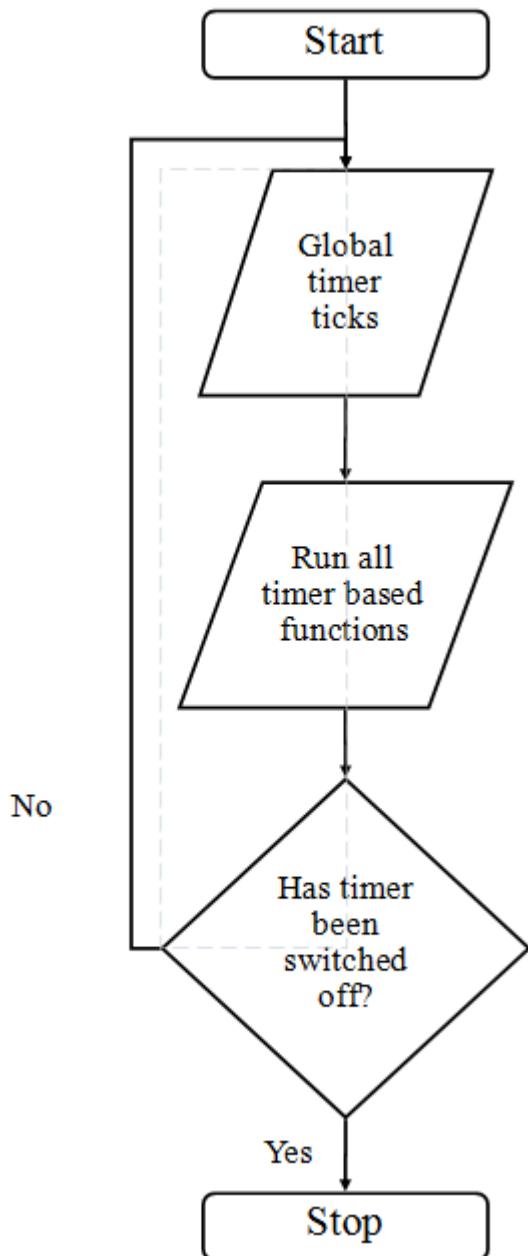


```
FUNCTION woodTimer()  
    resourceCollection(0, ref woodTimer, ref btnWood, lblWoodAmount, lblWoodRate, ref  
    pBarWood);  
END FUNCTION
```

Then repeat for each resource timer, so stoneTimer would reference all the forms components to do with the stone part. The first number references the element of the resources global variable.

After some thinking about this, I have realised that instead of many different timers, I could just use one, global timer that calls that function referencing each different part each tick. This would then save on performance because there would be only one timer instead of multiple all ticking at the same time.

The global timer will tick every second (1000ms) apart. There are 900 years of wars that the game will cover so far. If there is 1 year per minute, and there are 60 ticks per minute, then 900 years will be over in 900 minutes or 15 hours. This would mean that a war of 5 years would last 5 minutes in real life. This is a good amount of time because wars do not drag on for ages and the player does not get bored.



```
FUNCTION globalTimer()  
    resourceCollection(0, ref woodTimer, ref btnWood, IblWoodAmount, IblWoodRate, ref  
    pBarWood);  
    ...  
    housing();  
END FUNCTION
```

Functions

The main function is resourceCollection, which as seen above is called by the globalTimer for each resource type, referencing their control forms. This function is responsible for:

1. Fetching the rate and amount of each resource from the globalData variables
2. Sets the output for the resource's rate label
3. Calculates how much of each resource needs to be added to the amount each tick, and then update the progress bar based on how much of each resource there is

```
FUNCTION resourceCollection(INT arraySlot, ref Button buttonType, ref Label labelAmounts, ref
Label labelRates, ref ProgressBar pBarType)

    INT currentRate = globalData.resourcesData[1][arraySlot];
    INT currentAmount = globalData.resourcesData[0][arraySlot];

    IF btnToggled[arraySlot] = TRUE THEN
        buttonType.BackColour = Colour.Gray
    ELSE
        currentRate += globalData.resourcesData[3][arraySlot];
        buttonType.BackColour = Colour.Brown
    END IF

    amountCheck();
    labelRates.TEXT = ("+" + currentRate + "/sec");

    IF pBarType.VALUE <= (pBarType.MAXIMUM - currentRate) THEN
        globalData.resourcesData[0][arraySlot] += currentRate;
        labelAmounts.TEXT = currentAmount.TOSTRING();
```

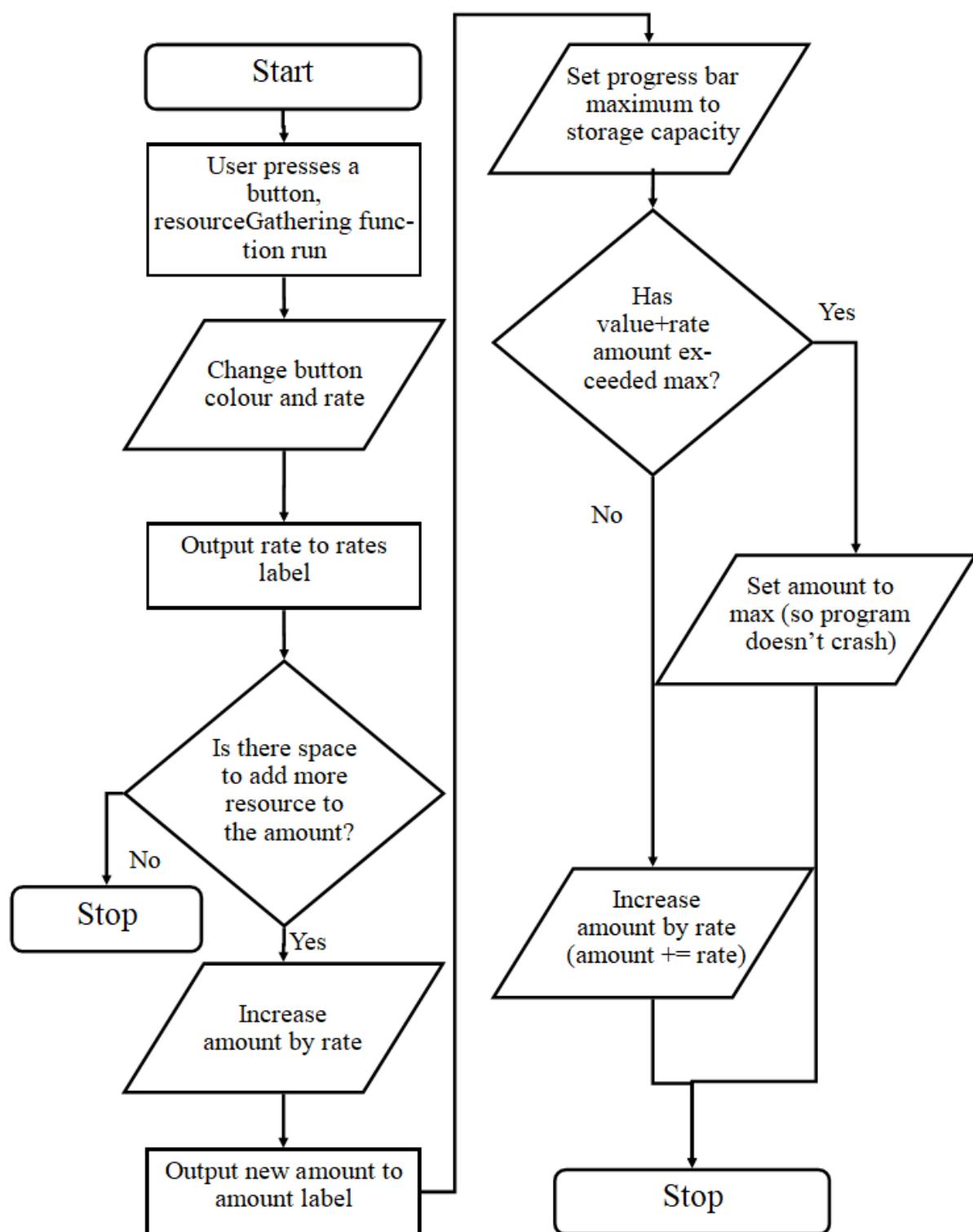
```
pBarType.MAXIMUM = globalData.resourcesData[2][arraySlot];
```

```
IF (pBarType.VALUE + currentRate) > pBarType.MAXIMUM THEN
    pBarType.VALUE = pBarType.MAXIMUM;
ELSE
    pBarType.VALUE += currentRate;
END IF
END IF
END FUNCTION
```

The first two lines get the current rate and amount from the resource's resouceData. Then, the changeBtnColour function is called, followed by amountCheck(). amountCheck() makes sure that if a resource amount goes below 1, set it to equal 1 because otherwise further down the line the resource may be divided by 0 thus crashing the game.

```
FUNCTION amountCheck()
FOR INT i = 0; i < 4; i++
    IF globalData.resourcesData[i][0] < 1 THEN
        globalData.resourcesData[i][0] = 1;
    END IF
NEXT
END FUNCTION
```

Next, the current rate is applied to the label before all the calculations start. Next, the if statement checks if there is space to add more resources to add, otherwise it will go over the maximum amount and cause issues, perhaps crashing the program as well because the progress bar goes over its maximum threshold.



Last but not least is the housing() function seen in the button event. This copies over data from the totalHousing and housingRemaining variables into locals, outputs the currentHousing amount to a label, sets the progress bar value to the currentHousing amount and also works out the maximum progress bar value.

```

FUNCTION housing()
    INT currentHousing = globalData.totalHousing;
    INT housingRemaining = globalData.housingRemaining;

    lblHousingAmount.TEXT = currentHousing.TOSTRING();
    pBarHousing.VALUE = currentHousing;
    pBarHousing.MAXIMUM = housingRemaining - currentHousing;
END FUNCTION

```

Upgrades [A]

In this class, I need to:

- Process when an upgrade is bought
- Increase the costs of each upgrade when bought to make the user fight rather than endlessly buy upgrades faster and faster

There will be two main parts of the upgrades class – buttons which will call the functions, and the functions which will process the upgrades. There will be 4 sets of functions – storage, workers, research & housing. These will calculate if the user has enough of the right resource to buy an upgrade, and if they do, minus the cost from the resource amount, increase the cost of the upgrade, and do whatever special process for each function. E.g. housing will need to increase the housing size, and in the workers function it needs to make sure there is enough housing space available.

Upgrade name	What it costs (start)	What it does	Cost multiplier	Year of appearance (if applicable)
Wood storage	50 wood	Increases storage by 1000	Current cost*3	
Stone storage	50 wood	Increases storage by 1000	Current cost*3	
Food storage	50 wood	Increases storage by 1000	Current cost*3	
Metal storage	100 wood	Increases storage by 500	Current cost*3	
Wood gatherer	100 food, 1 housing space	Increases wood rate by 1	Current cost*4	

Stone quarry-er	100 food, 1 housing space	Increases stone rate by 1	Current cost*4	
Food butcher	100 food, 1 housing space	Increases food rate by 1	Current cost*4	
Metal miner	200 food, 1 housing space	Increases metal rate by 1	Current cost*4	
Troop	400 food, 1 housing space	Increases troop amount by 1	Current cost*2	
Aqueducts research	100 science	Increases food and wood rate by 5	Current cost*5	400BC
Stamp-mill research	500 science	Increases stone rate by 2	Current cost*5	200BC
Trip-hammer research	1000 science	Increases metal rate by 2	Current cost*5	0AD
Hushing research	3000 science	Increases metal rate by 5	Current cost*5	200AD
Villa research	5000 science	Increases every housing space by *3	Current cost*5	The First Persian Wars
Troop health	200 metal	Increases troop health by 100	Current cost*2	
Troop block	500 metal	Increases troop block by 250	Current cost*2	
Troop damage	300 metal	Increases troop damage by 150	Current cost*2	
Shacks	50 stone	Increases housing space by 5	Current cost*2	
Boot camp	100 stone	Increases housing space by 10	Current cost*2	
Barracks	500 stone	Increases housing space by 20	Current cost*2	

Science will be gained through winning battles and wars, which I will get to when designing the combat system. But essentially the player will be forced to engage in combat because of the science incentives to increase their resource gains and move through the game.

Many of these costs are estimates, and once I get to a stable game stage, I can tweak and balance the upgrades and their costs to a more playable state, as the gain may progress incredibly slowly or incredibly quickly with these current values.

New upgrades will appear if the player:

- Wins a war
- Gets to a certain resource milestone e.g. 10,000 metal
- Gets to a certain year

```
FUNCTION btnWoodStorage()
```

```
    buyingStorage(globalData.upgradesCosts[0][0], 0, 0, 1000);
```

```
END FUNCTION
```

Resource type to
buy this

Resource type this
is being bought for

How much storage
is increased by

...

```
FUNCTION btnGatherer()
```

```
    buyingWorkers(globalData.upgradesCosts[1][0], 2, 0);
```

```
END FUNCTION
```

Resource type to
buy this

Resource type this
is being bought for

...

```
FUNCTION btnAquaducts()
```

```
    buyingResearch(globalData.upgradesCosts[2][0]);
```

```
END FUNCTION
```

...

```
FUNCTION btnShack()
```

```
    buyingHousing(globalData.upgradesCosts[3][0], 1, 0);
```

```
END FUNCTION
```

Resource type to
buy this

Housing type

...

```
FUNCTION btnHealth()
```

```
    buyingCombat(globalData.upgradesCosts[4][0], 3, 0, 100);
```

```
END FUNCTION
```

...

Resource type to
buy this

Combat type

How much it is
being increased by

```
FUNCTION buyingStorage(INT cost, INT typeToBuy, INT typeToBuyFor, INT storageIncrease)

    IF globalData.resourcesData[0][typeToBuy] < cost THEN

        PRINT "You do not have enough resources for this"

    ELSE

        globalData.resourcesData[0][typeToBuy] -= cost

        globalData.upgradesCosts[0][typeToBuy] *= globalData.costMultipliers[0]

        globalData.resourcesData[2][typeToBuyFor] += storageIncrease

        PRINT cost

    END IF

END FUNCTION

FUNCTION buyingWorkers(INT cost, INT typeToBuy, INT typeToBuyFor)

    IF globalData.resourcesData[0][typeToBuy] < cost OR globalData.housingRemaining == 0
THEN

        PRINT "You do not have enough food or housing for this"

    ELSE

        globalData.resourcesData[0][typeToBuy] -= cost

        globalData.upgradesCosts[1][typeToBuy] *= globalData.costMultipliers[1]

        globalData.resourcesData[1][typeToBuyFor] += 1

        globalData.housingRemaining -= 1

        PRINT cost

    END IF

END FUNCTION

FUNCTION buyingResearch(INT cost)

    IF globalData.scienceData < cost THEN

        PRINT "You do not have enough science for this"

    ELSE

        globalData.scienceData -= cost

        globalData.upgradesCosts[2][1] *= globalData.costMultipliers[2]

        PRINT cost

    END IF

END FUNCTION
```

```
FUNCTION buyingHousing(INT cost, INT typeToBuy, INT housingType)

    IF globalData.resourcesData[0][typeToBuy] < cost THEN

        PRINT "You do not have enough resources for this"

    ELSE

        globalData.resourcesData[0][typeToBuy] -= cost

        globalData.upgradesCosts[3][typeToBuy] *= globalData.costMultipliers[3]

        globalData.totalHousing += globalData.housingData[housingType]

        globalData.main.housing()

        PRINT "You have increased your housing space!"

        PRINT cost

    END IF

END FUNCTION
```

```
FUNCTION buyingCombat(INT cost, INT typeToBuy, INT combatType, INT buff)

    IF globalData.resourcesData[0][typeToBuy] < cost THEN

        PRINT "You do not have enough metal for this"

    ELSE

        globalData.resourcesData[0][typeToBuy] -= cost

        globalData.upgradesCosts[4][typeToBuy] *= globalData.costMultipliers[4]

        globalData.combatData[combatType][0] += buff

        PRINT cost

    END IF

END FUNCTION
```

```
FUNCTION buyingTroops(INT cost, INT typeToBuy)

    IF globalData.resourcesData[0][typeToBuy] < cost OR globalData.housingRemaining == 0
THEN

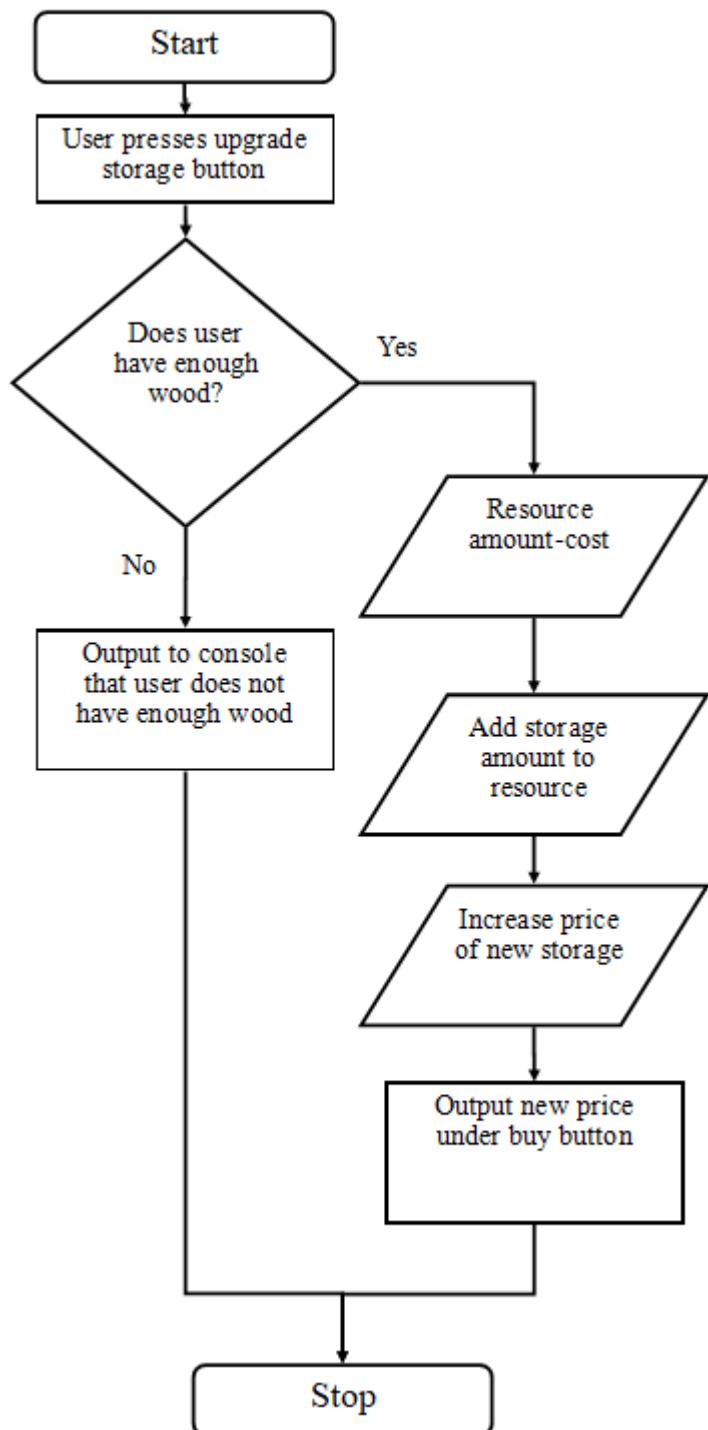
    PRINT "You do not have enough metal or housing for this"

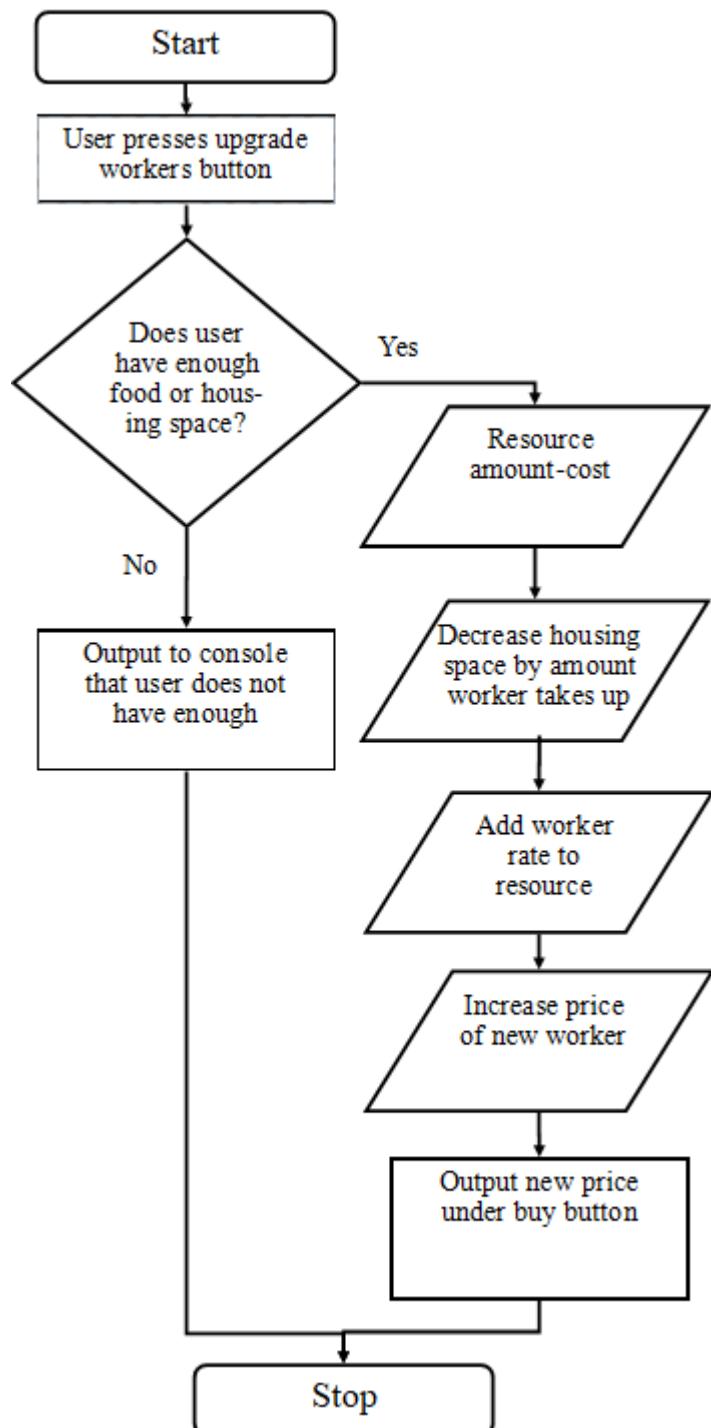
    ELSE

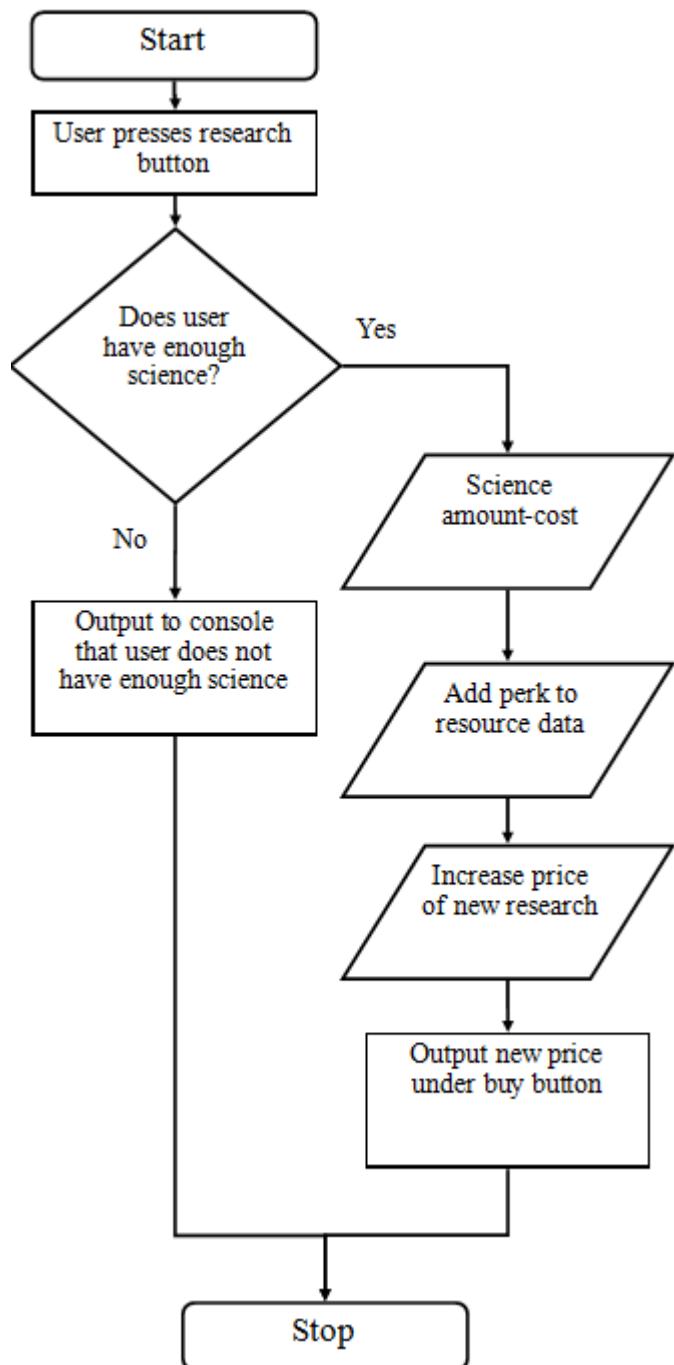
        globalData.resourcesData[0][typeToBuy] -= cost

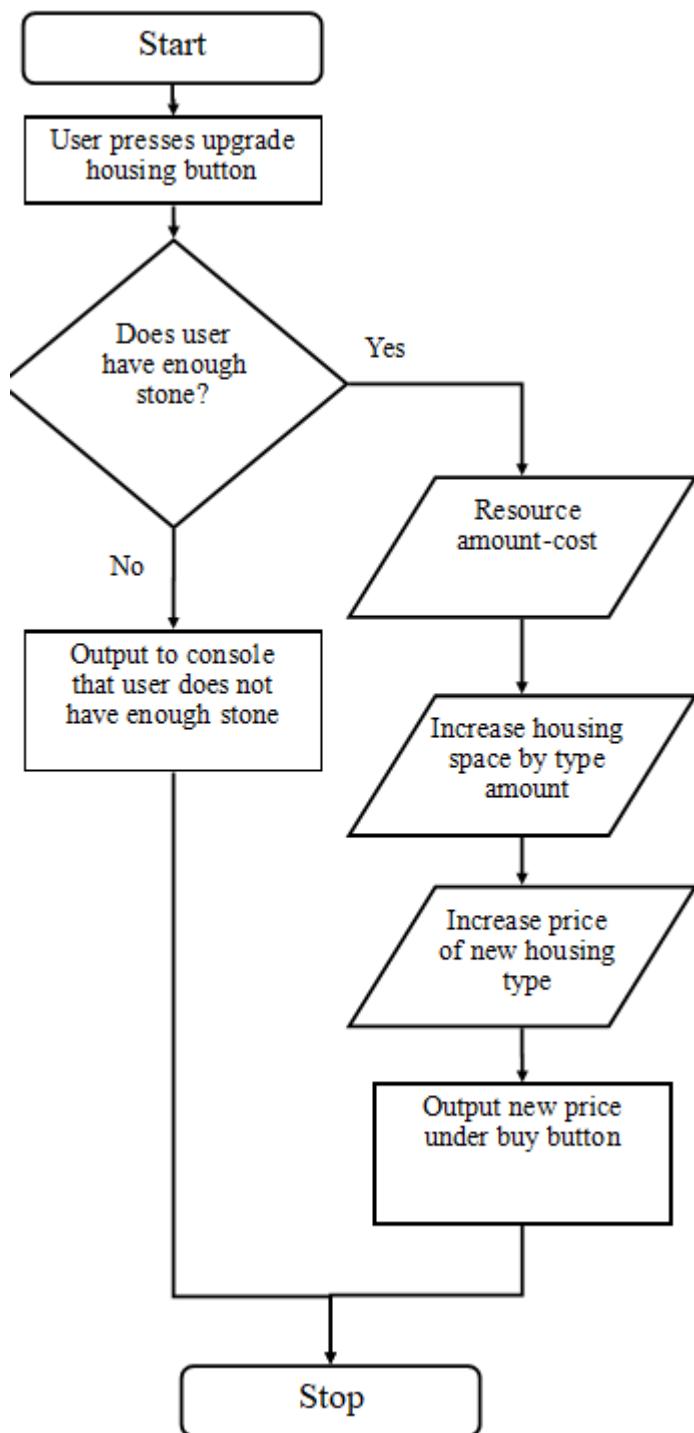
        globalData.upgradesCosts[4][typeToBuy] *= globalData.costMultipliers[4]
```

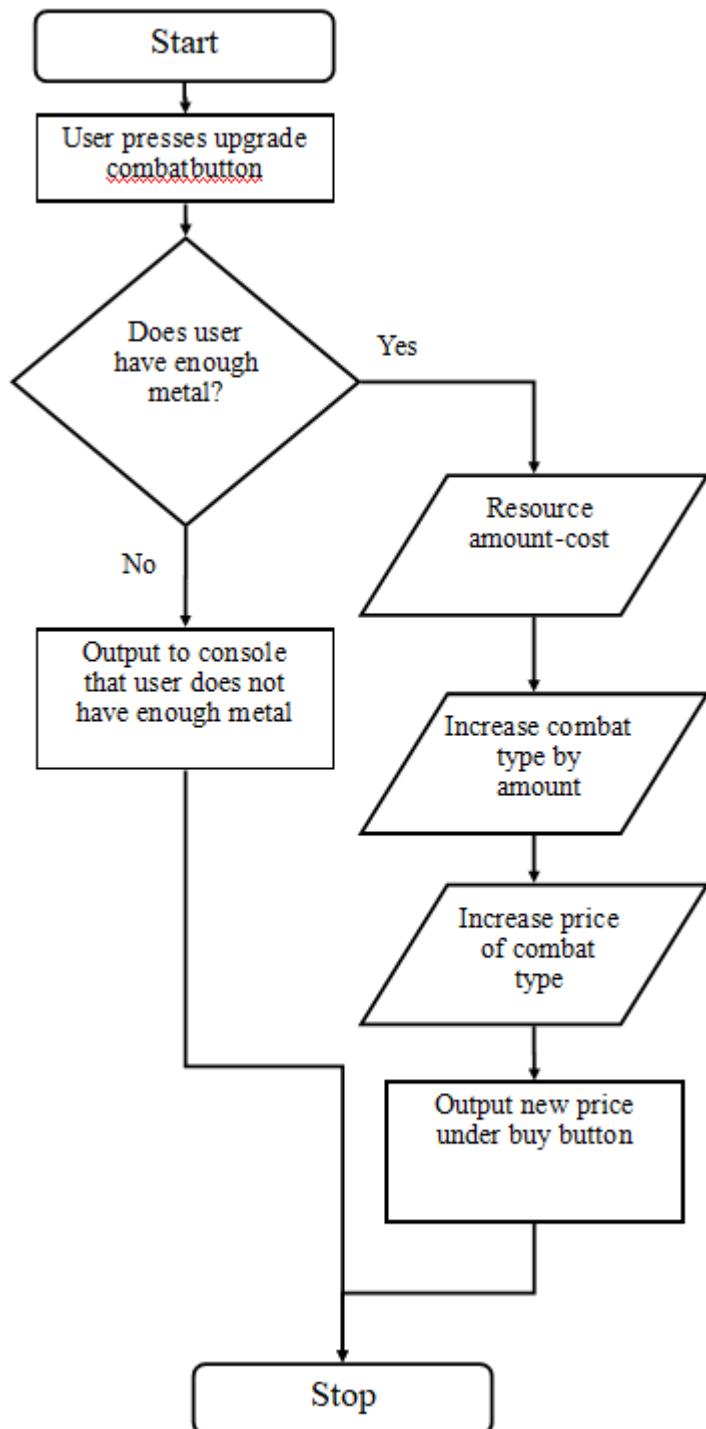
```
globalData.combatData[3][typeToBuyFor] += 1  
globalData.housingRemaining -= 1  
PRINT cost  
END IF  
END FUNCTION
```

Storage

Workers

Research

Housing

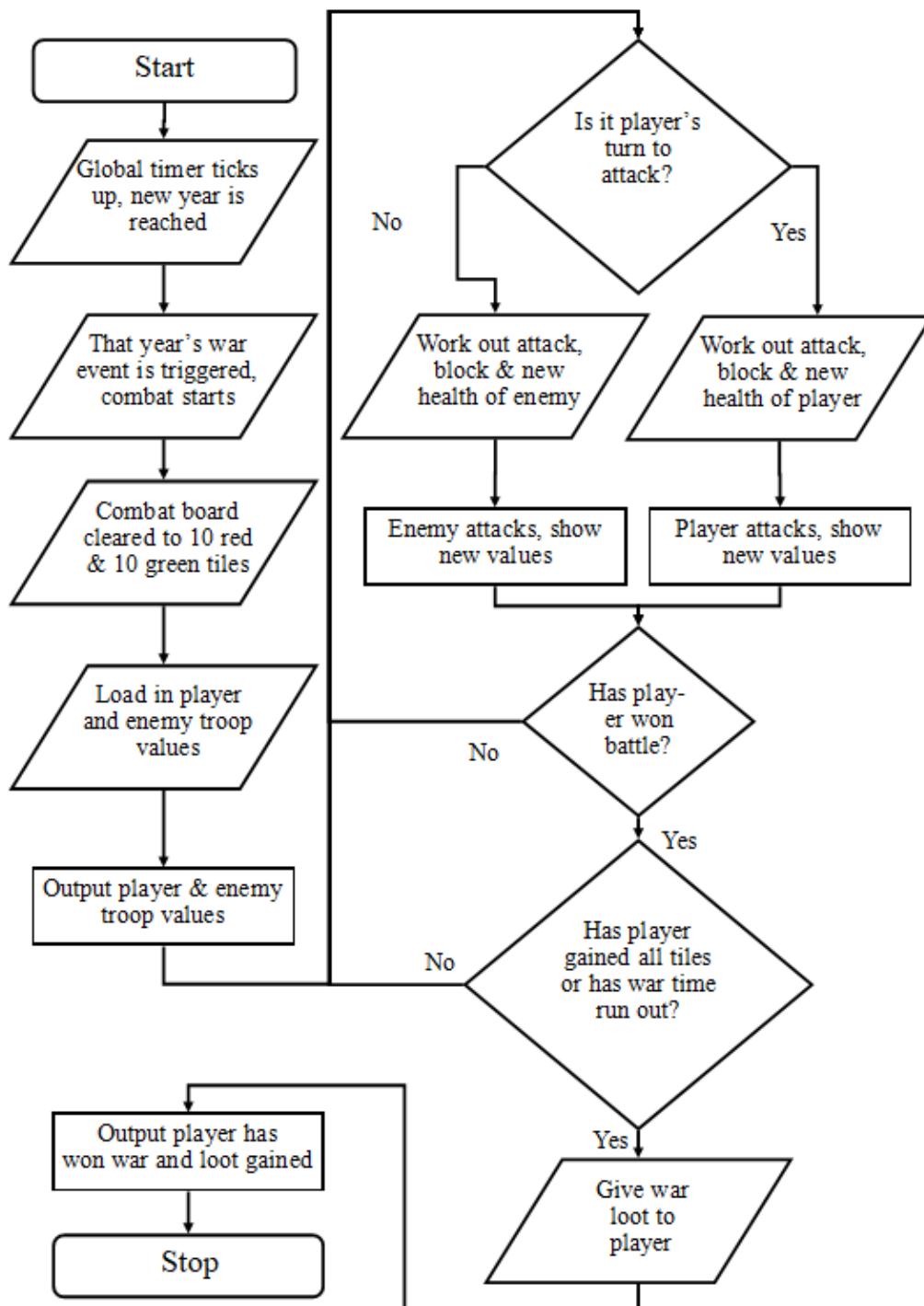
Combat

Combat [C]

V1 16/09/19

The combat system will consist of 6 main parts:

1. Calculating when a war will begin
2. Calculating attacks, block and health
3. Calculating board state
4. Calculating new enemy health, block and attack
5. Calculating when a side wins a battle or war
6. Calculating how much loot + science is made from each war



When wars begin

As I said before in the *ResourceCollection* section, each in-game year will be equivalent to 1 real life minute. Referring back to the list of wars decomposition diagram, one can find the event of every single war. In my program, these wars are listed as a number between 0 and 54,000, 1 being tick (second) 0 and 54,000 being tick (second) 54,000. For the sake of simplicity, each war will go on for 10 years (minutes) so that the program doesn't need to also work out how long each war goes on for and other overcomplicated calculations.

War year	Minute equivalent	Tick equivalent
495 BC	5	300
446 BC	54	3240
400 BC	100	6000
380 BC	120	7200
340 BC	160	9600
295 BC	205	12300
264 BC	236	14160
218 BC	282	16920
150 BC	350	21000
130 BC	370	22200
41 BC	459	27540
30 BC	470	28200
18 BC	482	28920
43 AD	543	32580
61 AD	561	33660
161 AD	661	39660
296 AD	796	47760
344 AD	844	50640

Calculating attacks

The attack system will work by switching between the enemy and player, so a war starts, it is the player's turn, then it is the enemy's turn etc. If the player or the enemy wins a battle this turn based system still continues over a new battle. The only time it resets is when a new war is started. The enemy starts with 2 rows of 5 red tiles at the top, and the player starts with 2 rows of 5 green tiles at the bottom. In order, the attacks would go something like this:

1. Battle for first tile commences. Player attacks first. Player does 50% of enemy's health.
2. Enemy attacks, and does 25% of player's health.
3. Player attacks, does another 50% of enemy's health, thus winning 1 of the enemy's red tiles.
Now the enemy has 9 tiles and the player has 11.
4. New battle. Enemy attacks, does 25% of player's health. Player now down 75% health.
5. Player attacks, does 45% enemy's health, enemy now has 55% health.
6. Enemy attacks, does 75% player's health. Player dies, the enemy gets back that tile.
7. This process continues until either:
 - a. War time runs out (5 minutes) and the number of enemy & player tiles are counted
 - b. One side gains all of the others' tiles (so has 20 total tiles)

8. If the war time runs out and both sides still have the same number of tiles (10), the war will be counted as a draw and appropriate loot will be given to the player.

Each attack will happen every 5 ticks (seconds), so a war could last up to 120 turns total, as a war can last up to 10 minutes. Along with all of this, the progress bars and values around the tile grid are updated to easily show the user what is happening during combat (see GUI design for combat). The actual damage is calculated by:

Blocked player damage = player damage – enemy block

New enemy health = enemy health – blocked player damage

E.g.

Player health = 100. Player attack = 50. Player block = 20

Enemy health = 100. Enemy attack = 30. Enemy block = 40

Player does 50 damage, however because the enemy has a block of 40, only 10 damage gets through, therefore the enemy's new health is 90.

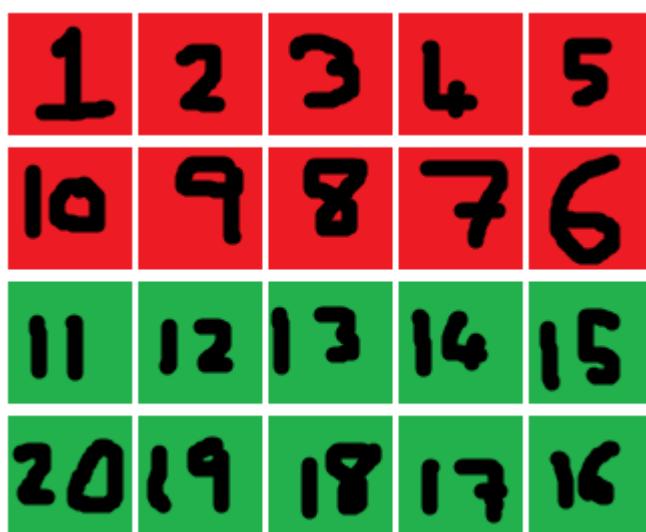
Enemy does 30 damage, but the player has a block of 20, so again only 10 damage is done, so the player's new health is 90.

The player total block, health and attack will be calculated by: 1. Troop amount * block 2. Troop amount * health 3. Troop amount * attack.

The enemy stats will be calculated before a war starts and will get generally more difficult to fight each war, as the player will also become stronger through upgrades. The values for each stat will be randomly chosen in a decided group of numbers – x% below the player's current stat and x% above. X is a variable yet to be decided because more thinking needs to be done on how much this needs to be, but it will probably be somewhere around 10-30%.

When a tile is lost or won by the player, the health of both sides will reset back to maximum.

The sequence of the attacking will go by:



Where the first tile each war being attacked is tile 10.

The function below shows the global timer, which ticks every second, and sets the pace for most parts of the program. The “...” parts just replace the other parts of the function which are not necessary to show here. Tick counter counts the number of ticks so far. This is necessary to know when a war will begin, as they are delayed by tick counts. The global timer also contains startWar() because every tick the program needs to check if a new war needs to be started. This function will be shown later. The combat timer runs calculateAttack() and winWarCheck() because the attacks are every 5 ticks, so this timer runs with an interval of 5000 milliseconds or 5 seconds.

```
FUNCTION globalTimer()  
    ...  
    tickCounter += 1  
    startWar()  
END FUNCTION
```

```
FUNCTION combatTimer()  
    calculateAttack()  
    winWarCheck()  
END FUNCTION
```

```
FUNCTION startWar()  
    FOR INT i = 0; i < warTimes.LENGTH; i++  
        IF tickCounter == warTimes[i] THEN  
            currentWar = warTimes[i]  
            resetBoard()  
        END IF  
    NEXT  
END FUNCTION
```

This start war function checks if a war has started, by comparing every value in the warTimes array with the current value of tickCounter. This then calls the resetBoard() function.

```
FUNCTION resetBoard()

    CONST INT rows = 4

    CONST INT columns = 5

    FOR INT x = 0, x < columns; x++

        FOR INT y = 0, y < rows; y++

            FillRectangle(green, x * 65, y * 65, 60, 60)

        NEXT

        FOR INT y = 2, y < rows; y++

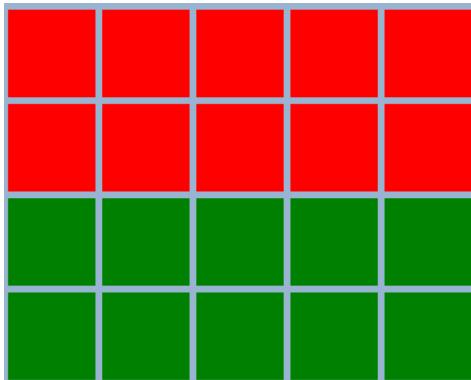
            FillRectangle(red, x * 65, y * 65, 60, 60)

        NEXT

    NEXT

END FUNCTION
```

The resetBoard() function sets the combat grid to be the normal 2 rows of 5 green tiles at the bottom and 2 rows of 5 red tiles at the top, as seen in the GUI design of the combat menu.



```
FUNCTION calculateAttack()

    IF playerTurn = TRUE THEN

        blockedPlayerDamage = playerDamage - enemyBlock

        enemyHealth -= blockedPlayerDamage

        IF enemyHealth <= 0 THEN

            playerTiles += 1

            FOR INT x = 0, x < columns; x++

                FOR INT y = 0, y < rows; y++

                    IF grid[y, x] != checkSquare THEN

                        grid[y + 1, x] = 1;

                    ELSE

                        grid[y + 1, x] = 0;

                    END IF

                NEXT

            NEXT

            updateBoard()

            RESET enemyHealth

        END IF

    ELSE

        blockedEnemyDamage = enemyDamage - playerBlock

        playerHealth -= blockedEnemyDamage

        IF playerHealth <= 0 THEN

            playerTiles -= 1

            FOR INT x = 0, x < columns; x++

                FOR INT y = 0, y < rows; y++

                    IF grid[y, x] != checkSquare THEN

                        grid[y + 1, x] = 1;

                    ELSE

                        grid[y + 1, x] = 0;

                    END IF

                NEXT

            NEXT

        END IF

    NEXT
```

```

        NEXT
        updateBoard()
        RESET playerHealth
    END IF
END IF
END FUNCTION

```

This calculateAttack() function does all the calculations on switching turns using the boolean playerTurn, on how much damage either side does to the other, and if a side loses all health, thus losing a tile. The nested loops inside the check of no health is the part that gives the right side the next tile in the grid. The grid variable is a 2d array that stores each tile as a 1 or 0. If it is a 1, then it is owned by the player. If it is a 0, it is owned by the enemy.

Because the grid variable been updated (if the enemy or player wins a battle), the actual board the player sees needs to be updated.

```

FUNCTION updateBoard()

    CONST INT gridSize = 60
    CONST INT rows = 4
    CONST INT columns = 5
    CONST INT checkSquare = grid[0, 0]
    FOR INT x = 0, x < columns; x++
        FOR INT y = 0, y < rows; y++
            FillRectangle(brushes[grid[y, x]], Rectangle.FROMLTRB(x * (gridSize), y *
                (gridSize), x * (gridSize) + 65, y * (gridSize) + 65))
            IF grid[y, x] != checkSquare THEN
                allEqual = TRUE
            END IF
        NEXT
    NEXT
END FUNCTION

```

This function actually works slightly differently to the resetBoard function, as it uses an array of brushes (green and red brush) to fill the new squares. Therefore, instead of having to make multiple loops to fill each square manually, the program just has to draw every square and then work out if it needs to be filled green or red (player or enemy respectively). There is also a check on here that

checks if the top left corner of the grid (checkSquare) is not equal to any other of the grid squares, then all the other squares must have been taken by the player or enemy, thus a side has won.

```

FUNCTION winWarCheck()

    IF allEqual = TRUE || (tickCounter + 600) >= (currentWar + 600) THEN

        INT offset = 0;

        FOR INT x = 0, x < columns; x++

            FOR INT y = 0, y < rows; y++

                offset += 1 - (2 * grid[y, x])

                SWITCH offset

                    CASE offset = 0

                        PRINT "Draw"

                    CASE offset > 0

                        PRINT "Enemy wins"

                    CASE offset < 0

                        PRINT "Player wins"

                END SWITCH

            NEXT

        NEXT

        calculateLoot()

    END IF

END FUNCTION

```

This function then takes each and every tile, and adds them up to get offset. 1 is added if the tile is owned by the enemy, and -1 is added if the tile is owned by the player. If the offset is 0, the enemy and player own the same number of tiles, if offset is more than 0 then the enemy wins, and if the offset is less than 0 the player wins.

There are two conditions that may need to be met for this to happen though – if the updateBoard function runs it with allEqual on true, or if the current war has been ongoing for 10 minutes. Then all the above said happens and the win condition is found. Finally, the calculateLoot() function is run which calculates how much science and other resources the player wins based on their performance during war.

The next issue is to work out how to increase the enemy's stats each war. This is necessary otherwise the player will thrash the enemy every single time and there would be no challenge in the game. Currently, based off the cost, cost increase and buff of the player combat upgrades, I will increase the enemy stats each war by a random number between *3-5 after every war. If a player loses however, the enemy will “capture” some of the player’s tech, thus increasing the range to *3-7. These calculations can be done during the winWar() conditions. Whilst I develop the program, I

may feel that I need to change some of these values in order to get more balance but that won't be an issue.

```

IF draw OR playerWins
    PICK randomNum(3, 5)
        globalData.combatData[0][1] *= randomNum
        globalData.combatData[1][1] *= randomNum
        globalData.combatData[2][1] *= randomNum
        globalData.combatData[3][1] += randomNum

ELSE IF enemyWins
    PICK randomNum(3, 7)
        globalData.combatData[0][1] *= randomNum
        globalData.combatData[1][1] *= randomNum
        globalData.combatData[2][1] *= randomNum
        globalData.combatData[3][1] += randomNum

```

The final issue is working out how much loot and science the player will gain.

1. Loot (if they win)
 - a. Gets half of the number of troops as the enemy had along with more housing space to accommodate them
 - b. Gets $500 * \text{war number}$ of each wood, stone, metal & food.
2. Loot (if they lose)
 - a. Loses between 5 and 10 troops to the enemy, but keeps the housing space
 - b. Gets $100 * \text{war number}$ of each wood, stone, metal & food.
3. Loot (if they draw)
 - a. Keeps all troops
 - b. Gains $200 * \text{war number}$ of each wood, stone, metal & food
4. Science
 - a. There are only a limited number of wars, so therefore enough science should be awarded every war to allow the upgrades to be bought multiple times
 - b. Therefore the player gains $5000 \text{ science} * \text{war number}$ per win, but only $1000 \text{ science} * \text{war number}$ on loss or draw. This is enough to buy the villa upgrade for the first war, but the player may decide to upgrade the first couple of upgrades a few times in order to increase their resource rates.

Again, this is all subject to change through the development of the program if I feel that the values should be changed.

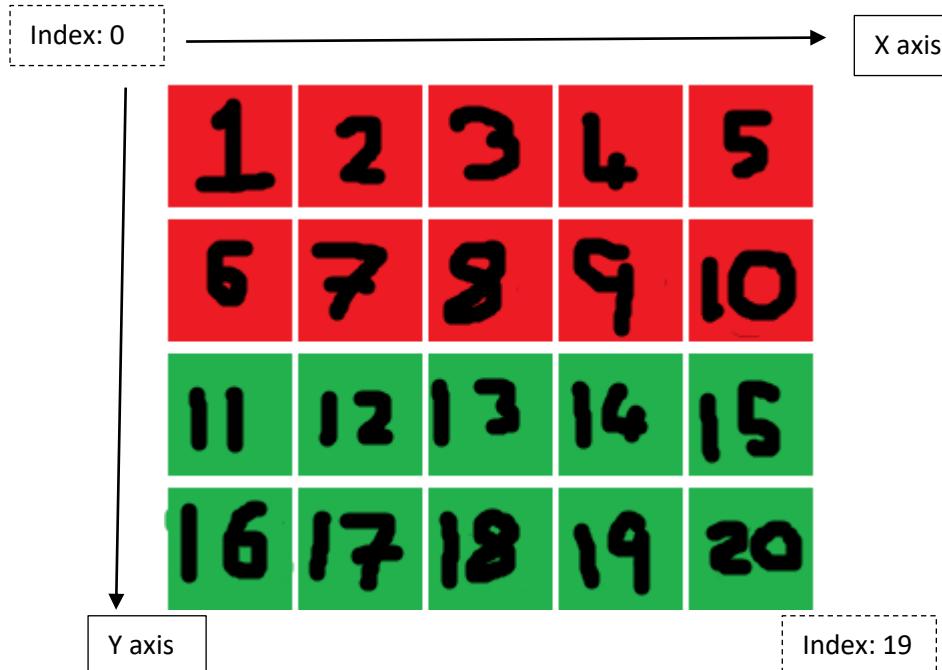
V2 11/11/19

Most of the combat system that I have design is fine. However, the actual section calculating which tiles the enemy and player owns and how to draw them is not, as I have come to realise, something that will work. The core problem lies in how I wanted the tiles order to work. I wanted it too snake around like this:

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16

This is a very odd way of doing things and requires a far more complex system to find which tile belongs to which side – the system being the offset system. This worked out how many tiles on either side of a certain tile (number 10 which was actually number 6 in my case) thus working out how many tiles each side had, and drawing the grid accordingly. I evidently did not work out how to code this correctly thus I have switched to a far nicer system.

This new system uses this order (shows the “tug of war” system clearer and makes more sense):



Therefore of course, the pseudocode of this algorithm must change to fit the new system. I will keep the same grid array of 1s for player occupied tiles and 0s for enemy occupied tiles. The tile the battle is about to be on will be called the “target” tile, starting from 0 at tile 1 and 20 at tile 19. Each tile will have its own “id” – a certain number based off the x and y values of the tile. This will be calculated by:

```
Int id(int x, int y)  
Return (y * 5) + x + 1
```

And then the target will just be the index of whichever tile is currently being battled for, e.g. the first one of each war is always tile 9 (which is of course actually tile 10 because arrays start at 0). So, if a battle is won or lost, the following part is run through:

```
IF enemy wins  
    Red tile = grid[(target + 1) MOD 5, (target + 1) DIV 5]  
    Target++  
ELSE IF player wins  
    Green tile = grid[target MOD 5, target DIV 5]  
    Target--  
END IF
```

V3 18/11/19

I have realised that this design is not actually needed thus I have switched back to design V1.

Save/Load [B]

The loading system is the most simple. Because of my file structure system, I can just check from each line for a '#' and split up the strings into an array of values between them (for each line). Then I can just loop through in order and assign each value of the array to each globalData variable for the program to continue running.

```
FUNCTION loadFromFile()
    CHAR splitter = '#'
    FOREACH line
        String[] stringParts = line.Split(separator)
        FOR i = 0 TO 3 STEP 1
            globalData.variableName[i][lineNumber] = Int.Parse(stringParts[i])
            ...
            globalData.anotherVariableName[i][lineNumber] =
            Int.Parse(stringParts[i])
        NEXT
        FOR i = 0 TO 2 STEP 1
            globalData.shorterVariableName[i][lineNumber] =
            Int.Parse(stringParts[i])
        NEXT
    END FUNCTION
```

There will be multiple loops with different number of iterations because not all variables have 4 properties.

The saving system will be much simpler; Manually input each globalData variable into the file, with a hashtag between each one, and a new line when necessary. I could therefore just have a long line that inputs all the values in manually the way I want them to, an it would not be complex as there would be no iterations.

Before all of this actually happens, the player needs to enter the name of the file to save to. This is so that when they want to go and load a file, they know what it is called. Therefore, when they click the save game button, a text box will appear to allow the player to enter the file name. Then, the program will check if this file name already exists or not. If it does, make the user enter another name again. If not, start the saving process above.

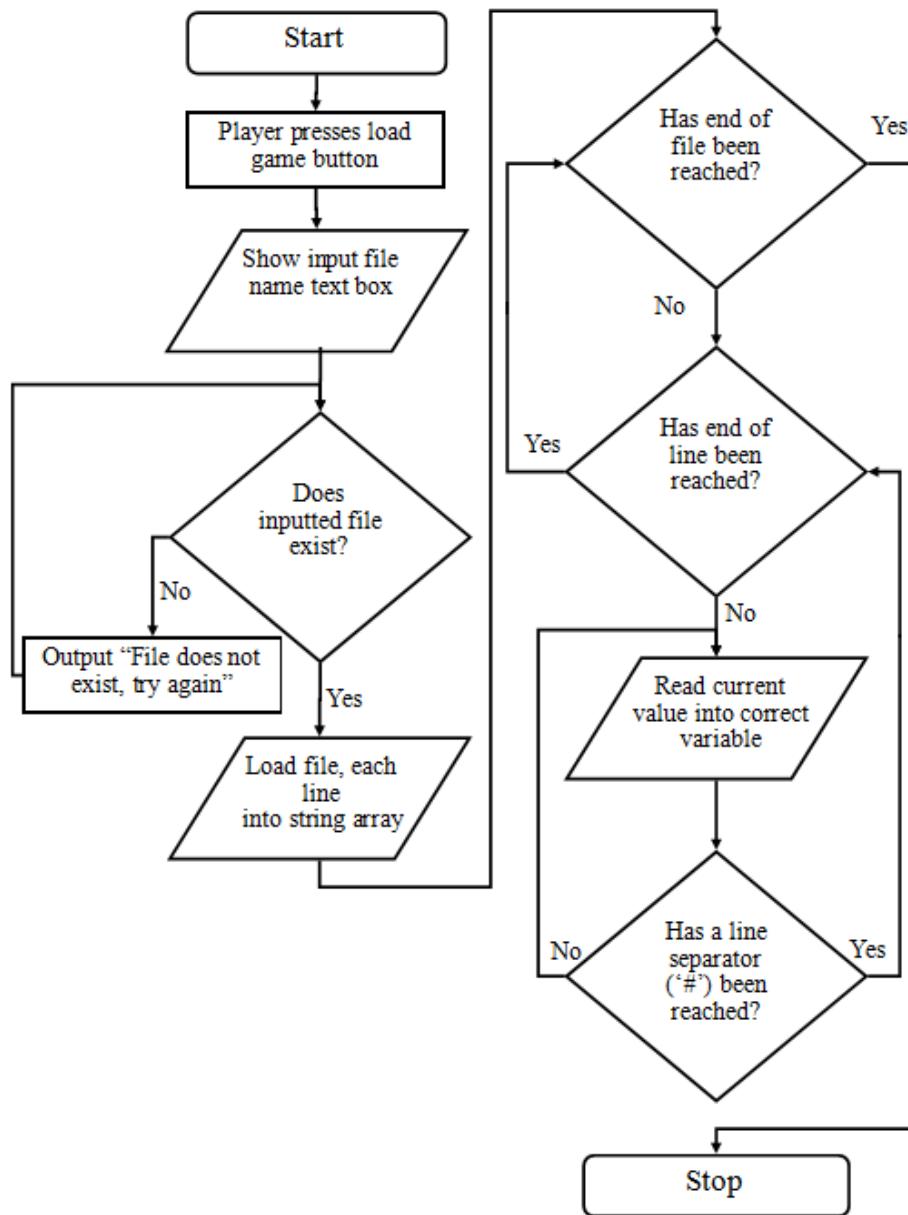
```
FUNCTION checkName()
    String typedName = txt.Text
    FOR i = 0 TO numOfFiles[].LENGTH STEP 1
        IF typedName == numOfFiles[i] THEN
            PRINT "Please enter a new name"
        ELSE
```

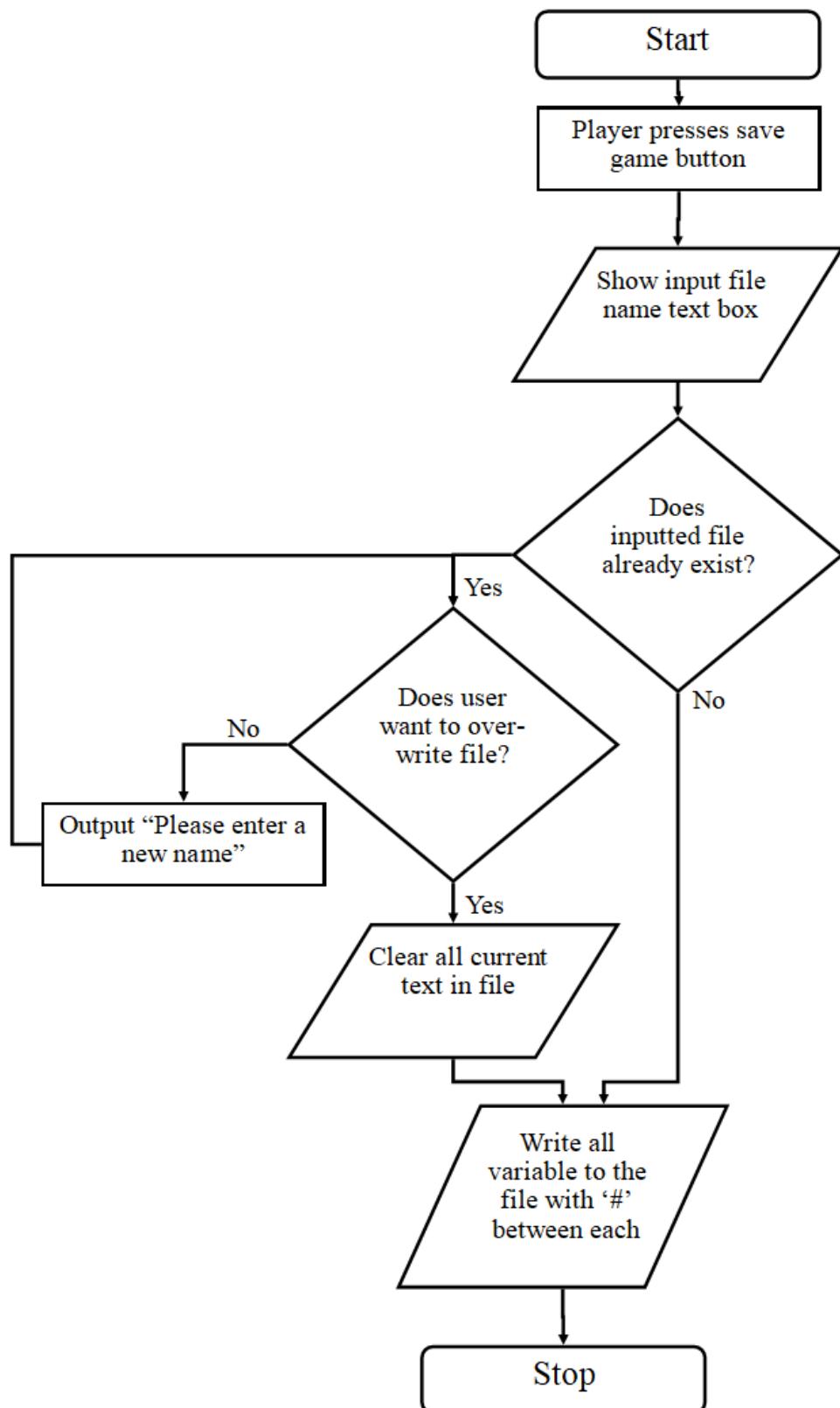
The final part is the autosaving system, and the user being able to toggle it on or off. The process is simple:

1. Every 2 minutes (or 120 ticks), run the savefile function. The autosaving feature will overwrite to a file called something like “autosave.txt”, and save all values the same way it would in a manual save
2. If the user toggles the feature off, this function is no longer run.

```
FUNCTION autosave()
    IF autosaveCounter = 120 && autosaveToggled = TRUE THEN
        autosaveCounter = 0
        savetoFile()
    END IF
END FUNCTION

FUNCTION autosaveToggled()
    IF autosaveToggled = TRUE THEN
        autosaveToggled = FALSE
    ELSE
        autosaveToggled = TRUE
    END IF
END FUNCTION
```

Load from file

Saving to file

Logs [D]

When an event in the program happens, e.g. 1000 metal is reached, or a war is won, a function in the logs class will be called with a certain event number that then prints the correct event type.

E.g. Player wins a war, and gains 1000 food, 5000 science and 2 troops. The following will happen:

```
FUNCTION winWar()
    ...
    logs.StoryEvents(0)
    logs.LootEvents("You have gained " + foodAmount + " food, " + scienceAmount + "
science & " + troopsAmount + " troop(s).")
END FUNCTION

FUNCTION storyEvents(int eventNumber)
    SWITCH (eventNumber)
        CASE 0:
            PRINT currentTime + " You have won a war!"
            BREAK
    END SWITCH
END FUNCTION

FUNCTION lootEvents(string loot)
    PRINT currentTime + " " + loot
END FUNCTION
```

Of course, instead of just printing to one big text box, I will be printing to one of the text boxes for their respective parts, so in this case the story part for winning the war and the loot part with the loot from the war. The reason I am printing the current time at the start of each message is so that the player knows when each thing happens, and if they see these two messages in their respective areas, they will know the messages are linked by the time they were printed. Additionally, the messages will be printed on separate lines, so one timestamped message per line. This means that the old messages are not cleared unless the user chooses to clear them using the clear button. It would look something like this:

```
14:27 You have won a war!
14:41 You have lost a war 😞
15:17 You have lost a war 😞
```

```
14:27 You have gained 1000 food, 5000 science & 2 troop(s).
14:41 You have gained 500 food & 5000 science but lost 2 troop(s).
15:17 You have gained 500 metal & 5000 science but lost 4 troop(s).
```

If the player wants to clear, lets say, the loot text box only, then they will have to click on the loot tab (to show the current messages) and click the clear button. The function below would be run every time this button is pressed.

```
FUNCTION clearText()
    currentTab = Tab.SELECTEDTAB
    SWITCH currentTab
        CASE "loot":
            rtxtLoot.TEXT = ""
        CASE "story":
            rtxtStory.TEXT = ""
        CASE "combat":
            rtxtCombat.TEXT = ""
        CASE "upgrades":
            rtxtUpgrades.TEXT = ""
        CASE "saves":
            rtxtSaves.TEXT = ""
    END SWITCH
END FUNCTION
```

Variables Table

Variable Name	Data type	What it stores	Why it is needed
Global data variables			
resourcesData	List of integer arrays	All information about wood, stone, food and metal – amount, rate, capacity and gather multiplier	Compactly stores all resource information with easy access using an iterator
scienceAmount	Integer	The amount of science the player has	To store the amount of science the user has for buying upgrades
housingData	Integer array	How much housing space each type of housing gives	So that the upgrades can change this based on how much each housing type adds
totalHousing	Integer	The amount of housing the player owns	To work out housing remaining
housingRemaining	Integer	The amount of housing space the player has left (that can be filled with workers/troops)	So that the upgrades system knows how many more workers/troops can be added
upgradesCosts	List of integer arrays	The costs for each storage type, worker type, science upgrade, housing type and combat type	So that the upgrades menu can display each cost and work out if the player has enough resource to buy something
costMultipliers	Integer array	Stores the increase in cost for each upgrade type	So that when an upgrade is bought the cost of the upgrade is increased
combatData	List of integer arrays	Stores the player and enemy combat data for block, attack, health and the number of troops	Compactly stores all the resource information with easy access using an iterator
tickCounter	Integer	Counts the number of ticks that have passed whilst the program has run	Used to work out if it is time for a new war to run
warTimes	Integer array	The time of each war in ticks	Allows the program to compare if the value of the tickCounter is equal to one of the values in warTimes in order to know if a knew war can start
currentWar	Integer	The time of the current war (if any) in ticks	So that when the game is saved it knows when the current war is happening

Resource collection variables [Section A]			
btnToggled	Boolean array	Stores if a button is toggled or not (true is toggled, false is not)	So that the program knows the state of each button
currentRate	Integer	Locally stores the rate of a certain resource at the time of the variable being assigned	To be used and manipulated in the resourceCollection function – so that the global data variable isn't changed
currentAmount	Integer	Locally stores the amount of a certain resource at the time of the variable being assigned	To be used and manipulated in the resourceCollection function – so that the global data variable isn't changed
currentResource	Integer	Locally stores the current resource being worked on	To be used and manipulated in the resourceCollection function – so that the global data variable isn't changed
currentHousing	Integer	Locally stores the amount of housing space that remains	To be used and manipulated in the resourceCollection function – so that the global data variable isn't changed
Upgrades variables [Section A]			
cost	Integer	Locally stores the cost of the upgrade	To be used and manipulated in the resourceCollection function – so that the global data variable isn't changed
typeToBuyFor	Integer	Locally stores the type of resource the upgrade is bought for	To be used as a parameter for each different upgrade in its respective function so that many similar functions do not have to be made
typeToBuy	Integer	Locally stores the type of resource that is being used to buy the upgrade	To be used as a parameter for each different upgrade in its respective function so that many similar functions do not have to be made
resourceProperty	Integer	Locally stores the property of a resource e.g. amount	To be used as a parameter for each

			different upgrade in its respective function so that many similar functions do not have to be made
storageIncrease	Integer	The amount of storage the upgrade increases it by	To be used as a parameter for each different upgrade in its respective function so that many similar functions do not have to be made
housingType	Integer	The type of housing that is being upgraded	To be used as a parameter for each different upgrade in its respective function so that many similar functions do not have to be made
combatType	Integer	The type of combat that is being upgraded	To be used as a parameter for each different upgrade in its respective function so that many similar functions do not have to be made
buff	Integer	The amount that the combat type is being increased by	To be used as a parameter for each different upgrade in its respective function so that many similar functions do not have to be made
Combat variables [Section C]			
rows	Constant Integer	The number of rows that the grid will have (4 rows)	The program knows how many rows it needs to draw
columns	Constant Integer	The number of columns that the grid will have (5 columns)	The program knows how many columns it needs to draw
grid	2D Integer Array	Stores the grid in 1s and 0s. 1 is player owned, 0 is enemy owned	To store the state of the grid for working out stats etc
bitmap	Bitmap	This will allow for the image to be displayed correctly in the picture box	Bitmap objects is needed to display the drawn images in the correct resolution for any computer
GFX	Graphics	This will use the bitmap object to display the image	Needed to actually display the image

greenBrush	SolidBrush	Green brush to fill the rectangles with	Needed to colour the bottom 2 rows green (to mark player)
redBrush	SolidBrush	Red brush to fill the rectangles with	Needed to colour the top 2 rows red (to mark enemy)
brushes	SolidBrush array	An array storing the red brush and green brush	So that the program can draw a default grid of boxes using these
checkSquare	Integer	The top left square of the grid (coordinates of 0, 0)	So that the program knows if the player owns this tile (thus has won)
allEqual	Boolean	A boolean that stores the state of the board (if all grid tiles are the same, set it to true)	So that the program knows if the player or enemy has won
tileSize	Constant Integer	Sets the size of each side of the tile	The program knows how large to draw each tile
offset	Integer	The number of tiles the player owns outside of the default amount (of 10), positive if the player owns more than the enemy, negative if the enemy owns more than the player	Used to recolour the correct tiles, and used for outputted data on the side
playerTurn	Boolean	Stores if it is the player's turn next or not (true if it is, false if not)	So the program knows which values to calculate
playerDamage	Integer	Locally stores the damage the player does to the enemy on that turn	To be used as a parameter for the attacking calculations
enemyDamage	Integer	Locally stores the damage the enemy does to the player on that turn	To be used as a parameter for the attacking calculations
enemyHealth	Integer	Stores the enemy health during that turn	To be used as a parameter for the attacking calculations
playerHealth	Integer	Stores the player health during that turn	To be used as a parameter for the attacking calculations
blockedPlayerDamage	Integer	Locally stores the playerDamage – enemyBlock on that turn	To be used as a parameter for the attacking calculations
blockedEnemyDamage	Integer	Locally stores the enemyDamage – playerBlock on that turn	To be used as a parameter for the attacking calculations

rng	Random	An object that can be used to select random numbers	So that the program can use random number generating
randomNum	Integer	Locally stores a random number based off selected ranges using the rng object	To be used for any local random needs
Logs variables [Section D]			
eventType	Integer	Locally stores the type of event that is being called	So that the program knows which even is currently being run to display the correct output
loot	String	Locally stores any loot, as a string, that has been called	So that it can be outputted in logs
currentTime	DateTime	Locally stores the time, in UTC	To show the timestamps of each message
File handling variables [Section B]			
filePath	String	Stores the local file path of the save files	Can be used to get the local directory of the program so that the save files can be read/written to/from
line	String	Locally stores the current line being read from the file	So that the program knows which line is being read from
separator	Char	Locally stores the character that is used to separate each string up (#)	So that the program knows which character is being used to separate data in a line
stringParts	String array	Locally stores each string part of the line excluding the separator	To be converted into the correct variable
lineNumber	Integer	Locally stores the current line number of the file in order to store the values in	So that the program knows which line is being read from
typedName	String	Locally stores the file name that the user has entered	So that the program can assign a custom name instead of generating bad ones
numOfFiles	String	Locally stores the number of files that the save files directory has	So that the program can loop through this number when checking for any files the same name as the typedName
autosaveToggled	Boolean	Stores whether or not the autosave feature is on or off	So that the program knows the state of the autosaves (for save files)

autosaveCounter	Integer	Stores the number of ticks that have passed	So the program knows when to autosave next
------------------------	---------	---	--

Testing Table

Note: In testing, I will employ the traffic light success system;

- Green – Full success
- Orange – It mostly works but needs a small fix
- Red – It does not work

Alpha Test Table

Input (or question)	Expected Output
	Usability Tests
How many max clicks does it take to get around from any parts of the program to any other part	Combat logs tab to deleting save game. To get there: click on save/load button on menu strip (1) -> click delete save button (2) -> click “yes” button (3)
The resource collection button on the menu strip is clicked	The resources menu is shown with each part for wood, food, stone, metal and housing
The upgrades button on the menu strip is clicked	The upgrades menu is shown with each tab for workers, storage, research, housing and combat
The combat button on the menu strip is clicked	Show the combat grid, the health bars and the other side features on the form
The save/load button on the menu strip is clicked	Show the save/load menu with the autosave, delete, save and load buttons
The logs button on the menu strip is clicked	Show the logs menu with the different logs tabs
Resource Collection Tests [Section A]	
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by 1/sec, and the colour goes back to grey (button is toggled off)
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and food rate is increased by 1/sec (button is toggled on)
The wood progress bar reaches maximum	The wood value no longer increases and the progress bar stays filled.
A worker upgrade is bought	The rate of increase increments
A storage upgrade is bought	Maximum amount of the resource is increased, causing the amount in the progress bar to be compressed into a small space
A housing type or upgrade is bought	The amount of free housing space is shown by the progress bar, and the amount left is increased in the amount label
A battle is won/lost/tied	Nothing happens

A war is won/lost/tied	Rate increases, science increases, other possible parts increase or decrease depending on the outcome and loot gained
Upgrades Tests [Section A]	
There are not enough resources to buy wood storage	Print "You do not have enough resources for this" in upgrades logs, upgrade is not bought
It costs 50 wood to buy wood storage, user has exactly 50 wood, buys storage	Wood amount goes down to 0, the upgrade goes through, the storage is increased by 1000, storage cost increased by 3x
It costs 50 wood to buy wood storage, user has reached full capacity of the wood storage, buys storage	Wood amount goes down by 50, the upgrade goes through, the storage is increased by 1000, so progress bar is pushed further back, storage cost increased by 3x
It costs 900 wood to buy metal storage, user has reached full metal capacity, buys storage	Wood amount goes down by 900, the upgrade goes through, the metal storage is increased by 500, so progress bar of both wood and meal is pushed further back, storage cost increased by 3x
Food storage upgrade is bought	The maximum of the wood stays the same, maximum of the food increases by 1000
The user buys wood storage, then buys stone storage, then metal storage	Wood amount goes down by 50 for wood storage, 50 for stone, 100 for metal. Wood, stone & metal storages are increased by 1000, 1000 and 500 respectively
The troop upgrade is bought (enough housing and metal)	The amount of metal goes down by cost, the housing amount goes down by 1, number of troops increased by 1, cost increased by 4 times
There is not enough food (but there is enough housing space) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought
There is not enough housing space (but there is enough food) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought
There is not enough food or housing space to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough housing space and food for all 4	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the miner upgrade, the upgrade cost is increased by 4 times, the rate of metal goes up by 1/sec. Housing space goes down by 1. The metal goes down by the cost of the troop upgrade, the upgrade cost is increased by 4 times, the number of troops is increased by 1. Housing space goes down by 1
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough housing space but only enough food for the first 2 upgrades	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The

	miner and troop upgrades cannot be bought as is not enough food available. Print "You do not have enough resources or housing space for this" in upgrades logs
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough food for all 4 but only enough housing space for the first 3 upgrades	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The miner and troop upgrades cannot be bought as is not enough housing space available. Print "You do not have enough resources or housing space for this" in upgrades logs
Food has reached full capacity, quarrier upgrade is bought	Food amount goes down by cost of upgrade, housing space goes down by 1, stone rate increased by 1/sec
Metal has reached full capacity, troop upgrade is bought	Metal amount goes down by cost of upgrade, housing space goes down by 1, troop amount increased by 1
Food storage upgrade is bought, then butcher upgrade	Wood amount goes down by cost of food storage, food capacity goes up by 1000, butcher upgrade bought so food amount goes down by cost, housing space reduced by 1 and food rate increased by 1/sec
Food has reached just about enough to buy gatherer upgrade, housing space left is 1	Food and housing goes down to 0, food rate increased by 1/sec
The shack is bought for 50 stone	50 stone deducted, housing space increased by 5, cost increased by 2x
The bootcamp is bought for 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x
The barracks is bought for 500 stone	500 stone deducted, housing space increased by 20, cost increased by 2x
The user tries to buy shack upgrade, but there is not enough stone	Upgrade does not go through, print not enough resources in upgrades logs
The user tries to buy shack upgrade, has enough stone, but then tries to buy barracks and there is not enough stone	50 stone deducted, housing space increased by 5, cost increased by 2x, print not enough resources in upgrades logs when trying to buy barracks upgrade
The user tries to buy shack upgrade, has enough stone, but then tries to buy shack again but there is not enough stone	50 stone deducted, housing space increased by 5, cost increased by 2x, print not enough resources in upgrades logs when trying to buy stone upgrade
The user buys bootcamp 3 times from base price of 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x. 200 stone deducted, housing space increased by 10, cost increased by 3x. 400 stone deducted, housing space increased by 10, cost increased by 2x
The user tries to buy shack upgrade, has enough stone, but then tries to buy shack again but there is not enough stone (more than 50 though)	50 stone deducted, housing space increased by 5, cost increased by 2x, print not enough resources in upgrades logs when trying to buy stone upgrade
Troop health upgrade bought	Metal reduced by cost, health goes up by 100
Troop block upgrade bought	Metal reduced by cost, block goes up by 250

Troop attack upgrade bought	Metal reduced by cost, attack goes up by 300
Troop health upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console
Troop block upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console
Troop attack upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console
Troop health upgrade bought 3 times from base value of 200 metal	Metal reduced by 200, health goes up by 100. Metal reduced by 400, health goes up by 100. Metal reduced by 800, health goes up by 100.
Troop block upgrade bought 3 times from base value of 200 metal	Metal reduced by 500, block goes up by 250. Metal reduced by 1000, block goes up by 250. Metal reduced by 2000, block goes up by 250.
Troop attack upgrade bought 3 times from base value of 200 metal	Metal reduced by 300, attack goes up by 300. Metal reduced by 600, attack goes up by 300. Metal reduced by 1200, attack goes up by 300.
Aqueducts upgrade bought	Science reduced by cost, food and wood rate go up by 5x, cost multiplied by 5x
Stamp-mill upgrade bought	Science reduced by cost, stone rate goes up by 2x, cost multiplied by 5x
Trip-hammer upgrade bought	Science reduced by cost, metal rate goes up by 2x, cost multiplied by 5x
Hushing upgrade bought	Science reduced by cost, metal rate goes up by 5x, cost multiplied by 5x
Villa upgrade bought	Science reduced by cost, housing space goes up by 3x, cost multiplied by 5x
Aqueducts upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Stamp-mill upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Trip-hammer upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Hushing upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Villa upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Aqueducts research bought 3 times from base value of 100 science, all at +1/sec	Science reduced by 100, food and wood rate go up to +5/sec. Science reduced by 500, food and wood rate go up to +25/sec. Science reduced by 2500, food and wood rate go up to +125/sec.
Stamp-mill research bought 3 times from base value of 500 science, all at +1/sec	Science reduced by 500, stone rate goes up to +2/sec. Science reduced by 2500, stone rate goes up to +4/sec. Science reduced by 12500, stone rate goes up to +8/sec.
Trip-hammer research bought 3 times from base value of 1000 science, all at +1/sec	Science reduced by 1000, metal rate goes up to +2/sec. Science reduced by 5000, metal rate goes up to +4/sec. Science reduced by 25000, metal rate goes up to +8/sec.
Hushing research bought 3 times from base value of 3000 science, all at +1/sec	Science reduced by 3000, metal rate goes up to +5/sec. Science reduced by 15000, metal rate goes up to +25/sec. Science reduced by 75000, metal rate goes up to up to +125/sec.

Villa research bought 3 times from base value of 5000 science, housing at 5	Science reduced by 5000, housing space increased to 15. Science reduced by 25000, housing space increased to 45. Science reduced by 125000, housing space increased to 135.
Logs Tests [Section D]	
Some text is in the story logs, clear button pressed	Only text in the story logs is cleared
Some text is in the loot logs, clear button pressed	Only text in the loot logs is cleared
Some text is in the upgrades collection logs, clear button pressed	Only text in the upgrades collection logs is cleared
Some text is in the combat logs, clear button pressed	Only text in the combat logs is cleared
Some text is in the saves logs, clear button pressed	Only text in the saves logs is cleared
Combat Tests [Section C]	
300 seconds (ticks) have passed since the game first loaded up (not from save file)	A new war starts, the health, attack and block labels are set to the correct values, the year and war name is shown, the grid is in the default state, new war message printed into combat logs
New war starts. Player has 100 health, 50 attack, 20 block, enemy has 200 health, 25 attack, 40 block	First round should go like this: <ol style="list-style-type: none"> 1. Player does 10 damage to enemy 2. Enemy does 5 damage to player 3. Repeat until all tiles are won Player wins, loot given to player, board and labels reset, outcome printed to combat logs, loot printed to loot logs.
New war starts. Both player and enemy have exact same stats.	The war will be a stalemate, with no sides gaining any territory at all. The draw is called, player gets appropriate loot, board and labels reset, outcome printed to combat logs, loot printed to loot logs.
New war starts. Player has the upper hand so beats the enemy.	The player receives winning loot. Board reset, labels reset, winning message and loot gained printed into loot logs.
A war has ended, another few ticks pass until a new war begins again.	Everything resets properly, new war message printed into combat logs.
Enemy wins tiles	Grid updated to show one more red tile
Player wins tiles	Grid updated to show one more green tile
Saving/Loading Tests [Section B]	
Save game button is pressed	Text box and submit button to enter filename to save to appears, load game and delete game buttons hide
Filename “asave” is entered and submit button pressed after save game button pressed	A new file is created in the executable file path called “asave.txt” with all the appropriate values inside. Output file saved successfully into logs and output box
Filename “asave.txt” is entered	A new file called “asave.txt.txt” is created with all the appropriate values inside. Output file saved successfully into logs and output box
Filename “478787” is entered	A new file called “478787.txt” is created with all the appropriate values inside. Output file saved successfully into logs and output box

Filename “bobby45800.txt” is entered	A new file called “bobby45800.txt.txt” is created with all the appropriate values inside. Output file saved successfully into logs and output box
Filename containing any of the following characters in its name is entered: “\/*?<> []@#{}”	The output box says that the file cannot be created because one of the invalid filename characters has been entered. Output file saved successfully into logs and output box
A file called “gavgood” is saved, and then the same name is entered for the next one	The file “gavgood” is saved, then the next time it is overwritten into the same file
Load game button is pressed	List box showing all possible files to load from appears with submit button for it. Save and delete games buttons hide
A file in the list box is selected and submit button is pressed	The game pauses whilst the information in the file is read from and put into the global data variables. Logs output that a file has been loaded into the game
Nothing is selected in the list box when the submit button is pressed	The output box tells the user to select a file to load from
The delete game button is pressed	The select file list box, submit button show
A file is selected, submit button is pressed	The “are you sure you want to delete this” panel show (with yes and no buttons inside)
A file is not selected, submit button is pressed	Output in box that a file must be selected to delete
The yes button is pressed	The selected file is deleted permanently, output this to user in logs
The no button is pressed	The selected file is saved, output this to user in logs
The delete button is pressed, file is deleted, then the delete button is pressed again	The selected file is deleted, output to logs. Next time files are shown, the deleted file no longer appears.
Game is first loaded up, 5 minutes (300 ticks) pass	First autosave happens, to file “autosave.txt”. If it already exists, overwrite file. Show autosave has been made in logs
The autosave button is pressed, autosave timer currently on	The autosave timer is toggled off, output that to logs and label
The autosave button is pressed, autosave timer currently off	The autosave timer is toggled on, output that to logs and label
A new file called “newfile” is made, then deleted	File called “newfile” created, all data saved into it. File deleted when file selected and submit button pressed. All proceedings printed to saves logs
A new file called “newfile” is made, then loaded from	File called “newfile” created, all data saved into it. File loaded from when file selected and submit button pressed. All proceedings printed to saves logs

Beta Test Table**Usability Tests**

Question
Can they switch between tabs easily?
Can they press a button to toggle faster rate for that resource?
Can they press it again to toggle it off?
Can they buy a storage upgrade if they have enough wood?
Can they buy a worker upgrade if they have enough food?
Can they buy a housing upgrade if they have enough stone?
Can they buy a combat upgrade if they have enough metal?
Can they buy a science upgrade if they have enough science points?
First war starts, can they work out what is going on in the combat menu?
If they do not have enough of a resource for an upgrade, can they easily find out why they cannot buy the upgrade from logs?
If they win/lose/draw a war, can they find the loot gained/outcome in logs?
If they save, load or the game autosaves, can they find the messages in logs?
Can they save the game to any file location?
Can they load the game from any file location?
Can they delete a save?
Can they toggle autosaving on or off?

Trying to break the game tests

Input (or question)	Expected Output
Resource Collection Tests [Section A]	
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by 1/sec, and the colour goes back to grey (button is toggled off)
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and food rate is increased by 1/sec (button is toggled on)
The wood progress bar reaches maximum	The wood value no longer increases and the progress bar stays filled.
A worker upgrade is bought	The rate of increase increments
A storage upgrade is bought	Maximum amount of the resource is increased, causing the amount in the progress bar to be compressed into a small space
A housing type or upgrade is bought	The amount of free housing space is shown by the progress bar, and the amount left is increased in the amount label
A battle is won/lost/tied	Nothing happens
A war is won/lost/tied	Rate increases, science increases, other possible parts increase or decrease depending on the outcome and loot gained

Upgrades Tests [Section A]	
There are not enough resources to buy wood storage	Print "You do not have enough resources for this" in upgrades logs, upgrade is not bought
It costs 50 wood to buy wood storage, user has reached full capacity of the wood storage, buys storage	Wood amount goes down by 50, the upgrade goes through, the storage is increased by 1000, so progress bar is pushed further back, storage cost increased by 3x
Food storage upgrade is bought	The maximum of the wood stays the same, maximum of the food increases by 1000
The troop upgrade is bought (enough housing and metal)	The amount of metal goes down by cost, the housing amount goes down by 1, number of troops increased by 1, cost increased by 4 times
There is not enough food (but there is enough housing space) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought
There is not enough housing space (but there is enough food) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought
There is not enough food or housing space to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought
Food has reached full capacity, quarrier upgrade is bought	Food amount goes down by cost of upgrade, housing space goes down by 1, stone rate increased by 1/sec
Metal has reached full capacity, troop upgrade is bought	Metal amount goes down by cost of upgrade, housing space goes down by 1, troop amount increased by 1
Food storage upgrade is bought, then butcher upgrade	Wood amount goes down by cost of food storage, food capacity goes up by 1000, butcher upgrade bought so food amount goes down by cost, housing space reduced by 1 and food rate increased by 1/sec
Food has reached just about enough to buy gatherer upgrade, housing space left is 1	Food and housing goes down to 0, food rate increased by 1/sec
The shack is bought for 50 stone	50 stone deducted, housing space increased by 5, cost increased by 2x
The bootcamp is bought for 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x
The barracks is bought for 500 stone	500 stone deducted, housing space increased by 20, cost increased by 2x
The user tries to buy shack upgrade, but there is not enough stone	Upgrade does not go through, print not enough resources in upgrades logs
Troop health upgrade bought	Metal reduced by cost, health goes up by 100
Troop block upgrade bought	Metal reduced by cost, block goes up by 250
Troop attack upgrade bought	Metal reduced by cost, attack goes up by 300
Troop health upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console
Troop block upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console
Troop attack upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console
Aqueducts upgrade bought	Science reduced by cost, food and wood rate go up by 5x, cost multiplied by 5x

Stamp-mill upgrade bought	Science reduced by cost, stone rate goes up by 2x, cost multiplied by 5x
Trip-hammer upgrade bought	Science reduced by cost, metal rate goes up by 2x, cost multiplied by 5x
Hushing upgrade bought	Science reduced by cost, metal rate goes up by 5x, cost multiplied by 5x
Villa upgrade bought	Science reduced by cost, housing space goes up by 3x, cost multiplied by 5x
Aqueducts upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Stamp-mill upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Trip-hammer upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Hushing upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Villa upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs
Logs Tests [Section D]	
Some text is in the story logs, clear button pressed	Only text in the story logs is cleared
Some text is in the loot logs, clear button pressed	Only text in the loot logs is cleared
Some text is in the upgrades collection logs, clear button pressed	Only text in the upgrades collection logs is cleared
Some text is in the combat logs, clear button pressed	Only text in the combat logs is cleared
Some text is in the saves logs, clear button pressed	Only text in the saves logs is cleared
Combat Tests [Section C]	
300 seconds (ticks) have passed since the game first loaded up (not from save file)	A new war starts, the health, attack and block labels are set to the correct values, the year and war name is shown, the grid is in the default state, new war message printed into combat logs
New war starts. Player has the upper hand so beats the enemy.	The player receives winning loot. Board reset, labels reset, winning message and loot gained printed into loot logs.
A war has ended, another few ticks pass until a new war begins again.	Everything resets properly, new war message printed into combat logs.
Enemy wins tiles	Grid updated to show one more red tile
Player wins tiles	Grid updated to show one more green tile
Saving>Loading Tests [Section B]	
Save game button is pressed	Text box and submit button to enter filename to save to appears, load game and delete game buttons hide
Filename "asave" is entered and submit button pressed after save game button pressed	A new file is created in the executable file path called "asave.txt" with all the appropriate values inside. Output file saved successfully into logs and output box
Filename "asave.txt" is entered	A new file called "asave.txt.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box

Filename “478787” is entered	A new file called “478787.txt” is created with all the appropriate values inside. Output file saved successfully into logs and output box
Filename “bobby45800.txt” is entered	A new file called “bobby45800.txt.txt” is created with all the appropriate values inside. Output file saved successfully into logs and output box
Filename containing any of the following characters in its name is entered: “\:*?<> []@#{}”	The output box says that the file cannot be created because one of the invalid filename characters has been entered. Output file saved successfully into logs and output box
A file called “gavgood” is saved, and then the same name is entered for the next one	The file “gavgood” is saved, then the next time it is overwritten into the same file
Load game button is pressed	List box showing all possible files to load from appears with submit button for it. Save and delete games buttons hide
A file in the list box is selected and submit button is pressed	The game pauses whilst the information in the file is read from and put into the global data variables. Logs output that a file has been loaded into the game
Nothing is selected in the list box when the submit button is pressed	The output box tells the user to select a file to load from
The delete game button is pressed	The select file list box, submit button show
A file is selected, submit button is pressed	The “are you sure you want to delete this” panel show (with yes and no buttons inside)
A file is not selected, submit button is pressed	Output in box that a file must be selected to delete
The yes button is pressed	The selected file is deleted permanently, output this to user in logs
The no button is pressed	The selected file is saved, output this to user in logs
The delete button is pressed, file is deleted, then the delete button is pressed again	The selected file is deleted, output to logs. Next time files are shown, the deleted file no longer appears.
Game is first loaded up, 5 minutes (300 ticks) pass	First autosave happens, to file “autosave.txt”. If it already exists, overwrite file. Show autosave has been made in logs
The autosave button is pressed, autosave timer currently on	The autosave timer is toggled off, output that to logs and label
The autosave button is pressed, autosave timer currently off	The autosave timer is toggled on, output that to logs and label
A new file called “newfile” is made, then deleted	File called “newfile” created, all data saved into it. File deleted when file selected and submit button pressed. All proceedings printed to saves logs
A new file called “newfile” is made, then loaded from	File called “newfile” created, all data saved into it. File loaded from when file selected and submit button pressed. All proceedings printed to saves logs

3. Development

For development, the process I will go through to get a beta-ready game is:

1. Building the windows forms GUI & coding a section and testing each part until it works
2. Checking with stakeholders about how they feel with each part; if they have any changes they want, make changes and retest/give new part to stakeholder
3. Building the next section and testing each part until it works, which may then include retesting previous sections that are linked to the current one
4. Check whole program with stakeholders, make any changes based on feedback
5. Retesting the entire program for minor bugs

If I work on the resource collection GUI, and then need to come back afterwards to make any changes, I will make a new heading underneath that section and increase the version number.

Main Setup 03/10/19

Following my design of the sub-program design, I have setup the MainForm class as shown

```
public partial class MainForm : Form {  
    public MainForm() {  
        InitializeComponent();  
        Point spawnLocation = new Point(0, 0);  
        Location = spawnLocation;  
    }  
  
    //Some code  
}
```

The purpose of the spawn location code is to make sure that the game loads up in the position of 0, 0, because I have noticed that loading this into other computers without this makes the form load into inconsistent places.

Now, with the use of the partial classes, I can separate my code throughout multiple different files, e.g. with the combat file I can just do

```
public partial class MainForm {  
    void Button1Click(object sender, EventArgs e) {  
        MessageBox.Show("This button has been clicked");  
    }  
}
```

Calling the class MainForm again just extends the MainForm class onto a new file.

GlobalData Setup 03/10/19

Because I know what variables I need for my GlobalData class, I can already put them all in. Of course I can add or remove variables as I develop the game, as I may find I need more or fewer to be saved in my files. Here is a snippet of the code inside my GlobalData class

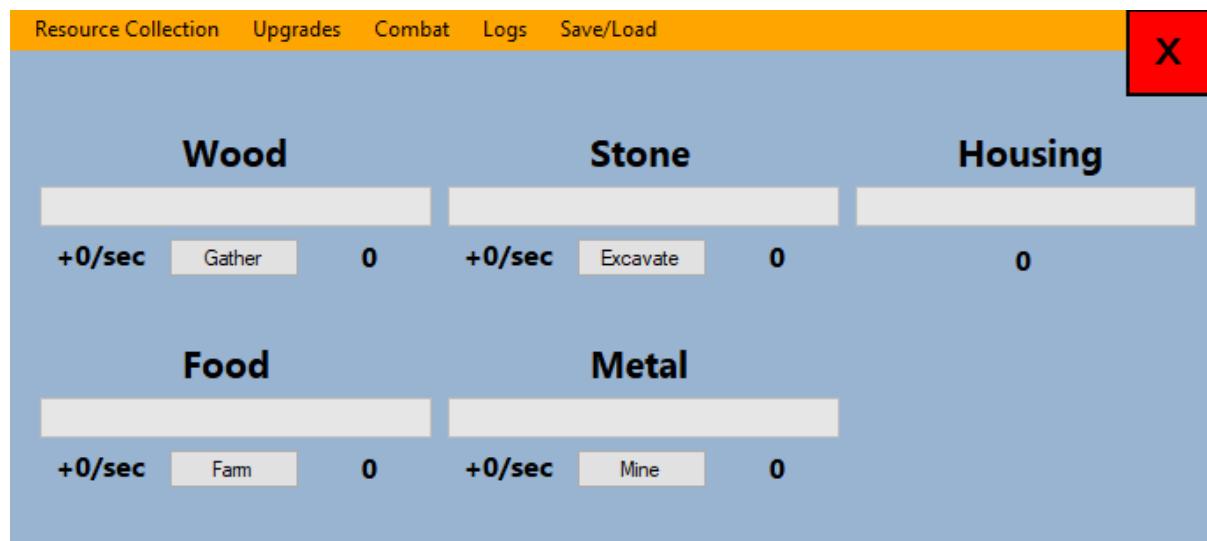
```
public static class GlobalData {
    //Resource data
    //[[0]Wood, [[1]Stone, [[2]Food, [[3]Metal
    public static List<int[]> resourcesData = new List<int[]> {
        new int[4] {0, 0, 0, 0}, //Amount
        new int[4] {1, 1, 1, 1}, //Rate, in milliseconds, by the timer (which
        is timed by 1000 later)
        new int[4] {100, 100, 100, 100}, //Capacity
        new int[4] {1, 1, 1, 1} //Gather multiplier (e.g. 2 would be *2/sec)
    };
    public static int scienceData = 0;
    public static int[] housingData = new int[3] {5, 10, 15}; //How much space
    each housing type gives
    public static int totalHousing = 0; //Total housing space
    public static int housingRemaining = 0;

    ...
}
```

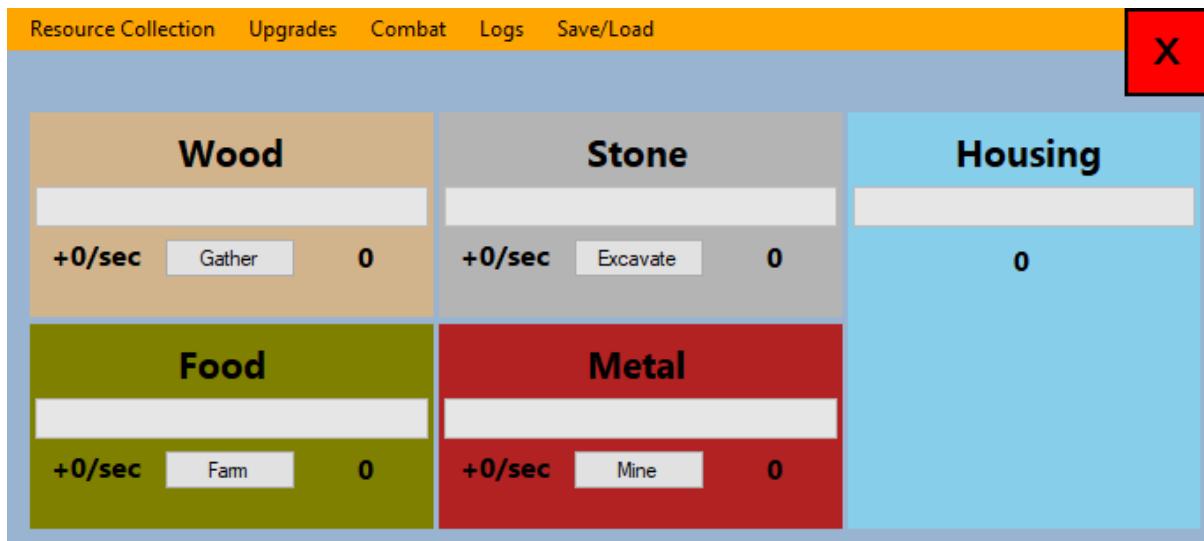
Resource Collection GUI [A]

V1 05/10/19

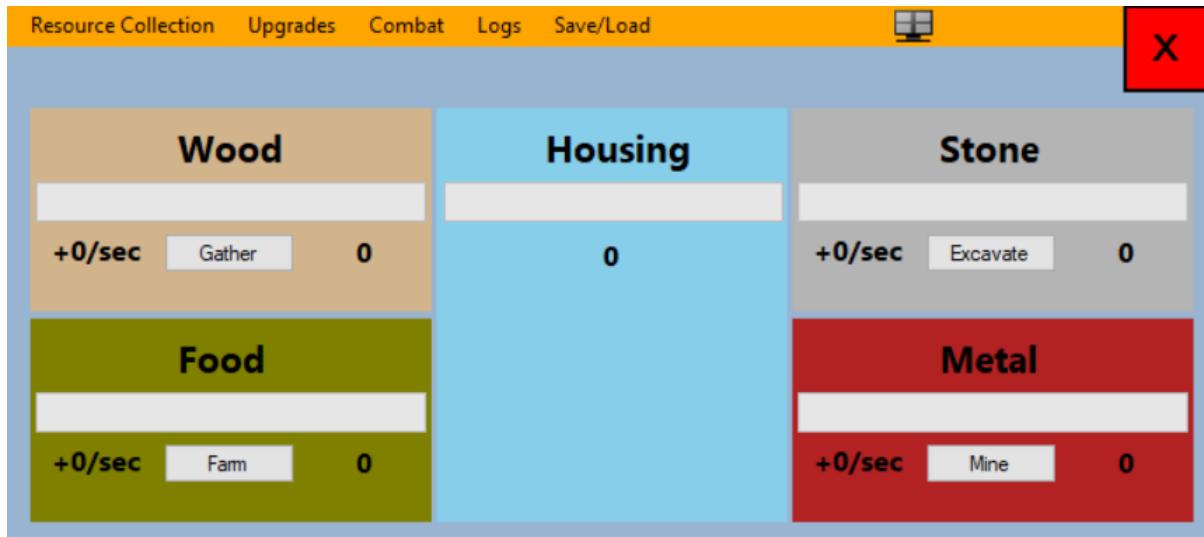
First I need to build the GUI for the resource collection menu. I will build it off what my stakeholders and I agreed upon in the design. Here's the first version based entirely off the design schematic:



My stakeholders had some feedback. First, they didn't know what the number on the right of each button did. This is not something I will need to fix, as they will notice it is the resource amount as soon as the program loads up, because it will begin to increment immediately. The second part is more aesthetical – they wanted the background of each resource type to have their own respective colour. E.g. wood with a tan background, stone with grey etc. So here is the next design:

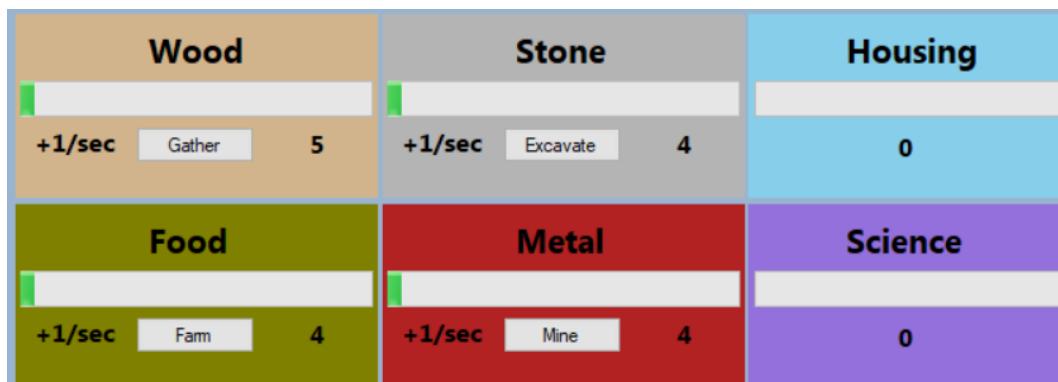


Now my stakeholders think that it looks weird to have the housing part on the end. They think it looks better to have this:



V2 18/10/19

My stakeholders are now happy about this. However, Christian has come up with a query – why isn't there an amount bar in the resource collection tab for science, even though it is acquired from combat? The housing is acquired from the upgrades, and there needs to be a way for the user to know how much of each resource they have easily. Therefore, I have added a new part in the resource collection section for the science.



V3 23/10/19

When working on the housing upgrades code, I noticed that the housing bit of the resources section was not fully complete, as it didn't show both the housing capacity and housing remaining in number form. So I made the simple change:

Wood	Stone	Housing
+1/sec Gather 2	+10/sec Excavate 20	0 housing capacity 0 housing remaining
Food	Metal	Science
+1/sec Farm 2	+1/sec Mine 2	0

V4 24/10/19

When working on the worker upgrades part, I set the rate of the stone to +100/sec, and this happened:



The label is not large enough to cope with the rate. The rate will most likely get this high, or even perhaps into the thousands. Therefore, to fit these extra characters in, I have opted to make a compromise on the size of the text in the labels, making them size 10 instead of size 12. With this, I will also change the sizes of the amount labels to match the rate labels for the sake of consistency. My stakeholders pointed out that I could just extend the labels downwards to fit the bottom left corner of each panel squarely – this was actually a good idea, and so I went ahead with it and changed the font size back to 12. Now, my rate labels can fit up to +999999999/sec (99 million per second), although the value should never reach that high.

Additionally, the amount labels could only fit up to 6 digits – or 999,999. Now, they can fit up to 12 digits, or 999 billion. However, if the values were to ever reach that high, I would definitely code the milestones section that is currently set to optional (see A3 in success criteria) so that the user can more easily see exactly how many resources they have with the use of symbols, K, M, B etc.

The only downside to this change is that when the rate is between 100 and 999, the “c” from “sec” is forced onto the line below, it looks odd:

+100/se
c

It is a compromise the stakeholders have said they are willing to make.

Resource Collection Code [A]

V1 09/10/19

Now that my stakeholders are happy with the GUI for resource collection, I need to begin to code the sub program. First, I will begin by coding the buttons, then I will move onto the actual resource collection, followed by the housing calculations.

Buttons

I have written the button code directly from the pseudocode in design. However, I have decided to temporarily include the change button colour code into the toggle button. This will make it much easier to tell if a button has been toggled on or not for testing purposes before I can then implement it with my main resource collection function later on.

```
bool[] btnToggled = new bool[4] {false, false, false, false};

void BtnGatherClick(object sender, EventArgs e) {
    toggleButton(0, btnGather);
}

void BtnQuarrierClick(object sender, EventArgs e) {
    toggleButton(1, btnQuarrier);
}

void BtnFarmClick(object sender, EventArgs e) {
    toggleButton(2, btnFarm);
}

void BtnMineClick(object sender, EventArgs e) {
    toggleButton(3, btnMine);
}
```

```

/// <summary>
/// When a button is clicked, the button is toggled on, or if it is already on,
/// toggle it off
/// </summary>
/// <param name="arraySlot">The arrayslot of the btnToggled array</param>
/// <param name="buttonName">The name of the button being toggled so that the
/// colour can be changed</param>
void toggleButton(int arraySlot, Button buttonName) {
    if (btnToggled[arraySlot]) {
        btnToggled[arraySlot] = false;
        buttonName.BackColor = SystemColors.ControlLight;
    } else {
        btnToggled[arraySlot] = true;
        buttonName.BackColor = Color.Chocolate;
        checkOtherButtons(arraySlot);
    }
}

/// <summary>
/// When a button is toggled on, make sure all other buttons are toggled off
/// </summary>
/// <param name="arraySlot">The arrayslot of the btnToggled array</param>
void checkOtherButtons(int arraySlot) {
    if (btnToggled[arraySlot]) {
        for (int i = 0; i < btnToggled.Length; i++) {
            if (!btnToggled[i].Equals(btnToggled[arraySlot])) {
                btnToggled[i] = true;
            }
        }
    }
}

```

Input	Expected Output	Actual Output
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)	It takes 2 clicks to turn toggle the button on (and go brown) once the program is first started, and then after that it only takes 1 click to toggle either way
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by 1/sec, and the colour goes back to grey (button is toggled off)	Same as above, except that it takes 1 click to toggle off once it is already on.
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and food rate is increased by 1/sec (button is toggled on)	Takes 2 clicks to turn on wood button, button goes brown, but when food button is clicked, it turns brown and the wood button also turns brown.

After some debugging using the breakpoint and message boxes, I have realised that the reason none of this works is because:

1. In the checkOtherButtons procedure, `btnToggled[i] = true`, when it should be set to false. This is because the line is meant to toggle *off* all other buttons instead of on
2. In the line below, there should be a way to change all of these buttons' colours back to grey as well, but that would require a button object array, which begins to get too complex. Therefore, my solution to this is to go back to my original pseudocode and change the buttons' colours as part of the resourceCollection procedure

Here is the fixed code:

```

/// <summary>
/// When a button is clicked, the button is toggled on, or if it is already on,
/// toggle it off
/// </summary>
/// <param name="arraySlot">The arrayslot of the btnToggled array</param>
void toggleButton(int arraySlot) {
    if (btnToggled[arraySlot]) {
        btnToggled[arraySlot] = false;
    } else {
        btnToggled[arraySlot] = true;
        checkOtherButtons(arraySlot);
    }
}

/// <summary>
/// When a button is toggled on, make sure all other buttons are toggled off
/// </summary>
/// <param name="arraySlot">The arrayslot of the btnToggled array</param>
void checkOtherButtons(int arraySlot) {
    if (btnToggled[arraySlot]) {
        for (int i = 0; i < btnToggled.Length; i++) {
            if (!btnToggled[i].Equals(btnToggled[arraySlot])) {
                btnToggled[i] = false;
            }
        }
    }
}

```

Input	Expected Output	Actual Output
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)	The button is toggled on
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by 1/sec, and the colour goes back to grey (button is toggled off)	The button is toggled on, then off in second click
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and	The wood button is toggled on, then off when the food button is clicked. The food button is also toggled on

	food rate is increased by 1/sec (button is toggled on)	
--	---	--

I will redo these tests once the changeButtonColour function is in the game, so for now the buttons will not change colour.

ResourceCollection

I have written the resourceCollection code very closely to the pseudocode, however I have noticed that;

1. I do not need to call reference to the timerType because I do not need to change the timer interval every tick
2. In the line labelAmounts.TEXT = globalData.resourcesData[0][arraySlot]; I have replaced the globalData part with currentAmount since I have that value stored locally anyway

Here is the code for it:

```
/// <summary>
/// Calculates how much each resource will be increased by each tick
/// </summary>
/// <param name="arraySlot">The resource type</param>
/// <param name="buttonType">The button type</param>
/// <param name="labelAmounts">The resource type's labelAmount</param>
/// <param name="labelRates">The resource type's labelRate</param>
/// <param name="pBarType">The progress bar type</param>
void resourceCollection(int arraySlot, Button buttonType, Label labelAmounts, Label labelRates, ProgressBar pBarType) {
    int currentRate = GlobalData.resourcesData[1][arraySlot];
    int currentAmount = GlobalData.resourcesData[0][arraySlot];
```

```
// If button is toggled on
if (btnToggled[arraySlot]) {
    // Set its colour to grey
    buttonType.BackColor = SystemColors.ControlLight;
} else {
    // Set its colour to brown, and increase the currentRate by
gatherMultiplier
    currentRate += GlobalData.resourcesData[3][arraySlot];
    buttonType.BackColor = Color.Chocolate;
}

// If the currentAmount = 0, set it to 1 just in case it is divided by 0
(which would crash program)
amountCheck();

// Set the text of label rates
labelRates.Text = ("+" + currentRate + "/sec");

// If the current value of the progress bar is less than or equal to the
next value of the progress bar
if (pBarType.Value <= (pBarType.Maximum - currentRate)) {
    // Add the amount of resources to resourcesData for that tick
    GlobalData.resourcesData[0][arraySlot] += currentRate;
    // Set the text of label amount
    labelAmounts.Text = currentAmount.ToString();
    // Set the progress bar maximum to the capacity of the resource
    pBarType.Maximum = GlobalData.resourcesData[2][arraySlot];

    // If the next value of the progress bar will exceed the maximum
    if ((pBarType.Value + currentRate) > pBarType.Maximum) {
        // Set the maximum to be the value
        pBarType.Value = pBarType.Maximum;
    } else {
        // Add the current rate to the progress bar value
        pBarType.Value += currentRate;
    }
}
```

And here is the code for the amountCheck() procedure:

```
    /// <summary>
    /// Makes sure the amount of a resource NEVER goes below 1
    /// </summary>
    void amountCheck() {
        for (int i = 0; i < 4; i++) {
            if (GlobalData.resourcesData[i][0] < 1) {
                GlobalData.resourcesData[i][0] = 1;
            }
        }
    }
}
```

Of course, to be able to run this, I need to implement the globalTimer(). Here is the code for that:

```
void GlobalTimerTick(object sender, EventArgs e) {
    GlobalData.tickCounter += 1;
    resourceCollection(0, btnGather, lblWoodAmount, lblWoodRate, pbWood);
    resourceCollection(1, btnQuarrier, lblStoneAmount, lblStoneRate, pbStone);
    resourceCollection(2, btnFarm, lblFoodAmount, lblFoodRate, pbFood);
    resourceCollection(3, btnMine, lblMetalAmount, lblMetalRate, pbMetal);

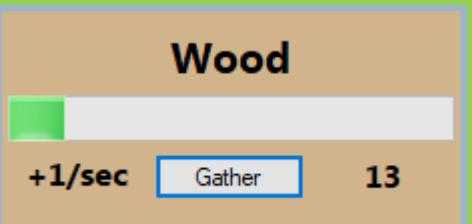
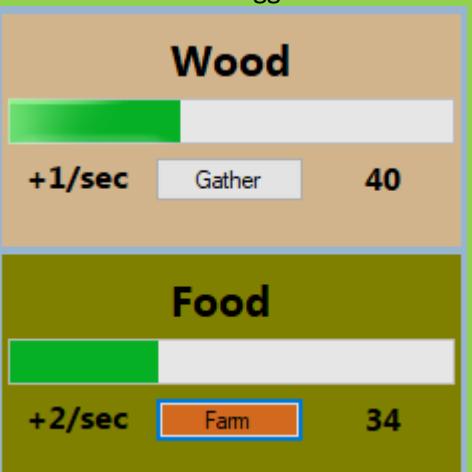
    housing();
}
```

The housing() procedure is there but there is nothing in there for now, but I will get to that later.

Immediately after running the program, I have noticed that all the buttons are toggled on by default. Since this change has happened after I added the resource collection and timer code, I can assume the issue is to do with that. It turned out that in the changing button colour part of the code, I just needed to set the if condition to if btnToggled[arraySlot] is false rather than true.

I have also discovered another issue, not that I have the button colours inside the resourceCollection function. The issue is that when I click on a button, and then other one, the original button does not get toggled off. This shows that the changeBtnColour function no longer works. Here is my fix for it:

```
/// <summary>
/// When a button is toggled on, make sure all other buttons are toggled off
/// </summary>
/// <param name="arraySlot">The arrayslot of the btnToggled array</param>
void checkOtherButtons(int arraySlot) {
    //Check if the button being clicked is being toggled on
    if (btnToggled[arraySlot]) {
        //Loop through all the buttons
        for (int i = 0; i < btnToggled.Length; i++) {
            // Set all the buttons to false
            btnToggled[i] = false;
        }
        // Then set currently clicked button to true
        btnToggled[arraySlot] = true;
    }
}
```

Input	Expected Output	Actual Output
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)	<p>The button is toggled on</p> 
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by 1/sec, and the colour goes back to grey (button is toggled off)	<p>The button is toggled on, then off in second click</p> 
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and food rate is increased by 1/sec (button is toggled on)	<p>The wood button is toggled on, then off when the food button is clicked. The food button is also toggled on</p> 
The wood progress bar reaches maximum	The wood value no longer increases and the progress bar stays filled.	<p>(Here the max is 100) The bars only fill up to the maximum – currentRate, because of one of the overflow checks</p> 

In order to fix this, I will need to change the resourceCollection function to have the correct conditions of when the bar is filled up or not. The following code now fixes this issue:

```

void resourceCollection(int arraySlot, Button buttonType, Label labelAmounts, Label
labelRates, ProgressBar pBarType) {
    int currentRate = GlobalData.resourcesData[1][arraySlot];
    int currentAmount = GlobalData.resourcesData[0][arraySlot];
    int currentCapacity = GlobalData.resourcesData[2][arraySlot];

    // If button is toggled on
    if (!btnToggled[arraySlot]) {
        // Set its colour to grey
        buttonType.BackColor = SystemColors.ControlLight;
    } else {
        // Set its colour to brown, and increase the currentRate by
        gatherMultiplier
        currentRate += GlobalData.resourcesData[3][arraySlot];
        buttonType.BackColor = Color.Chocolate;
    }

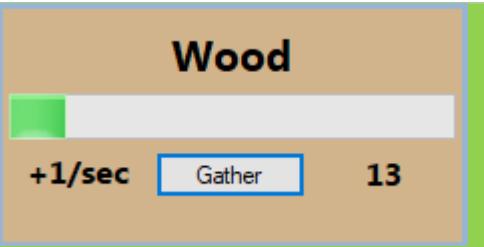
    // If the currentAmount = 0, set it to 1 just in case it is divided by 0
    (which would crash program)
    amountCheck();

    // Set the text of label rates
    labelRates.Text = ("+" + currentRate + "/sec");

    // Set the progress bar maximum to the capacity of the resource
    pBarType.Maximum = currentCapacity;
    // If there is space to increase the value of the bar
    if (currentAmount <= (currentCapacity - currentRate)) {
        currentAmount += currentRate;
    } else {
        currentAmount = currentCapacity;
    }
    // Update the progress bar
    pBarType.Value = currentAmount;
    // Set the new current amount to show in the output label
    GlobalData.resourcesData[0][arraySlot] = currentAmount;
    labelAmounts.Text = currentAmount.ToString();
}

```

Input	Expected Output	Actual Output
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)	<p>The button is toggled on</p> 
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by	The button is toggled on, then off in second click

	1/sec, and the colour goes back to grey (button is toggled off)	
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and food rate is increased by 1/sec (button is toggled on)	The wood button is toggled on, then off when the food button is clicked. The food button is also toggled on 
The wood progress bar reaches maximum	The wood value no longer increases and the progress bar stays filled.	

Finally, I need to code the quick housing() function, although I cannot yet test it until I have properly implemented the upgrades section. Even though I could just manually change the globalData variables before I start the program, I need to test whether or not it updates and adapts correctly to a housing upgrade that has just been bought. Therefore, I will do any housing tests and fixes in my upgrades section, but here is the code for it now:

```

void housing() {
    // Make locally scoped variables for easier access
    int currentTotalHousing = GlobalData.totalHousing;
    int housingRemaining = GlobalData.housingRemaining;

    // Set the current housing amount to the amount in global data
    lblHousingAmount.Text = currentTotalHousing.ToString();
    // Set the value and maximum of the housing progress bar
    pbHousing.Value = currentTotalHousing;
    pbHousing.Maximum = housingRemaining - currentTotalHousing;
}

```

V2 17/09/19

I will therefore have to add code for the science visualising, but this will be very similar to the housing procedure:

```

void science() {
    // Make locally scoped variables for easier access
    int currentScience = GlobalData.scienceData;

    // Set the current science amount in the global data
    lblScienceAmount.Text = currentScience.ToString();
}

```

As you can see from the code, there is no progress bar handling. This is because as I was coding it, I realised that there is no cap on the science, thus nothing for the bar to work towards. So now that I've removed the bar, there is just the amount label left over.



Menu Strip [E]

V1 12/09/19

Before I get into actually creating the upgrades GUI, I need to make the menu strip function, so that when the player clicks on the upgrades button, it displays the upgrades sections and hides the resource collection sections. This was very easy to code:

```
Panel[] panelSections = new Panel[5];

// Set each section panel to an element of the panel array
panelSections[0] = pnlResourceCollection;
panelSections[1] = pnlUpgrades;
panelSections[2] = pnlCombat;
panelSections[3] = pnlLogs;
panelSections[4] = pnlFileHandling;

// Hide every single panel except the resource collection panel (on program startup)
for (int i = 0; i < panelSections.Length; i++) {
    // If the current panel is the resource collection one, show it
    if (panelSections[i] == pnlResourceCollection) {
        panelSections[0].Show();
    } else {
        panelSections[i].Hide();
    }
}

void MsMainItemClicked(object sender, ToolStripItemClickedEventArgs e) {
    // When a button on the menu strip is clicked, hide all panels
    for (int i = 0; i < panelSections.Length; i++) {
        panelSections[i].Hide();
    }
}

private void resourceCollectionToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the resource collection panel is clicked, show it
    pnlResourceCollection.Show();
}

private void upgradesToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the upgrades panel is clicked, show it
    pnlUpgrades.Show();
}

private void combatToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the combat panel is clicked, show it
    pnlCombat.Show();
}

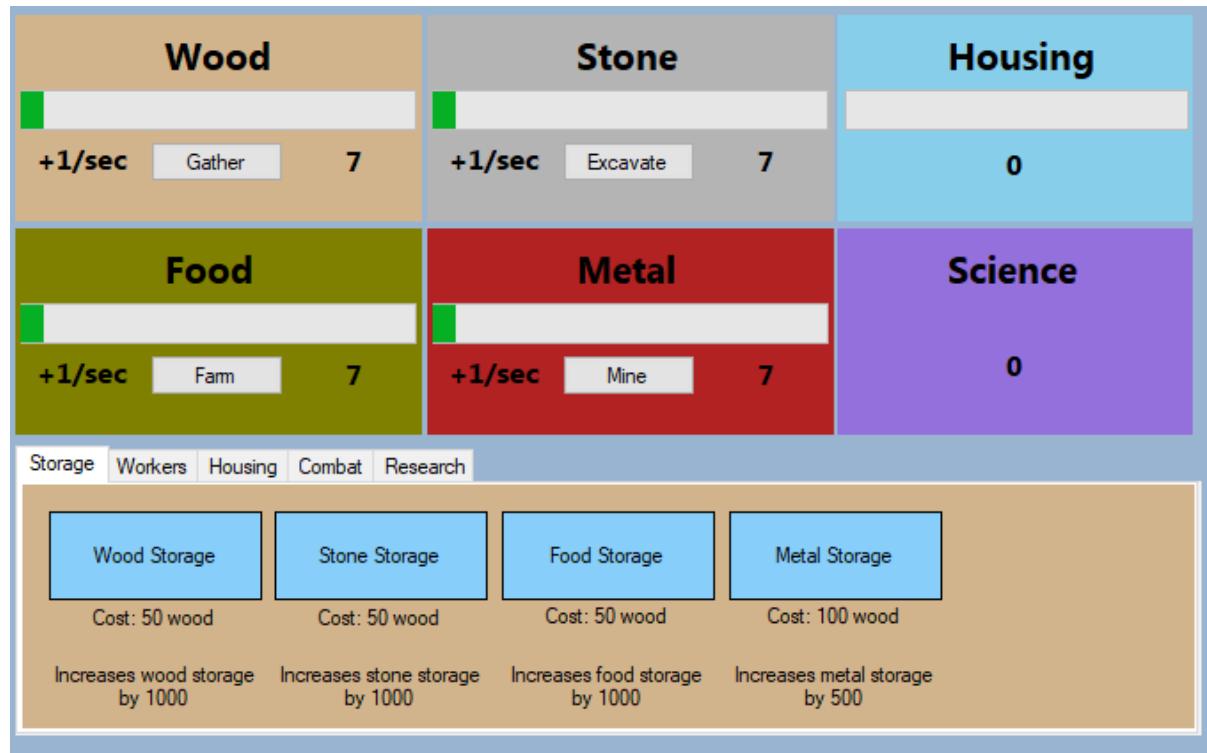
private void logsToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the logs panel is clicked, show it
    pnlLogs.Show();
}

private void saveLoadToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the file handling panel is clicked, show it
    pnlFileHandling.Show();
}
```

Input (or question)	Expected Output	Actual Output
The resource collection button on the menu strip is clicked	The resources menu is shown with each part for wood, food, stone, metal and housing	The resources menu is shown with each part for wood, food, stone, metal and housing
The upgrades button on the menu strip is clicked	The upgrades menu is shown with each tab for workers, storage, research, housing and combat	The upgrades menu is shown with each tab for workers, storage, research, housing and combat

V2 21/10/19

When I was about to test this storage code, one of my stakeholders pointed out that it's annoying to have to keep switching between the upgrades and the resources tabs in order to see which upgrades the player wants to buy.



I accomplished this by simply removing the pnlUpgrades, and replacing it with the extended pnlResourceCollection to cover the upgrades part below it. Now, I will remove the menu strip button called "Upgrades", and rename the "Resource Collection" part to "Resource Collection and Upgrades".

Resource Collection and Upgrades Combat Logs Save/Load

My stakeholders are now happy about this.

[name]

Candidate No: xxxx

Centre No:xxxxx

V3 02/11/19

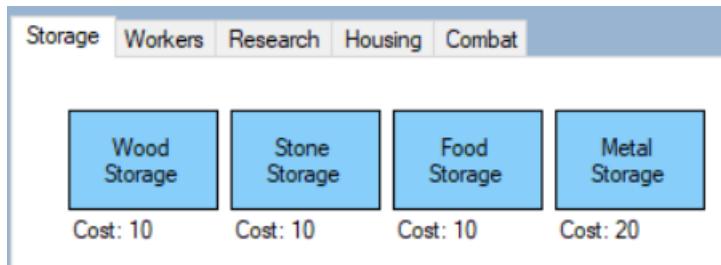
Whilst working on the combat section, I realised that compared with the rest of the form, the menu strip text was way too small, so I made it slightly larger:

Resource Collection and Upgrades Combat Logs Save/Load

Upgrades GUI [A]

V1 15/09/19

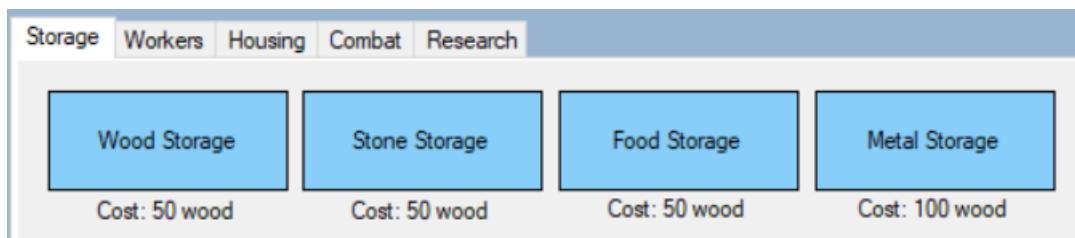
First, I will start by making the GUI based directly off my designs.



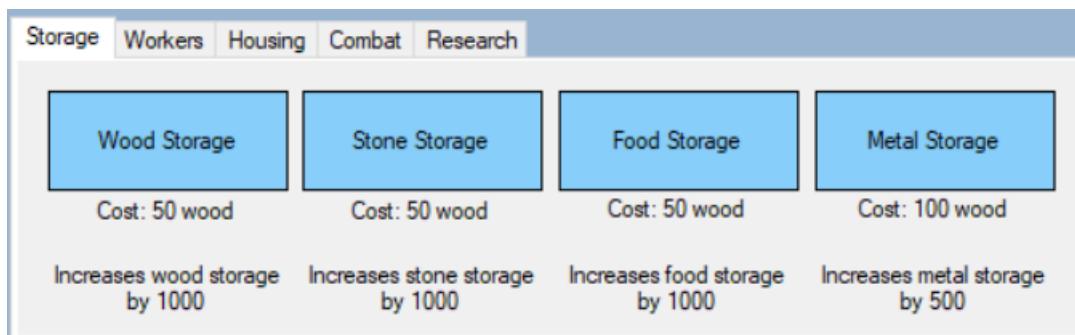
After talking to my stakeholders, we have realised that:

1. The costs labels do not show which resource they actually cost
2. Once the upgrades have been bought multiple times, the costs will get very large thus the labels will need to be large enough to accommodate this
3. If the labels need to be large, the buttons will need to also be wider to be the same length of the labels

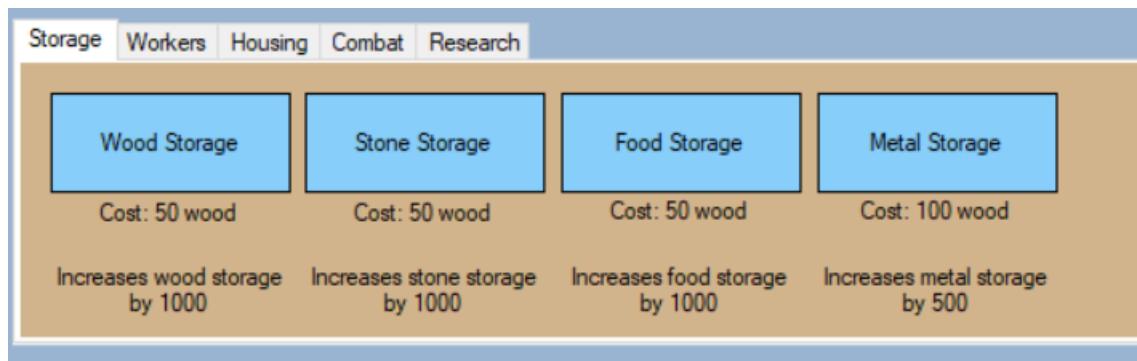
Therefore, I came up with this solution



They were happy about this, until Alex realised that there are no indications anywhere telling the player what the upgrade is for. Therefore, I added these labels

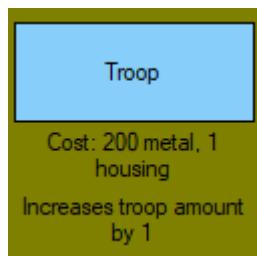


My stakeholders had one last suggestion, but purely aesthetical – changing the back colours of each upgrade to their respective costs e.g. storage would be tan coloured, as the cost is wood.



V2 24/10/19

I have noticed that when working on the workers upgrade section, the label for the troop cost is very slightly wrong, in that it says it costs 400 food when it actually costs 200 metal. Change:



Upgrades Code [A]

Storage

V1 22/10/19

The tab controls in C#.NET are automatically created with their code to switch between each tab part and show + hide the controls in each form automatically.

First, I will start off by making the storage code, as it is first in line and only links to the resource collection section.

```
private void btnWoodStorage_Click(object sender, EventArgs e) {
    buyingStorage(GlobalData.upgradesCosts[0][0], 0, 0, 1000,
lblWoodStorageCost);
}

private void btnStoneStorage_Click(object sender, EventArgs e) {
    buyingStorage(GlobalData.upgradesCosts[0][1], 0, 1, 1000,
lblStoneStorageCost);
}

private void btnFoodStorage_Click(object sender, EventArgs e) {
    buyingStorage(GlobalData.upgradesCosts[0][2], 0, 2, 1000,
lblFoodStorageCost);
}

private void btnMetalStorage_Click(object sender, EventArgs e) {
    buyingStorage(GlobalData.upgradesCosts[0][3], 0, 3, 1000,
lblMetalStorageCost);
}
```

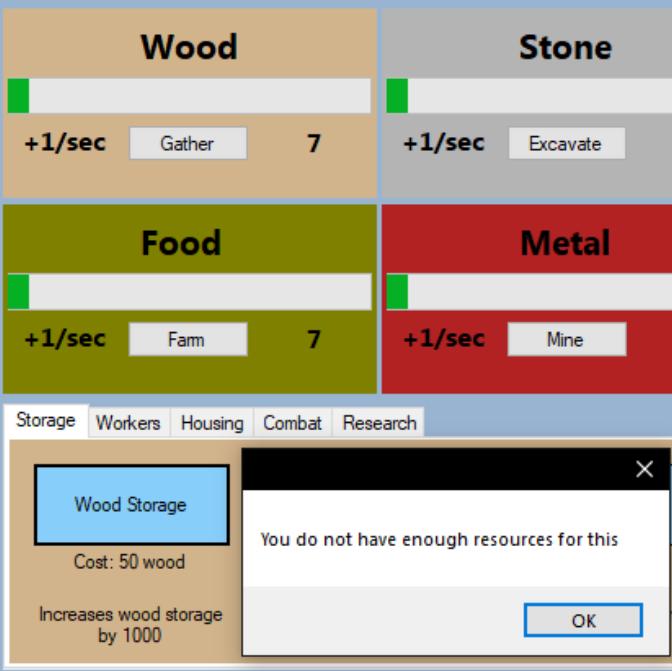
Next is the actual buyingStorage procedure:

```

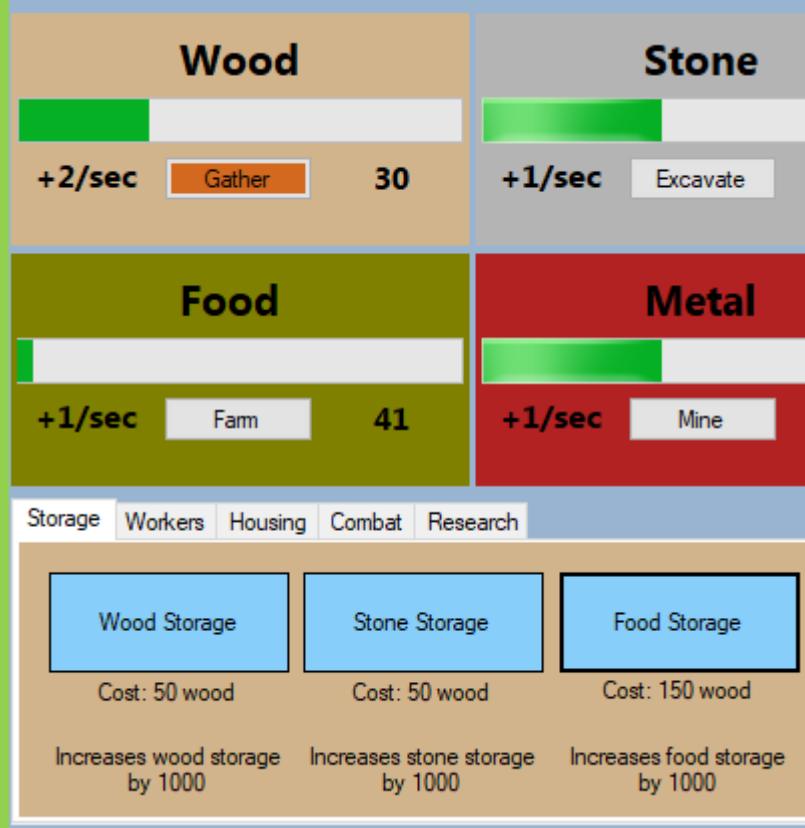
/// <param name="cost">The cost of the storage</param>
/// <param name="typeToBuy">What resource is needed to buy it</param>
/// <param name="typeToBuyFor">What resource storage it is being bought for</param>
/// <param name="storageIncrease">The amount of storage the upgrade increased
//by</param>
void buyingStorage(int cost, int typeToBuy, int typeToBuyFor, int storageIncrease,
Label labelCosts) {
    // If the player does not have enough wood
    if (GlobalData.resourcesData[0][typeToBuy] < cost) {
        // Print to upgrades tab in logs
    // If they do
    } else {
        // Deduct the cost from the number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase the cost of this upgrade
        GlobalData.upgradesCosts[0][typeToBuy] *=
GlobalData.costMultipliers[0];
        // Increase the storage amount by the amount specified
        GlobalData.resourcesData[2][typeToBuyFor] += storageIncrease;
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[0][typeToBuy]
+ " wood");
    }
}

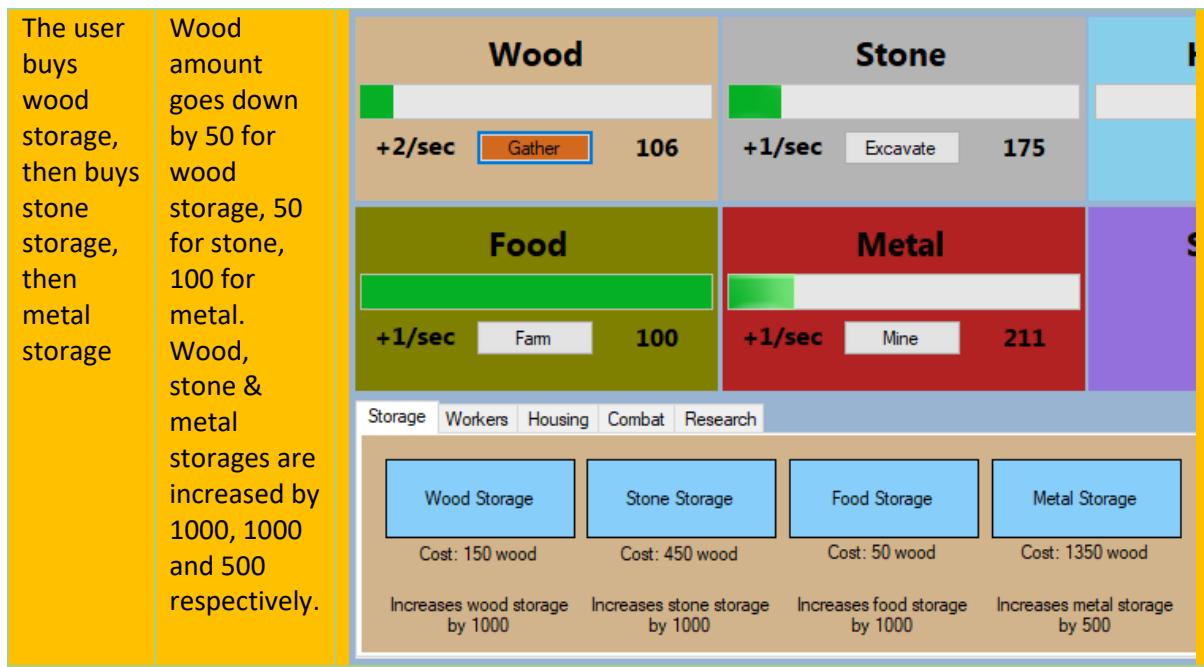
```

As I have not created the logs part of the program, I will just output the “not enough resources” parts in a message box temporarily for easier debugging.

Input	Expected Output	Actual Output
There are not enough resources to buy wood storage	Print “You do not have enough resources for this” in upgrades logs, upgrade is not bought	

It costs 50 wood to buy wood storage, user has exactly 50 wood, buys storage	Wood amount goes down to 0, the upgrade goes through, the storage is increased by 1000, storage cost increased by 3x	<p>Wood</p> <p>+1/sec Gather 6</p> <p>Food</p> <p>+1/sec Farm 40</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Stone Storage</p> <p>Cost: 150 wood Cost: 50 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000</p>
It costs 50 wood to buy wood storage, user has reached full capacity of the wood storage, buys storage	Wood amount goes down by 50, the upgrade goes through, the storage is increased by 1000, so progress bar is pushed further back, storage cost increased by 3x	<p>Wood</p> <p>+2/sec Gather 52</p> <p>Food</p> <p>+1/sec Farm 57</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Stone Storage</p> <p>Cost: 150 wood Cost: 50 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000</p>
It costs 900 wood to buy metal	Wood amount goes down by 900, the upgrade	Wood amount goes down by 900, the upgrade goes through, the metal storage is increased by 500, so progress bar of both wood and meal is pushed further back, storage cost increased by 3x (difficult to get screenshot)

	<p>storage, user has reached full metal capacity, buys storage</p> <p>goes through, the metal storage is increased by 500, so progress bar of both wood and meal is pushed further back, storage cost increased by 3x</p>	
<p>Food storage upgrade is bought</p>	<p>The maximum of the wood stays the same, maximum of the food increases by 1000</p>	 <p>Wood</p> <p>+2/sec Gather 30</p> <p>Stone</p> <p>+1/sec Excavate</p> <p>Food</p> <p>+1/sec Farm 41</p> <p>Metal</p> <p>+1/sec Mine</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Stone Storage Food Storage</p> <p>Cost: 50 wood Cost: 50 wood Cost: 150 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000 Increases food storage by 1000</p>



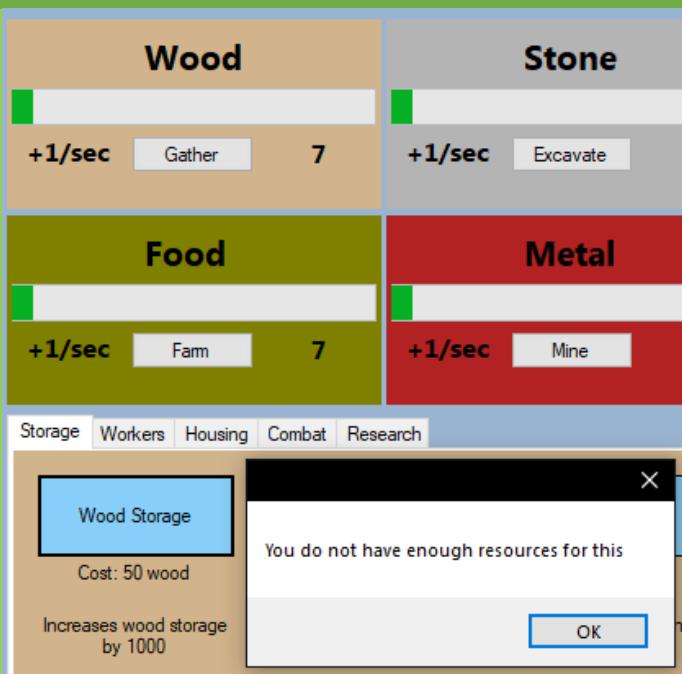
The issue with this last test is that even though the first of each upgrade is the base of 50, they are multiplied by the previous one, rather than itself. E.g. the wood storage costs 50, and the stone storage also costs 50 on first buys. However, the next price of the stone storage is 3 times that of the wood storage which is incorrect – both should go up to 50. From this, I have discovered what is wrong; the line:

```
GlobalData.upgradesCosts[0][typeToBuy] *= GlobalData.costMultipliers[0];
```

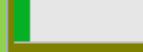
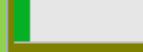
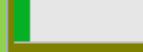
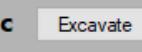
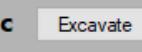
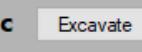
Should actually be:

```
GlobalData.upgradesCosts[0][typeToBuyFor] *= GlobalData.costMultipliers[0];
```

Because it needs to increase the cost of the resource type that the storage is being bought for, not just from the wood type.

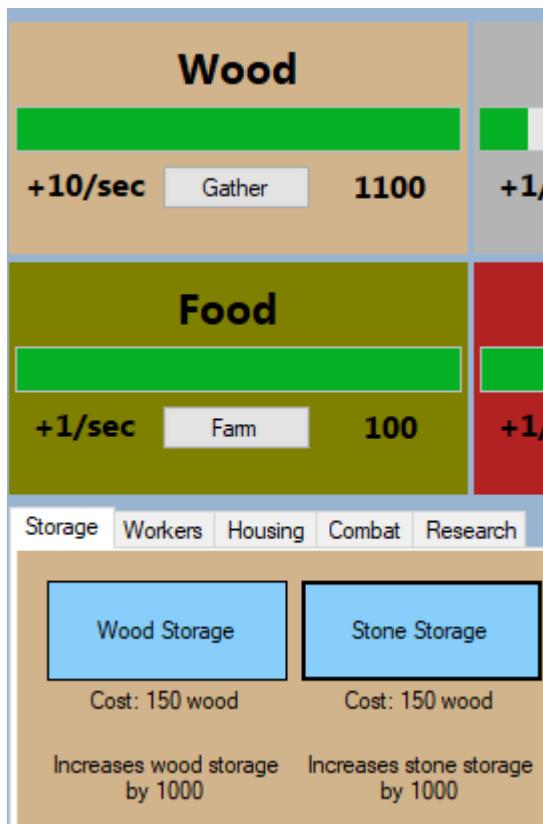
Input	Expected Output	Actual Output
There are not enough resources to buy wood storage	Print "You do not have enough resources for this" in upgrades logs, upgrade is not bought	
It costs 50 wood to buy wood storage, user has exactly 50 wood, buys storage	Wood amount goes down to 0, the upgrade goes through, the storage is increased by 1000, storage cost increased by 3x	

It costs 50 wood to buy wood storage, user has reached full capacity of the wood storage, buys storage	Wood amount goes down by 50, the upgrade goes through, the storage is increased by 1000, so progress bar is pushed further back, storage cost increased by 3x	 <p>Wood</p> <p>+2/sec Gather 52</p> <p>Food</p> <p>+1/sec Farm 57</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Cost: 150 wood</p> <p>Stone Storage Cost: 50 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000</p>	
It costs 900 wood to buy metal storage, user has reached full metal capacity, buys storage	Wood amount goes down by 900, the upgrade goes through, the metal storage is increased by 500, so progress bar of both wood and meal is pushed further back, storage cost increased by 3x (difficult to get screenshot)		

	<p>Food storage upgrade is bought</p> <p>The maximum of the wood stays the same, maximum of the food increases by 1000</p>	<table border="1"> <thead> <tr> <th colspan="2">Wood</th><th colspan="2">Stone</th></tr> </thead> <tbody> <tr> <td></td><td></td><td></td><td></td></tr> <tr> <td>+2/sec</td><td>Gather</td><td>30</td><td>+1/sec</td><td>Excavate</td></tr> <tr> <th colspan="2">Food</th><th colspan="2">Metal</th></tr> <tr> <td></td><td></td><td></td><td></td></tr> <tr> <td>+1/sec</td><td>Farm</td><td>41</td><td>+1/sec</td><td>Mine</td></tr> <tr> <td colspan="4"> Storage Workers Housing Combat Research </td></tr> <tr> <td colspan="2">Wood Storage</td><td>Stone Storage</td><td>Food Storage</td></tr> <tr> <td colspan="2">Cost: 50 wood</td><td>Cost: 50 wood</td><td>Cost: 150 wood</td></tr> <tr> <td colspan="2" rowspan="2">Increases wood storage by 1000</td><td>Increases stone storage by 1000</td><td>Increases food storage by 1000</td></tr> </tbody> </table>	Wood		Stone						+2/sec	Gather	30	+1/sec	Excavate	Food		Metal						+1/sec	Farm	41	+1/sec	Mine	Storage Workers Housing Combat Research				Wood Storage		Stone Storage	Food Storage	Cost: 50 wood		Cost: 50 wood	Cost: 150 wood	Increases wood storage by 1000		Increases stone storage by 1000	Increases food storage by 1000										
Wood		Stone																																																				
																																																						
+2/sec	Gather	30	+1/sec	Excavate																																																		
Food		Metal																																																				
																																																						
+1/sec	Farm	41	+1/sec	Mine																																																		
Storage Workers Housing Combat Research																																																						
Wood Storage		Stone Storage	Food Storage																																																			
Cost: 50 wood		Cost: 50 wood	Cost: 150 wood																																																			
Increases wood storage by 1000		Increases stone storage by 1000	Increases food storage by 1000																																																			
		<table border="1"> <thead> <tr> <th colspan="2">Wood</th> <th colspan="2">Stone</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>+10/sec</td> <td>Gather</td> <td>30</td> <td>+1/sec</td> <td>Excavate</td> </tr> <tr> <th colspan="2">Food</th> <th colspan="2">Metal</th> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>+1/sec</td> <td>Farm</td> <td>38</td> <td>+1/sec</td> <td>Mine</td> </tr> <tr> <td colspan="4"> Storage Workers Housing Combat Research </td></tr> <tr> <td colspan="2">Wood Storage</td><td>Stone Storage</td><td>Food Storage</td></tr> <tr> <td colspan="2">Cost: 150 wood</td><td>Cost: 150 wood</td><td>Cost: 50 wood</td></tr> <tr> <td colspan="2">Increases wood storage by 1000</td><td>Increases stone storage by 1000</td><td>Increases food storage by 1000</td></tr> <tr> <td colspan="2"></td><td colspan="2">Metal Storage</td></tr> <tr> <td colspan="2">Cost: 150 wood</td><td colspan="2">Increases metal storage by 500</td></tr> <tr> <td colspan="2">Increases wood storage by 1000</td><td colspan="2">Increases stone storage by 1000</td></tr> </tbody> </table>	Wood		Stone						+10/sec	Gather	30	+1/sec	Excavate	Food		Metal						+1/sec	Farm	38	+1/sec	Mine	Storage Workers Housing Combat Research				Wood Storage		Stone Storage	Food Storage	Cost: 150 wood		Cost: 150 wood	Cost: 50 wood	Increases wood storage by 1000		Increases stone storage by 1000	Increases food storage by 1000			Metal Storage		Cost: 150 wood		Increases metal storage by 500		Increases wood storage by 1000	
Wood		Stone																																																				
																																																						
+10/sec	Gather	30	+1/sec	Excavate																																																		
Food		Metal																																																				
																																																						
+1/sec	Farm	38	+1/sec	Mine																																																		
Storage Workers Housing Combat Research																																																						
Wood Storage		Stone Storage	Food Storage																																																			
Cost: 150 wood		Cost: 150 wood	Cost: 50 wood																																																			
Increases wood storage by 1000		Increases stone storage by 1000	Increases food storage by 1000																																																			
		Metal Storage																																																				
Cost: 150 wood		Increases metal storage by 500																																																				
Increases wood storage by 1000		Increases stone storage by 1000																																																				

Note: the wood rate has been manually increased to +10/sec so that it is faster to reach the number of resources required to buy the upgrade. It does not effect the tests.

The final test still does not work properly – here I have clicked the stone storage button twice, yet the labelled cost does not increase to 450 (as that is $50 * 3 * 3$). What I have noticed though is that even though the label of the cost is stuck at 150, the internals still increase the costs correctly. Here if I click the stone storage button it says I do not have enough resources, even though according to the cost label I do (it should cost $50 * 3 * 3 * 3$ or 1350, as I have clicked it 3 times):



This is also a very simple fix, and also the same issue as before; the line:

```
labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[0][typeToBuy] + " wood");
```

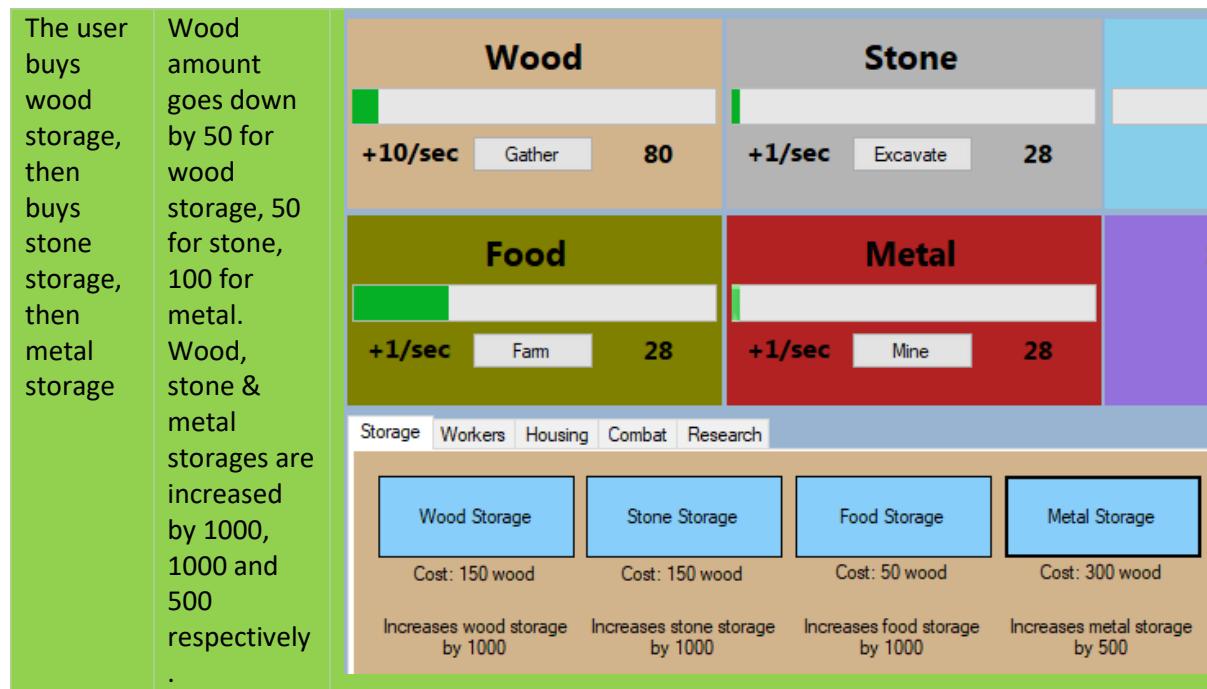
Should be:

```
labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[0][typeToBuyFor] + " wood");
```

Input	Expected Output	Actual Output
There are not enough resources to buy wood storage	Print "You do not have enough resources for this" in upgrades logs, upgrade is not bought	

<p>It costs 50 wood to buy wood storage, user has exactly 50 wood, buys storage</p>	<p>Wood amount goes down to 0, the upgrade goes through, the storage is increased by 1000, storage cost increased by 3x</p> 	
<p>It costs 50 wood to buy wood storage, user has reached full capacity of the wood storage, buys storage</p>	<p>Wood amount goes down by 50, the upgrade goes through, the storage is increased by 1000, so progress bar is pushed further back, storage cost increased by 3x</p> 	
<p>It costs 900 wood to buy metal</p>	<p>Wood amount goes down by 900, the upgrade goes through, the metal storage is increased by 500, so progress bar of both wood and meal is pushed further back, storage cost increased by 3x (difficult to get screenshot)</p>	

storage, user has reached full metal capacity, buys storage	goes through, the metal storage is increased by 500, so progress bar of both wood and meal is pushed further back, storage cost increased by 3x	
Food storage upgrade is bought	The maximum of the wood stays the same, maximum of the food increases by 1000	<p>Wood</p> <p>+2/sec Gather 30</p> <p>Stone</p> <p>+1/sec Excavate</p> <p>Food</p> <p>+1/sec Farm 41</p> <p>Metal</p> <p>+1/sec Mine</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Cost: 50 wood Increases wood storage by 1000</p> <p>Stone Storage Cost: 50 wood Increases stone storage by 1000</p> <p>Food Storage Cost: 150 wood Increases food storage by 1000</p>



My stakeholders all like this.

Housing

[V1 23/10/19](#)

Now that the storage works properly, I will work on the housing next. This is because the workers section requires a working housing system in order to run (without manual edits to the variables).

The buttons code:

```

private void btnShack_Click(object sender, EventArgs e) {
    buyingHousing(GlobalData.upgradesCosts[3][0], 1, 0, lblShackCost);
}

private void btnBootcamp_Click(object sender, EventArgs e) {
    buyingHousing(GlobalData.upgradesCosts[3][1], 1, 1, lblBootcampCost);
}

private void btnBarracks_Click(object sender, EventArgs e) {
    buyingHousing(GlobalData.upgradesCosts[3][2], 1, 2, lblBarracksCost);
}

```

```

void buyingHousing(int cost, int typeToBuy, int housingType, Label labelCosts) {
    // If there are enough resources to buy the upgrade
    if (GlobalData.resourcesData[0][typeToBuy] < cost) {
        // Print to upgrades tab in logs
        MessageBox.Show("You do not have enough resources for this");
        // TODO: Link buyingHousing to upgrades part of logs
    } else {
        // Deduct cost from number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase cost of this upgrade
        GlobalData.upgradesCosts[3][typeToBuy] *=
        GlobalData.costMultipliers[3];
        // Increase the total housing this upgrade provides
        GlobalData.totalHousing += GlobalData.housingData[housingType];
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[3][typeToBuy]
+ " stone");
    }
}

```

Again, I will 1. Set stone rate to +10/sec to buy the upgrades faster – this will not affect my tests 2. Use message boxes to relay not enough resources rather than use the non-existent logs temporarily until I code the logs section.

Input	Expected Output	Actual Output
The shack is bought for 50 stone	50 stone deducted, housing space increased by 5, cost increased by 2x	<p>Exception Unhandled</p> <p>System.ArgumentOutOfRangeException: 'Value of '5' is not valid for 'Value'. 'Value' should be between 'minimum' and 'maximum'. Parameter name: Value'</p>

I know I cannot do anymore tests until this (first crash!) is fixed. The issue was that in the housing() function in ResourceCollection.cs, the value of the progress bar was trying to be set before the maximum, leading to the value being out of range of the progress bar. Once that was fixed, I realised that the value of the progress bar was being set incorrectly – I was setting the value to equal the housing remaining minus the housing capacity, essentially setting the value to negative. The fix for this was to just remove the mathematical operation altogether leaving the value to just the housing remaining bit. I also renamed some of the variables to make it easier to read:

```

void housing() {
    // Make locally scoped variables for easier access
    int currentHousingCapacity = GlobalData.totalHousing;
    int currentHousingRemaining = GlobalData.housingRemaining;

    // Set the current housing amount to the amount in global data
    lblHousingCapacity.Text = currentHousingCapacity.ToString() + " housing
capacity";
    lblHousingSpace.Text = currentHousingRemaining.ToString() + " housing
remaining";
    // Set the value and maximum of the housing progress bar
    pbHousing.Maximum = currentHousingCapacity;
    pbHousing.Value = currentHousingRemaining;
}

```

Now I can do all of the tests.

Input	Expected Output	Actual Output
The shack is bought for 50 stone	50 stone deducted, housing space increased by 5, cost increased by 2x	
The bootcamp is bought for 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x	
The barracks is bought for 500 stone	500 stone deducted, housing space increased by 20, cost increased by 2x	<pre> 1 reference private void btnBarracks_Click(object sender, EventArgs e) { buyingHousing(GlobalData.upgradesCosts[3][2], 1, 2, lblBarracksCost); } #endregion #region Procedures /// <param name="cost" /// <param name="type" /// <param name="capacity" /// <param name="remaining" </pre> <p>Exception Unhandled System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'</p>
The user tries to buy shack upgrade, but there is not enough stone	Upgrade does not go through, print not enough resources in upgrades logs	
The user tries to buy shack upgrade, has enough stone, but	50 stone deducted, housing space increased by 5, cost increased by 2x, print not enough resources in	<pre> 1 reference private void btnBarracks_Click(object sender, EventArgs e) { buyingHousing(GlobalData.upgradesCosts[3][2], 1, 2, lblBarracksCost); } #endregion #region Procedures /// <param name="cost" /// <param name="type" /// <param name="capacity" /// <param name="remaining" </pre> <p>Exception Unhandled System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'</p>

then tries to buy barracks and there is not enough stone	upgrades logs when trying to buy barracks upgrade	
The user tries to buy shack upgrade, has enough stone, but then tries to buy shack again but there is not enough stone	50 stone deducted, housing space increased by 5, cost increased by 2x, print not enough resources in upgrades logs when trying to buy stone upgrade	
The user tries to buy shack upgrade, has enough stone, but then tries to buy shack again but there is not enough stone (more than 50 though)	50 stone deducted, housing space increased by 5, cost increased by 2x, print not enough resources in upgrades logs when trying to buy stone upgrade	
The user buys bootcamp 3 times from base price of 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x. 200 stone deducted, housing space increased by 10, cost increased by 2x. 400 stone deducted, housing space increased by 10, cost increased by 2x	

From these tests I can deduct 3 issues:

1. The code calling the buyingHousing in the btnBarracks button is incorrect – I'm assuming the issue is that I inputted the parameters incorrectly

2. The shack upgrade can be bought anytime there is more than 50 stone, so therefore the actual shack upgrade cost variable is not being updated even though the label is
3. All the upgrades costs are being based off the shack cost, not their own e.g. the barracks costs are multiplied by the shack cost not its own cost

I have discovered the issue to the first and third problems:

```
// Upgrades data
public static List<int[]> upgradesCosts = new List<int[]> {
    new int[4] {50, 50, 50, 100}, // Storage Costs
    new int[4] {100, 100, 100, 200}, // Workers Costs
    new int[1] {100}, // Science Costs
    new int[2] {50, 50}, // Housing Costs
    new int[4] {200, 500, 300, 200} // Combat Costs ([][0]Health, [][1]Block,
    [][2]Damage, [][3]NoTroops)
};
```

The line:

```
new int[2] {50, 50}, // Housing Costs
```

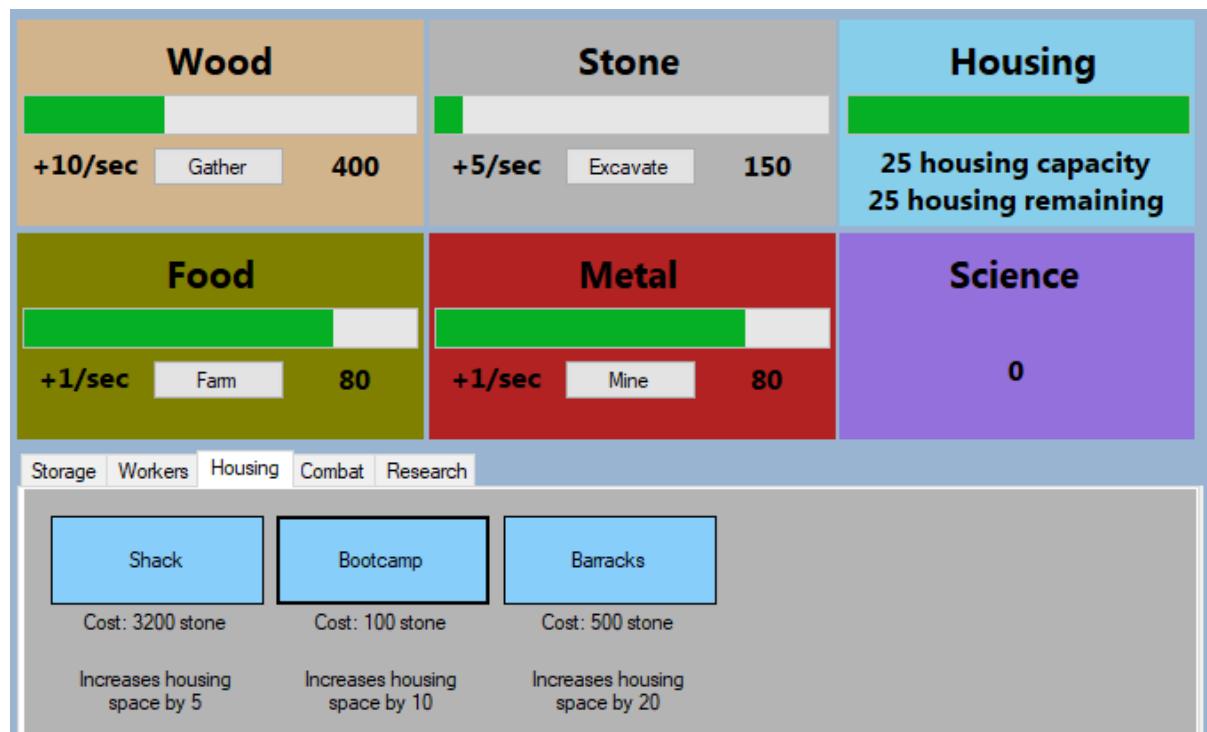
Is wrong because

1. There are not 3 elements for each housing type – explains why the buying barracks part gives an out of bounds crash
2. The cost for the second element – the bootcamp – is wrong; it should be 100 not 50

Now that I have edited the line to:

```
new int[3] {50, 100, 500}, // Housing Costs
```

I can work on the second point about the actual cost not being increased. When checking that the second point still doesn't work (needed to be sure) I took this screenshot to show how broken this is:



As you can see, as long as I have more than 50 stone for the shack, I can buy infinite upgrades. But when I try to buy the bootcamp upgrade (of I have enough stone here) I cannot, because it applied the cost of 3200 stone here. I must have gotten some code mixed up somewhere. After only a few seconds, I have realised what the issue is: I put the typeToBuy variable in some areas that the housingType variable should go in – so it was only increasing and outputting the cost of element 1, aka the bootcamp type. So I have replaced the lines:

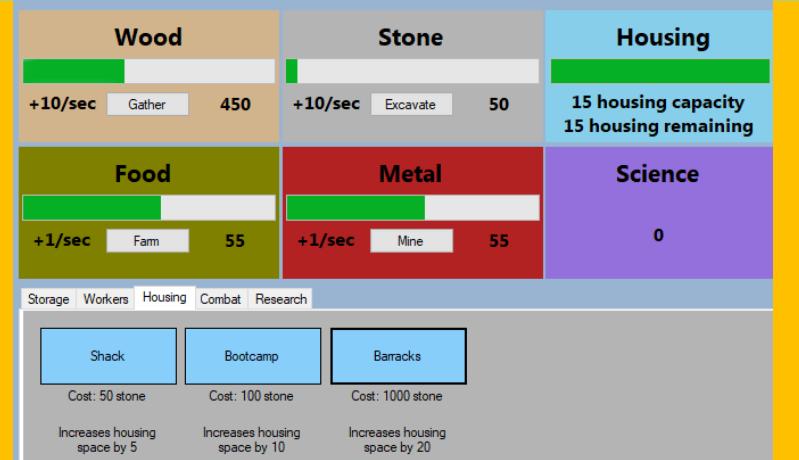
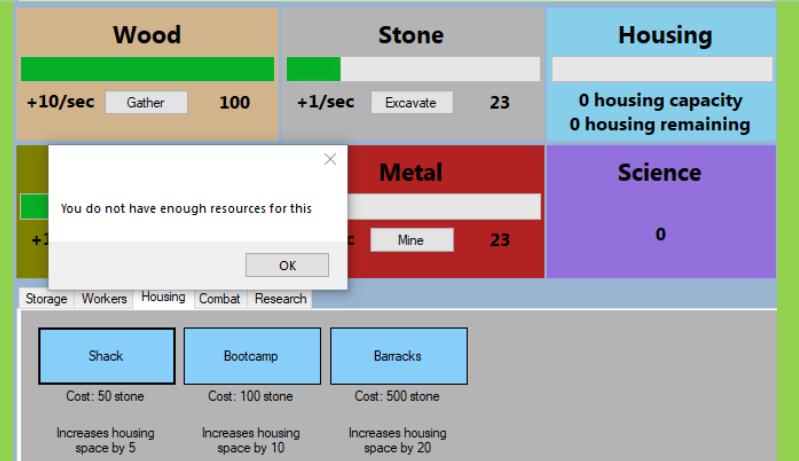
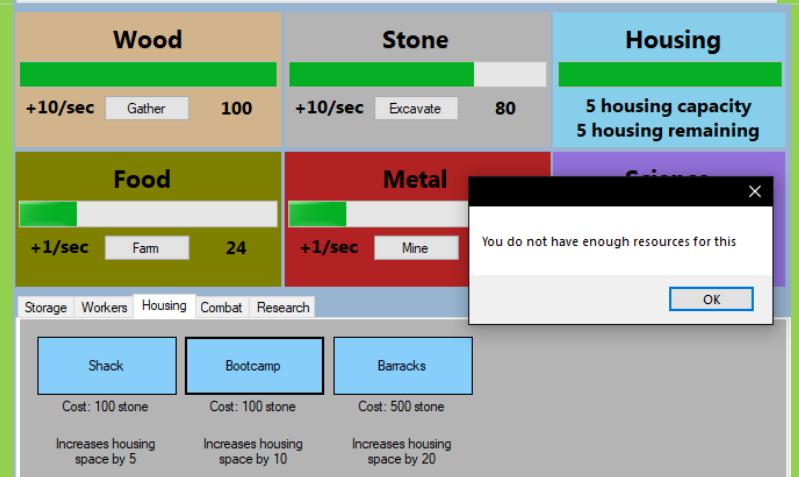
```
GlobalData.upgradesCosts[3][typeToBuy] *= GlobalData.costMultipliers[3];
labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[3][typeToBuy] + " stone");
```

With:

```
GlobalData.upgradesCosts[3][housingType] *= GlobalData.costMultipliers[3];
labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[3][housingType] + " stone");
```

Now that I think I have fixed all of the issues, I will redo all of the tests.

Input	Expected Output	Actual Output
The shack is bought for 50 stone	50 stone deducted, housing space increased by 5, cost increased by 2x	
The bootcamp is bought for 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x	

<p>The barracks is bought for 500 stone</p>	 <table border="1"> <thead> <tr> <th>Resource</th> <th>Value</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>Wood</td> <td>450</td> <td>+10/sec Gather</td> </tr> <tr> <td>Stone</td> <td>50</td> <td>+10/sec Excavate</td> </tr> <tr> <td>Housing</td> <td>15 housing capacity 15 housing remaining</td> <td></td> </tr> <tr> <td>Food</td> <td>55</td> <td>+1/sec Farm</td> </tr> <tr> <td>Metal</td> <td>55</td> <td>+1/sec Mine</td> </tr> <tr> <td>Science</td> <td>0</td> <td></td> </tr> </tbody> </table> <p>Storage Workers Housing Combat Research</p> <p>Shack Bootcamp Barracks</p> <p>Cost: 50 stone Cost: 100 stone Cost: 1000 stone</p> <p>Increases housing space by 5 Increases housing space by 10 Increases housing space by 20</p>	Resource	Value	Action	Wood	450	+10/sec Gather	Stone	50	+10/sec Excavate	Housing	15 housing capacity 15 housing remaining		Food	55	+1/sec Farm	Metal	55	+1/sec Mine	Science	0	
Resource	Value	Action																				
Wood	450	+10/sec Gather																				
Stone	50	+10/sec Excavate																				
Housing	15 housing capacity 15 housing remaining																					
Food	55	+1/sec Farm																				
Metal	55	+1/sec Mine																				
Science	0																					
<p>The user tries to buy shack upgrade, but there is not enough stone</p>	 <table border="1"> <thead> <tr> <th>Resource</th> <th>Value</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>Wood</td> <td>100</td> <td>+10/sec Gather</td> </tr> <tr> <td>Stone</td> <td>23</td> <td>+1/sec Excavate</td> </tr> <tr> <td>Housing</td> <td>0 housing capacity 0 housing remaining</td> <td></td> </tr> <tr> <td>Food</td> <td>23</td> <td>+1/sec Farm</td> </tr> <tr> <td>Metal</td> <td>23</td> <td>+1/sec Mine</td> </tr> <tr> <td>Science</td> <td>0</td> <td></td> </tr> </tbody> </table> <p>You do not have enough resources for this</p> <p>OK</p> <p>Storage Workers Housing Combat Research</p> <p>Shack Bootcamp Barracks</p> <p>Cost: 50 stone Cost: 100 stone Cost: 500 stone</p> <p>Increases housing space by 5 Increases housing space by 10 Increases housing space by 20</p>	Resource	Value	Action	Wood	100	+10/sec Gather	Stone	23	+1/sec Excavate	Housing	0 housing capacity 0 housing remaining		Food	23	+1/sec Farm	Metal	23	+1/sec Mine	Science	0	
Resource	Value	Action																				
Wood	100	+10/sec Gather																				
Stone	23	+1/sec Excavate																				
Housing	0 housing capacity 0 housing remaining																					
Food	23	+1/sec Farm																				
Metal	23	+1/sec Mine																				
Science	0																					
<p>The user tries to buy shack upgrade, has enough stone, but then tries to buy barracks and there is not enough stone</p>	 <table border="1"> <thead> <tr> <th>Resource</th> <th>Value</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>Wood</td> <td>100</td> <td>+10/sec Gather</td> </tr> <tr> <td>Stone</td> <td>80</td> <td>+10/sec Excavate</td> </tr> <tr> <td>Housing</td> <td>5 housing capacity 5 housing remaining</td> <td></td> </tr> <tr> <td>Food</td> <td>24</td> <td>+1/sec Farm</td> </tr> <tr> <td>Metal</td> <td>24</td> <td>+1/sec Mine</td> </tr> <tr> <td>Science</td> <td>0</td> <td></td> </tr> </tbody> </table> <p>You do not have enough resources for this</p> <p>OK</p> <p>Storage Workers Housing Combat Research</p> <p>Shack Bootcamp Barracks</p> <p>Cost: 100 stone Cost: 100 stone Cost: 500 stone</p> <p>Increases housing space by 5 Increases housing space by 10 Increases housing space by 20</p>	Resource	Value	Action	Wood	100	+10/sec Gather	Stone	80	+10/sec Excavate	Housing	5 housing capacity 5 housing remaining		Food	24	+1/sec Farm	Metal	24	+1/sec Mine	Science	0	
Resource	Value	Action																				
Wood	100	+10/sec Gather																				
Stone	80	+10/sec Excavate																				
Housing	5 housing capacity 5 housing remaining																					
Food	24	+1/sec Farm																				
Metal	24	+1/sec Mine																				
Science	0																					

<p>The user tries to buy shack upgrade, has enough stone, but then tries to buy shack again but there is not enough stone</p>	
<p>The user tries to buy shack upgrade, has enough stone, but then tries to buy shack again but there is not enough stone (more than 50 though)</p>	
<p>The user buys bootcamp 3 times from base price of 100 stone</p>	

There is a very small error that I have made that easily fixes the one orange test – the barracks housing space global data variable was set to 15. I have now changed it to 20, and the test now works as it should.

Alex says, “the housing menu makes sense”.

Workers

[V1 24/10/19](#)

Now that the housing works properly, I will work on the workers upgrades next. This is also easy to do because again, nothing else is linked to it other than the resource collection class, one that I have already done.

Again, I start by showing the buttons code:

```
private void btnGatherer_Click(object sender, EventArgs e) {
    buyingWorkers(GlobalData.upgradesCosts[1][0], 2, 0, lblGathererCost);
}

private void btnQuarrying_Click(object sender, EventArgs e) {
    buyingWorkers(GlobalData.upgradesCosts[1][1], 2, 1, lblQuarrierCost);
}

private void btnButcher_Click(object sender, EventArgs e) {
    buyingWorkers(GlobalData.upgradesCosts[1][2], 2, 2, lblButcherCost);
}

private void btnMiner_Click(object sender, EventArgs e) {
    buyingWorkers(GlobalData.upgradesCosts[1][3], 2, 3, lblMinerCost);
}

private void btnTroop_Click(object sender, EventArgs e) {
    buyingWorkers(GlobalData.upgradesCosts[1][4], 2, 4, lblTroopCost);
}
```

Now the actual buyingWorkers function:

```
void buyingWorkers(int cost, int typeToBuy, int typeToBuyFor, Label labelCosts) {
    // If the player does not have enough food OR enough housing space
    if (GlobalData.resourcesData[0][typeToBuy] < cost ||
        GlobalData.housingRemaining == 0) {
        // Print to upgrades tab in logs
        MessageBox.Show("You do not have enough resources or housing space
for this");
        // If they do
    } else {
        // Deduct the cost from the number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase the cost of this upgrade
        GlobalData.upgradesCosts[1][typeToBuyFor] *=
        GlobalData.costMultipliers[1];
        // Increase the rate of the resource
        GlobalData.resourcesData[1][typeToBuyFor] += 1;
        // Decreasing the housing space by 1
        GlobalData.housingRemaining -= 1;
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " +
        GlobalData.upgradesCosts[0][typeToBuyFor] + " food");
    }
}
```

Just before I begin testing, I have realised that I forgot to code the buyingTroops function, as the outcome differs slightly to that of the buyingWorkers function (it increases troop numbers not resource rate). The button code:

```
private void btnTroop_Click(object sender, EventArgs e) {
    buyingTroops(GlobalData.upgradesCosts[3][3], 3, lblTroopCost);
}
```

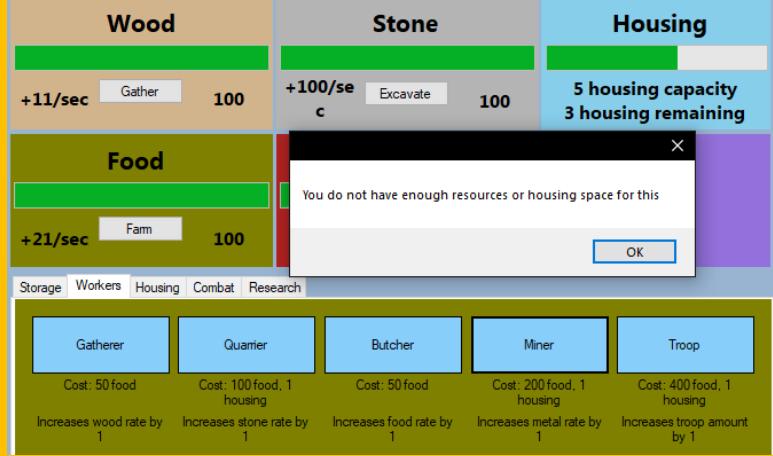
The buyingTroops function:

```
void buyingTroops(int cost, int typeToBuy, Label labelCosts) {
    // If there is not enough metal or housing space
    if (GlobalData.resourcesData[0][typeToBuy] < cost || 
        GlobalData.housingRemaining == 0) {
        // Print to upgrades tab in logs
        MessageBox.Show("You do not have enough metal or housing for this");
    // If there is
    } else {
        // Deduct cost from number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase cost of this upgrade
        GlobalData.upgradesCosts[4][typeToBuy] *=
        GlobalData.costMultipliers[4];
        // Increase the number of troops in the army
        GlobalData.combatData[3][0] += 1;
        // Decrease the housing space by 1
        GlobalData.housingRemaining -= 1;
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[4][typeToBuy]
+ " food");
    }
}
```

Again, as I have not yet started the logs section, I will just print not enough resources into a message box for testing. I will also set the food rate to +10/sec or more because it will not take as long to get to the amount needed to buy the needed upgrade. This will not affect the tests.

Input	Expected Output	Actual Output
The troop upgrade is bought (enough housing and food)	The amount of food goes down by cost, the housing amount goes down by 1, number of troops increased by 1, cost increased by 4 times	<pre>1 reference private void btnTroop_Click(object sender, EventArgs e) { buyingTroops(GlobalData.upgradesCosts[3][3], 3, lblTroopCost); X } /// <summary> /// All housin /// </summary></pre> <p>Exception Unhandled</p> <p>System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'</p>

There is not enough food (but there is enough housing space) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	<p>Wood</p> <p>+10/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>0 housing capacity 0 housing remaining</p> <p>Food</p> <p>+5/sec Farm 100</p> <p>You do not have enough resources or housing space for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 100 food, 1 housing Increases wood rate by 1 Increases stone rate by 1 Increases food rate by 1 Increases metal rate by 1 Increases troop amount by 1</p>
There is not enough housing space (but there is enough food) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	<p>Wood</p> <p>+10/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>0 housing capacity 0 housing remaining</p> <p>Food</p> <p>+5/sec Farm 100</p> <p>You do not have enough resources or housing space for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 100 food, 1 housing Increases wood rate by 1 Increases stone rate by 1 Increases food rate by 1 Increases metal rate by 1 Increases troop amount by 1</p>
There is not enough food or housing space to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	<p>Wood</p> <p>+10/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>0 housing capacity 0 housing remaining</p> <p>Food</p> <p>+5/sec Farm 85</p> <p>You do not have enough resources or housing space for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 100 food, 1 housing Increases wood rate by 1 Increases stone rate by 1 Increases food rate by 1 Increases metal rate by 1 Increases troop amount by 1</p>
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades,	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food	<pre>private void btnTroop_Click(object sender, EventArgs e) { buyingTroops(GlobalData.upgradesCosts[3][3], 3, lblTroopCost); } /// <summary> /// All housing /// </summary> 1 reference private void b</pre> <p>Exception Unhandled</p> <p>System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'</p>

<p>with enough housing space and food for all 4</p>	<p>goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the miner upgrade, the upgrade cost is increased by 4 times, the rate of metal goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the troop upgrade, the upgrade cost is increased by 4 times, the number of troops is increased by 1. Housing space goes down by 1</p>																
<p>The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough housing space but only enough food for the first 2 upgrades</p>	<p>The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1</p>	 <p>Wood</p> <p>+11/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>5 housing capacity 3 housing remaining</p> <p>Food</p> <p>+21/sec Farm 100</p> <p>You do not have enough resources or housing space for this</p> <p>Storage Workers Housing Combat Research</p> <table border="1"> <tbody> <tr> <td>Gatherer</td> <td>Quarier</td> <td>Butcher</td> <td>Miner</td> <td>Troop</td> </tr> <tr> <td>Cost: 50 food</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 50 food</td> <td>Cost: 200 food, 1 housing</td> <td>Cost: 400 food, 1 housing</td> </tr> <tr> <td>Increases wood rate by 1</td> <td>Increases stone rate by 1</td> <td>Increases food rate by 1</td> <td>Increases metal rate by 1</td> <td>Increases troop amount by 1</td> </tr> </tbody> </table>	Gatherer	Quarier	Butcher	Miner	Troop	Cost: 50 food	Cost: 100 food, 1 housing	Cost: 50 food	Cost: 200 food, 1 housing	Cost: 400 food, 1 housing	Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1	Increases troop amount by 1
Gatherer	Quarier	Butcher	Miner	Troop													
Cost: 50 food	Cost: 100 food, 1 housing	Cost: 50 food	Cost: 200 food, 1 housing	Cost: 400 food, 1 housing													
Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1	Increases troop amount by 1													

	by 1. The miner and troop upgrades cannot be bought as is not enough food available. Print “You do not have enough resources or housing space for this” in upgrades logs	
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough food for all 4 but only enough housing space for the first 3 upgrades	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The miner and troop upgrades cannot be bought as is not enough housing space available. Print “You do not have enough resources or housing space for this” in upgrades logs	A screenshot of a C# code editor window. The code shown is a button click event handler: <pre>1 reference private void btnTroop_Click(object sender, EventArgs e) { buyingTroops(GlobalData.upgradesCosts[3][3], 3, lblTroopCost); }</pre> A tooltip is displayed over the line of code `buyingTroops(GlobalData.upgradesCosts[3][3], 3, lblTroopCost);` with the text "Exception Unhandled". Below the tooltip, the error message "System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'" is visible.

Food has reached full capacity, quarrier upgrade is bought	Food amount goes down by cost of upgrade, housing space goes down by 1, stone rate increased by 1/sec	
Food has reached full capacity, troop upgrade is bought	Food amount goes down by cost of upgrade, housing space goes down by 1, troop amount increased by 1	<pre>1 reference private void btnTroop_Click(object sender, EventArgs e) { buyingTroops(GlobalData.upgradesCosts[3][3], 3, lblTroopCost); } /// <summary> /// All housin /// </summary> 1 reference</pre> <p>Exception Unhandled System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'</p>
Food storage upgrade is bought, then butcher upgrade	Wood amount goes down by cost of food storage, food capacity goes up by 1000, butcher upgrade bought so food amount goes down by cost, housing space reduced by 1 and food rate increased by 1/sec	
Food has reached just about enough to buy gatherer upgrade, housing space left is 1	Food and housing goes down to 0, food rate increased by 1/sec	

I have found 2 problems:

1. When pressing the troop button, something is called out of bounds of an array and subsequently crashes the program
2. When a button is pressed, the cost label is reset to 50, when the internal cost still goes up by 4x

The fixes both times were very simple:

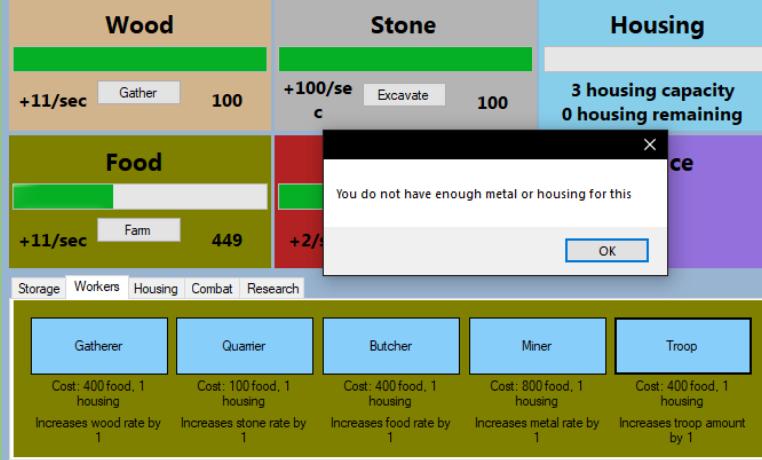
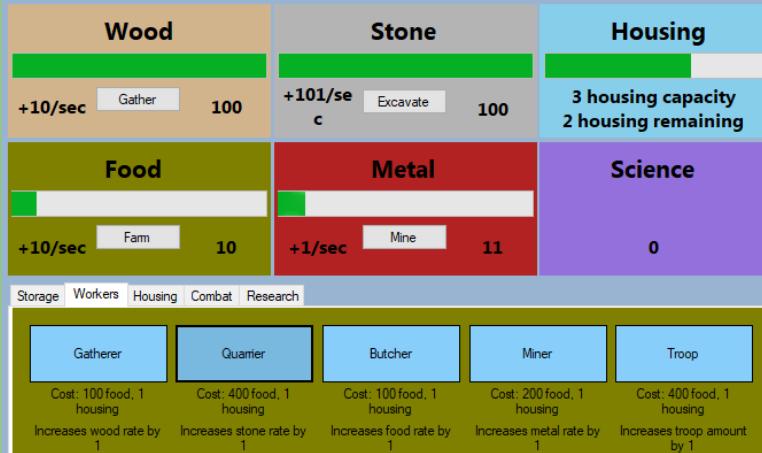
1. The code in the troop button calling buyingTroops was trying to take a value from a non-existent index of the upgradesCosts array – I just had to fix the element it needed to take from
2. The printed cost was taking the cost from the storage parts of the upgradesCosts variable. I also added a “, 1 housing” bit at the end in order to match the starting cost print to the automatic prints

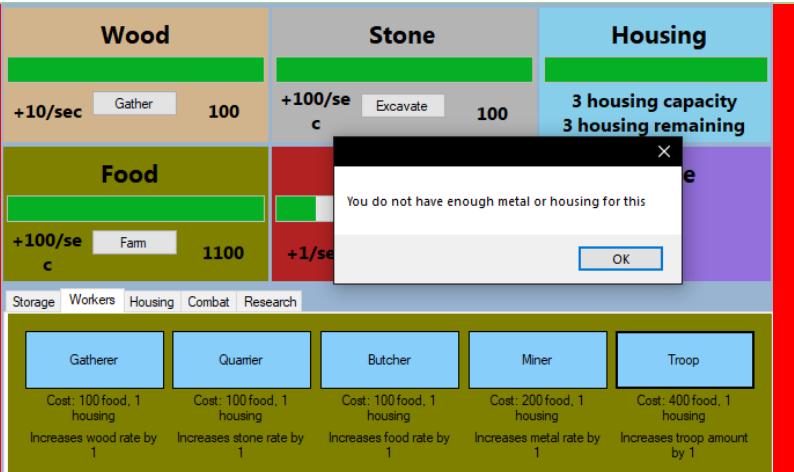
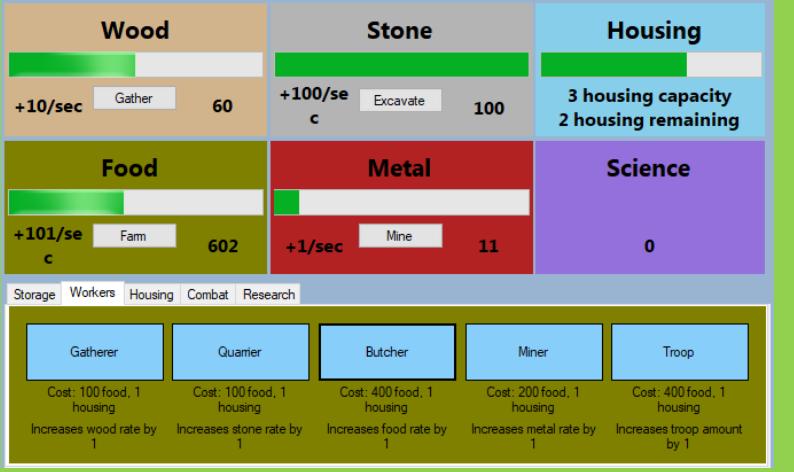
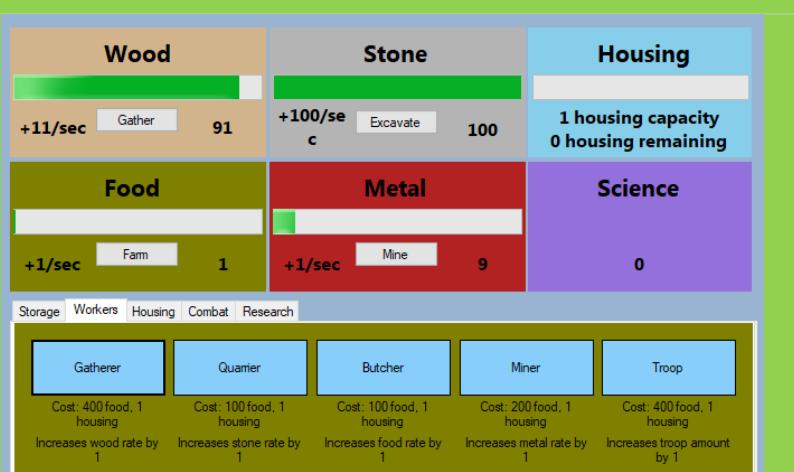
Now to retest the whole section.

Input	Expected Output	Actual Output
The troop upgrade is bought (enough housing and metal)	The amount of metal goes down by cost, the housing amount goes down by 1, number of troops increased by 1, cost increased by 4 times	<p>The screenshot shows a game interface with three main resource sections: Wood, Stone, and Housing. The Wood section has a green bar at 100, a +10/sec Gather button, and a 100 value. The Stone section has a green bar at 100, a +100/sec Excavate button, and a 100 value. The Housing section has a green bar at 1, labeled "1 housing capacity" and "1 housing remaining". Below these are Food and Farm sections. A tooltip window is open, stating "You do not have enough metal or housing for this" with an OK button. At the bottom, there are tabs for Storage, Workers, Housing, Combat, and Research, followed by five worker buttons: Gatherer, Quarier, Butcher, Miner, and Troop, each with their respective costs and descriptions.</p>
There is not enough food (but there is enough housing space) to buy the butcher upgrade	Print “You do not have enough resources or housing space for this” in upgrades logs, upgrade is not bought	<p>The screenshot shows a game interface similar to the previous one, but with different resource levels. The Wood section has a green bar at 100, a +10/sec Gather button, and a 100 value. The Stone section has a green bar at 100, a +100/sec Excavate button, and a 100 value. The Housing section has a green bar at 5, labeled "5 housing capacity" and "5 housing remaining". Below these are Food and Farm sections. A tooltip window is open, stating "You do not have enough resources or housing space for this" with an OK button. At the bottom, there are tabs for Storage, Workers, Housing, Combat, and Research, followed by five worker buttons: Gatherer, Quarier, Butcher, Miner, and Troop, each with their respective costs and descriptions.</p>

There is not enough housing space (but there is enough food) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	<p>Wood</p> <p>+10/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>0 housing capacity 0 housing remaining</p> <p>Food</p> <p>+5/sec Farm 100</p> <p>You do not have enough resources or housing space for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 100 food, 1 housing Increases wood rate by 1</p> <p>Cost: 100 food, 1 housing Increases stone rate by 1</p> <p>Cost: 100 food, 1 housing Increases food rate by 1</p> <p>Cost: 200 food, 1 housing Increases metal rate by 1</p> <p>Cost: 400 food, 1 housing Increases troop amount by 1</p>
There is not enough food or housing space to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	<p>Wood</p> <p>+10/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>0 housing capacity 0 housing remaining</p> <p>Food</p> <p>+5/sec Farm 85</p> <p>You do not have enough resources or housing space for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 100 food, 1 housing Increases wood rate by 1</p> <p>Cost: 100 food, 1 housing Increases stone rate by 1</p> <p>Cost: 100 food, 1 housing Increases food rate by 1</p> <p>Cost: 200 food, 1 housing Increases metal rate by 1</p> <p>Cost: 400 food, 1 housing Increases troop amount by 1</p>
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough housing space and food for all 4	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the miner upgrade, the upgrade cost	<p>Wood</p> <p>+11/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>6 housing capacity 3 housing remaining</p> <p>Food</p> <p>+11/sec Farm 784</p> <p>You do not have enough metal or housing for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 400 food, 1 housing Increases wood rate by 1</p> <p>Cost: 100 food, 1 housing Increases stone rate by 1</p> <p>Cost: 400 food, 1 housing Increases food rate by 1</p> <p>Cost: 800 food, 1 housing Increases metal rate by 1</p> <p>Cost: 400 food, 1 housing Increases troop amount by 1</p>

	<p>is increased by 4 times, the rate of metal goes up by 1/sec. Housing space goes down by 1. The metal goes down by the cost of the troop upgrade, the upgrade cost is increased by 4 times, the number of troops is increased by 1. Housing space goes down by 1</p>	
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough housing space but only enough food for the first 2 upgrades	<p>The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The miner and troop upgrades cannot be bought as is not enough food available. Print "You do not have enough resources or housing space for this" in upgrades logs</p>	<p>Wood</p> <p>+11/sec Gather 100</p> <p>Stone</p> <p>+100/sec Excavate 100</p> <p>Housing</p> <p>6 housing capacity 4 housing remaining</p> <p>Food</p> <p>+11/sec Farm 55</p> <p>You do not have enough resources or housing space for this</p> <p>Gatherer Quarier Butcher Miner Troop</p> <p>Cost: 400 food, 1 housing Increases wood rate by 1</p> <p>Cost: 100 food, 1 housing Increases stone rate by 1</p> <p>Cost: 400 food, 1 housing Increases food rate by 1</p> <p>Cost: 200 food, 1 housing Increases metal rate by 1</p> <p>Cost: 400 food, 1 housing Increases troop amount by 1</p>

<p>The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough food for all 4 but only enough housing space for the first 3 upgrades</p>	<p>The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The miner and troop upgrades cannot be bought as is not enough housing space available. Print "You do not have enough resources or housing space for this" in upgrades logs</p>	
<p>Food has reached full capacity, quarrier upgrade is bought</p>	<p>Food amount goes down by cost of upgrade, housing space goes down by 1, stone rate increased by 1/sec</p>	

Metal has reached full capacity, troop upgrade is bought	Metal amount goes down by cost of upgrade, housing space goes down by 1, troop amount increased by 1	
Food storage upgrade is bought, then butcher upgrade	Wood amount goes down by cost of food storage, food capacity goes up by 1000, butcher upgrade bought so food amount goes down by cost, housing space reduced by 1 and food rate increased by 1/sec	
Food has reached just about enough to buy gatherer upgrade, housing space left is 1	Food and housing goes down to 0, food rate increased by 1/sec	

When doing these tests, I incorrectly thought that the troop costs food, when actually it costs metal. Therefore, I marked many tests incorrect, because the code still had the cost of metal. I have also noticed that the GUI is slightly wrong, so I have corrected it in Upgrades GUI V2. I will not repeat the tests that I thought failed:

Input	Expected Output	Actual Output															
The troop upgrade is bought (enough housing and metal)	The amount of metal goes down by cost, the housing amount goes down by 1, number of troops increased by 1, cost increased by 4 times	<p>Wood</p> <p>Stone</p> <p>Housing</p> <p>Food</p> <p>Metal</p> <p>Science</p> <p>Storage: Workers, Housing, Combat, Research</p> <table border="1"> <tr> <td>Gatherer</td> <td>Quarier</td> <td>Butcher</td> <td>Miner</td> <td>Troop</td> </tr> <tr> <td>Cost: 100 food, 1 housing</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 200 food, 1 housing</td> <td>Cost: 800 metal, 1 housing</td> </tr> <tr> <td>Increases wood rate by 1</td> <td>Increases stone rate by 1</td> <td>Increases food rate by 1</td> <td>Increases metal rate by 1</td> <td>Increases troop amount by 1</td> </tr> </table>	Gatherer	Quarier	Butcher	Miner	Troop	Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 200 food, 1 housing	Cost: 800 metal, 1 housing	Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1	Increases troop amount by 1
Gatherer	Quarier	Butcher	Miner	Troop													
Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 200 food, 1 housing	Cost: 800 metal, 1 housing													
Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1	Increases troop amount by 1													
The gatherer upgrade is bought, then the butcher upgrade, then the miner and troop upgrades, with enough housing space and food for all 4	The food goes down by the cost of the gatherer upgrade, the upgrade cost is increased by 4 times, the rate of wood goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the butcher upgrade, the upgrade cost is increased by 4 times, the rate of food goes up by 1/sec. Housing space goes down by 1. The food goes down by the cost of the miner upgrade, the upgrade cost is increased by 4 times, the rate of metal goes up by 1/sec. Housing space goes down by 1. The metal goes down by the cost of the troop upgrade, the upgrade cost is increased by 4 times, the	<p>Wood</p> <p>Stone</p> <p>Housing</p> <p>Food</p> <p>Metal</p> <p>Science</p> <p>Storage: Workers, Housing, Combat, Research</p> <table border="1"> <tr> <td>Gatherer</td> <td>Quarier</td> <td>Butcher</td> <td>Miner</td> <td>Troop</td> </tr> <tr> <td>Cost: 400 food, 1 housing</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 400 food, 1 housing</td> <td>Cost: 800 food, 1 housing</td> <td>Cost: 800 metal, 1 housing</td> </tr> <tr> <td>Increases wood rate by 1</td> <td>Increases stone rate by 1</td> <td>Increases food rate by 1</td> <td>Increases metal rate by 1</td> <td>Increases troop amount by 1</td> </tr> </table>	Gatherer	Quarier	Butcher	Miner	Troop	Cost: 400 food, 1 housing	Cost: 100 food, 1 housing	Cost: 400 food, 1 housing	Cost: 800 food, 1 housing	Cost: 800 metal, 1 housing	Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1	Increases troop amount by 1
Gatherer	Quarier	Butcher	Miner	Troop													
Cost: 400 food, 1 housing	Cost: 100 food, 1 housing	Cost: 400 food, 1 housing	Cost: 800 food, 1 housing	Cost: 800 metal, 1 housing													
Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1	Increases troop amount by 1													

	number of troops is increased by 1. Housing space goes down by 1																					
Metal has reached full capacity, troop upgrade is bought	Metal amount goes down by cost of upgrade, housing space goes down by 1, troop amount increased by 1	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #e0c0a0;">Wood</th> <th style="background-color: #d0e0ff;">Stone</th> <th style="background-color: #b0ffb0;">Housing</th> </tr> </thead> <tbody> <tr> <td>+10/sec Gather 40</td> <td>+100/sec Excavate 100</td> <td>3 housing capacity 2 housing remaining</td> </tr> <tr> <th style="background-color: #a0c090;">Food</th> <th style="background-color: #f08080;">Metal</th> <th style="background-color: #b0a0ff;">Science</th> </tr> <tr> <td>+100/sec Farm 100</td> <td>+100/sec Mine 300</td> <td>0</td> </tr> <tr> <td colspan="3" style="text-align: center; font-size: small;"> Storage Workers Housing Combat Research </td> </tr> <tr> <td>Gatherer Cost: 100 food, 1 housing Increases wood rate by 1</td> <td>Quarier Cost: 100 food, 1 housing Increases stone rate by 1</td> <td>Butcher Cost: 100 food, 1 housing Increases food rate by 1</td> <td>Miner Cost: 200 food, 1 housing Increases metal rate by 1</td> <td>Troop Cost: 800 metal, 1 housing Increases troop amount by 1</td> </tr> </tbody> </table>	Wood	Stone	Housing	+10/sec Gather 40	+100/sec Excavate 100	3 housing capacity 2 housing remaining	Food	Metal	Science	+100/sec Farm 100	+100/sec Mine 300	0	Storage Workers Housing Combat Research			Gatherer Cost: 100 food, 1 housing Increases wood rate by 1	Quarier Cost: 100 food, 1 housing Increases stone rate by 1	Butcher Cost: 100 food, 1 housing Increases food rate by 1	Miner Cost: 200 food, 1 housing Increases metal rate by 1	Troop Cost: 800 metal, 1 housing Increases troop amount by 1
Wood	Stone	Housing																				
+10/sec Gather 40	+100/sec Excavate 100	3 housing capacity 2 housing remaining																				
Food	Metal	Science																				
+100/sec Farm 100	+100/sec Mine 300	0																				
Storage Workers Housing Combat Research																						
Gatherer Cost: 100 food, 1 housing Increases wood rate by 1	Quarier Cost: 100 food, 1 housing Increases stone rate by 1	Butcher Cost: 100 food, 1 housing Increases food rate by 1	Miner Cost: 200 food, 1 housing Increases metal rate by 1	Troop Cost: 800 metal, 1 housing Increases troop amount by 1																		

V2 25/10/19

Whilst working on the combat section, I am printing the new health, block and attack after each upgrade is bought into logs (temporarily into a message box until the logs section is ready for it to be switched). I realised that because there is no indicator for number of troops or troops data in the resources collection tab, I have decided to also add this message about the number of troops. A couple of new lines have been added to the buyingTroops function:

```
// Print the new number of troops to upgrades tab
MessageBox.Show("Troop health increased by 100, total is " +
GlobalData.combatData[0][0].ToString());
```

Combat

V1 25/10/19

```

private void btnHealth_Click(object sender, EventArgs e) {
    buyingCombat(GlobalData.upgradesCosts[4][0], 3, 0, 100, lblHealthCost);
}

private void btnBlock_Click(object sender, EventArgs e) {
    buyingCombat(GlobalData.upgradesCosts[4][1], 3, 1, 250, lblBlockCost);
}

private void btnAttack_Click(object sender, EventArgs e) {
    buyingCombat(GlobalData.upgradesCosts[4][2], 3, 2, 150, lblAttackCost);
}

```

```

void buyingCombat(int cost, int typeToBuy, int combatType, int buff, Label
labelCosts) {
    // If the player has enough resources
    if (GlobalData.resourcesData[0][typeToBuy] < cost) {
        // Print message into upgrades logs
        MessageBox.Show("You do not have enough metal for this");
        // TODO: Link buyingCombat to upgrades part of logs
    } else {
        // Deduct cost from number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase cost of this upgrade
        GlobalData.upgradesCosts[4][combatType] *=
GlobalData.costMultipliers[4];
        // Buff the combat type
        GlobalData.combatData[combatType][0] += buff;
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[4][combatType]
+ " metal");

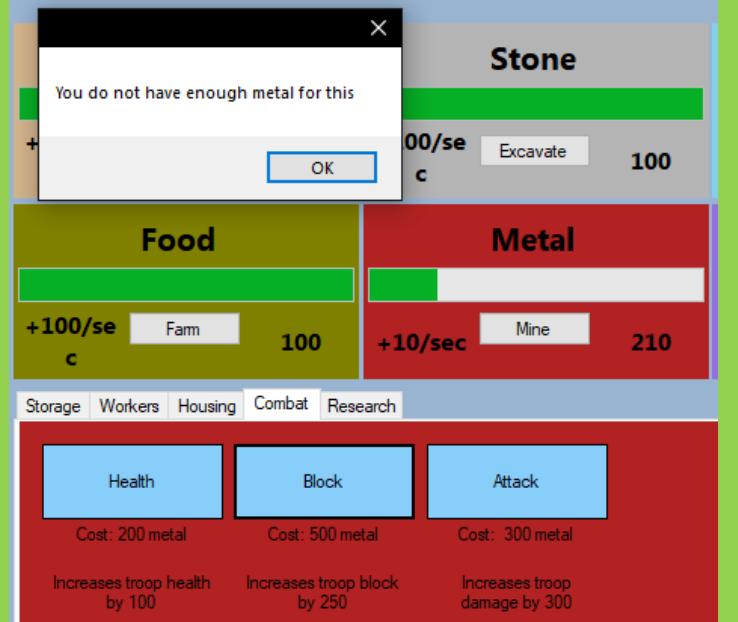
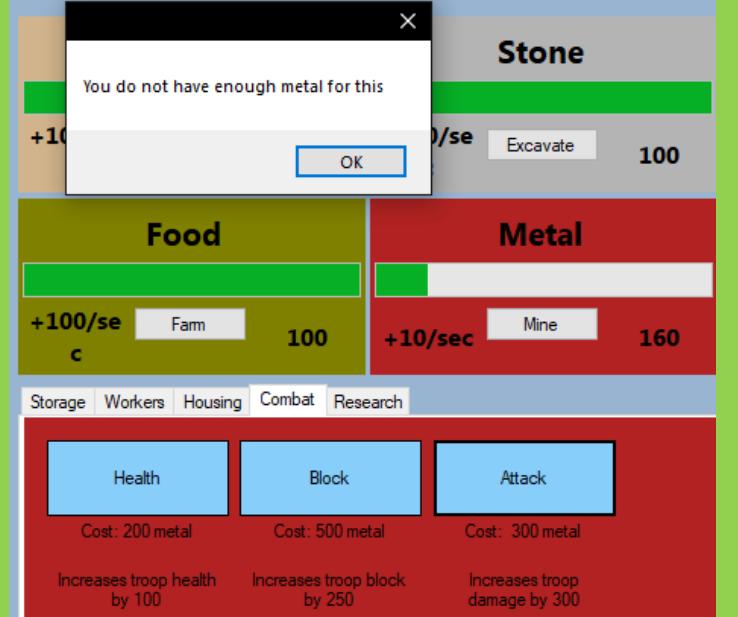
        switch (combatType) {
            case 0:
                // Print health increase to logs
                MessageBox.Show("Troop health increased by 100, total
is " + GlobalData.combatData[0][0].ToString());
                break;
            case 1:
                // Print block increase to logs
                MessageBox.Show("Troop health increased by 250, total
is " + GlobalData.combatData[1][0].ToString());
                break;
            case 2:
                // Print attack increase to logs
                MessageBox.Show("Troop health increased by 250, total
is " + GlobalData.combatData[2][0].ToString());
                break;
        }
    }
}

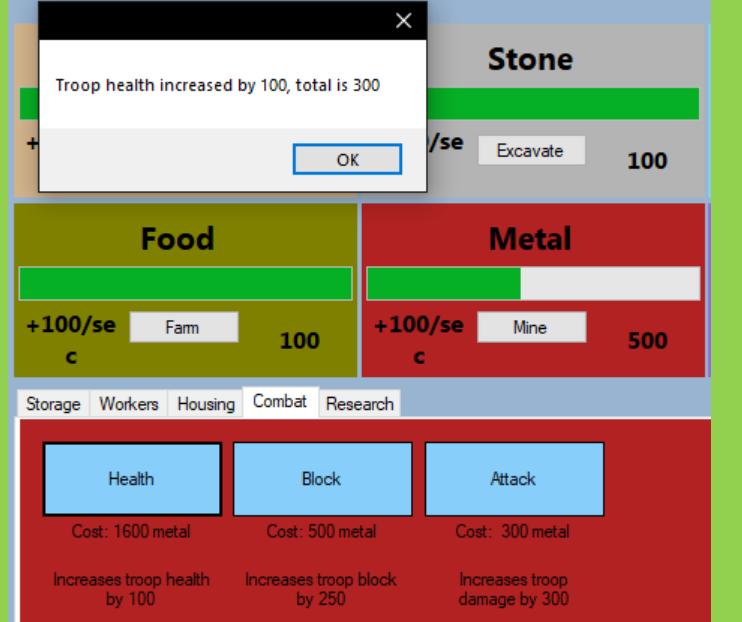
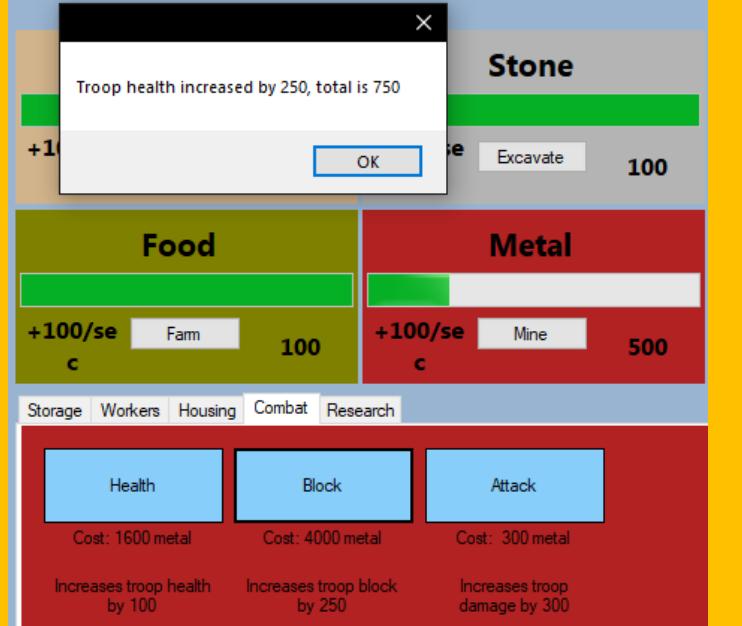
```

As the logs section has not been coded yet, the “printing to logs” part will be temporarily replaced with showing a message box for testing purposes.

Input	Expected Output	Actual Output
Troop health upgrade bought	Metal reduced by cost, health goes up by 100	<p>Troop health increased by 100, total is 100</p> <p>The screenshot shows a game interface with resource storage and troop statistics. At the top right, there's a 'Stone' storage bar at 100. Below it, there are two resource bars: 'Food' (green) at +100/se and 'Metal' (red) at +100/se. Underneath these are buttons for 'Farm' and 'Mine'. Troop stats at the bottom show Health at 100, Block at 500 metal, and Attack at 300 metal. Descriptions below the stats say 'Increases troop health by 100', 'Increases troop block by 250', and 'Increases troop damage by 300'.</p>
Troop block upgrade bought	Metal reduced by cost, block goes up by 250	<p>Troop health increased by 250, total is 250</p> <p>The screenshot shows a game interface with resource storage and troop statistics. At the top right, there's a 'Stone' storage bar at 100. Below it, there are two resource bars: 'Food' (green) at +100/se and 'Metal' (red) at +100/se. Underneath these are buttons for 'Farm' and 'Mine'. Troop stats at the bottom show Health at 100, Block at 1000 metal, and Attack at 300 metal. Descriptions below the stats say 'Increases troop health by 100', 'Increases troop block by 250', and 'Increases troop damage by 300'.</p>

Troop attack upgrade bought	Metal reduced by cost, attack goes up by 300	<p>A screenshot of a game interface. A tooltip box is open, stating "Troop health increased by 250, total is 150". Below the tooltip is a resource bar for "Stone" with a value of 100. Underneath the resource bars are three buttons: "Health" (Cost: 200 metal, Increases troop health by 100), "Block" (Cost: 500 metal, Increases troop block by 250), and "Attack" (Cost: 600 metal, Increases troop damage by 300). The background shows a "Food" section with a green bar and a "Metal" section with a red bar.</p>
Troop health upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console	<p>A screenshot of a game interface. A tooltip box is open, stating "You do not have enough metal for this". Below the tooltip is a resource bar for "Stone" with a value of 100. Underneath the resource bars are three buttons: "Health" (Cost: 200 metal, Increases troop health by 100), "Block" (Cost: 500 metal, Increases troop block by 250), and "Attack" (Cost: 300 metal, Increases troop damage by 300). The background shows a "Food" section with a green bar and a "Metal" section with a red bar.</p>

Troop block upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console	
Troop attack upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console	

Troop health upgrade bought 3 times from base value of 200 metal	Metal reduced by 200, health goes up by 100. Metal reduced by 400, health goes up by 100. Metal reduced by 800, health goes up by 100.	 <p>A screenshot of a game interface showing resource management and troop upgrade details. At the top right, there's a 'Stone' storage bar with 100 units, and an 'Excavate' button. Below it are 'Food' and 'Metal' storage bars, both with '+100/se c' labels and 'Farm' and 'Mine' buttons respectively. The 'Food' bar has 100 units, and the 'Metal' bar has 500 units. A central message box says 'Troop health increased by 100, total is 300' with an 'OK' button. Below the storage bars are tabs for 'Storage', 'Workers', 'Housing', 'Combat', and 'Research'. Under the 'Combat' tab, three buttons are shown: 'Health' (Cost: 1600 metal, Increases troop health by 100), 'Block' (Cost: 500 metal, Increases troop block by 250), and 'Attack' (Cost: 300 metal, Increases troop damage by 300).</p>
Troop block upgrade bought 3 times from base value of 200 metal	Metal reduced by 500, block goes up by 250. Metal reduced by 1000, block goes up by 250. Metal reduced by 2000, block goes up by 250.	 <p>A screenshot of a game interface showing resource management and troop upgrade details. At the top right, there's a 'Stone' storage bar with 100 units, and an 'Excavate' button. Below it are 'Food' and 'Metal' storage bars, both with '+100/se c' labels and 'Farm' and 'Mine' buttons respectively. The 'Food' bar has 100 units, and the 'Metal' bar has 500 units. A central message box says 'Troop health increased by 250, total is 750' with an 'OK' button. Below the storage bars are tabs for 'Storage', 'Workers', 'Housing', 'Combat', and 'Research'. Under the 'Combat' tab, three buttons are shown: 'Health' (Cost: 1600 metal, Increases troop health by 100), 'Block' (Cost: 4000 metal, Increases troop block by 250), and 'Attack' (Cost: 300 metal, Increases troop damage by 300).</p>

<p>Troop attack upgrade bought 3 times from base value of 200 metal</p> <p>Metal reduced by 300, attack goes up by 300. Metal reduced by 600, attack goes up by 300. Metal reduced by 1200, attack goes up by 300.</p>	 <p>The screenshot shows a game interface with resource bars and upgrade options. The top right shows a green bar for Stone with a value of 100, and an 'Excavate' button. Below it are two more resource bars: Food (green, +100/se, Farm, 100) and Metal (red, +100/se, Mine, 500). A central message box says 'Troop health increased by 250, total is 450'. At the bottom, there are three upgrade buttons: Health (Cost: 1600 metal), Block (Cost: 4000 metal), and Attack (Cost: 2400 metal). Each button has a corresponding description below it: 'Increases troop health by 100', 'Increases troop block by 250', and 'Increases troop damage by 300' respectively.</p>
--	--

The reason there are 4 orange tests is because with the code:

```
        switch (combatType) {
            case 0:
                // Print health increase to logs
                MessageBox.Show("Troop health increased by 100, total
is " + GlobalData.combatData[0][0].ToString());
                break;
            case 1:
                // Print block increase to logs
                MessageBox.Show("Troop health increased by 250, total
is " + GlobalData.combatData[1][0].ToString());
                break;
            case 2:
                // Print attack increase to logs
                MessageBox.Show("Troop health increased by 250, total
is " + GlobalData.combatData[2][0].ToString());
                break;
        }
    }
}
```

I accidentally left the messages all saying “health” instead of block and attack respectively, and I also forgot to change the attack part from 250 to 150. This explains why the tests work but the output is wrong. Now that I have corrected them, I will not need to retest because it was just an outputting error.

Research

V1 26/10/19

```

private void btnAqueducts_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][0], 0, lblAqueductsCost);
}

private void btnStampMill_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][1], 1, lblStampMillCost);
}

private void btnTripHammer_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][2], 2, lblTripHammerCost);
}

private void btnHushing_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][3], 3, lblHushingCost);
}

private void btnVilla_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][4], 4, lblVillaCost);
}

```

```

void buyingResearch(int cost, int researchType, Label labelCosts) {
    // If the player does not have enough science
    if (GlobalData.scienceData < cost) {
        // Print in upgrades logs
        MessageBox.Show("You do not have enough science for this");
    } else {
        // Deduct cost from total science
        GlobalData.scienceData -= cost;
        // Increase cost
        GlobalData.upgradesCosts[2][researchType] *=
GlobalData.costMultipliers[2];
        // Output new cost
        labelCosts.Text = ("Cost: " +
GlobalData.upgradesCosts[2][researchType] + " science");

        switch (researchType) {
            // Aqueducts
            case 0:
                GlobalData.resourcesData[1][0] *= 5;
                GlobalData.resourcesData[1][2] *= 5;
                break;
            // Stamp-mill
            case 1:
                GlobalData.resourcesData[1][1] *= 2;
                break;
            // Trip-hammer
            case 2:
                GlobalData.resourcesData[1][3] *= 2;
                break;
        }
    }
}

```

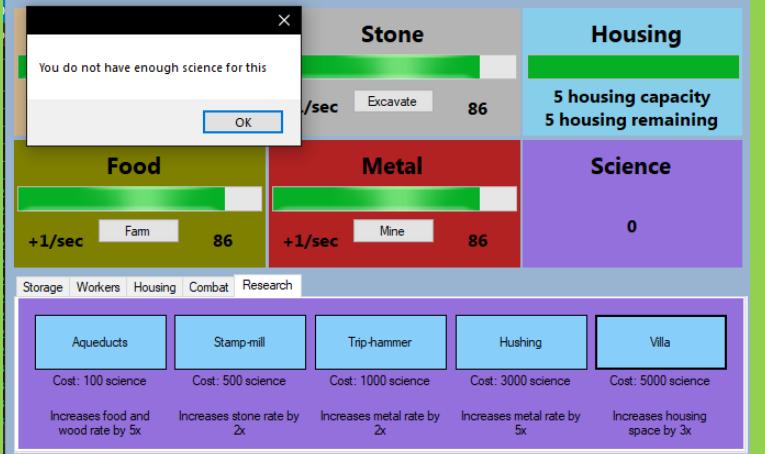
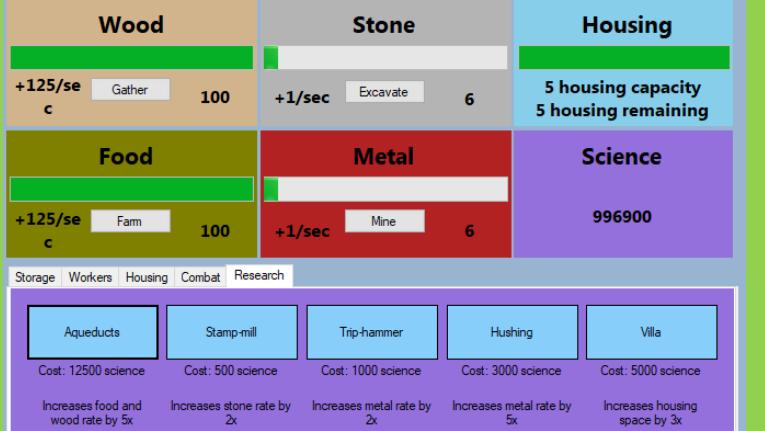
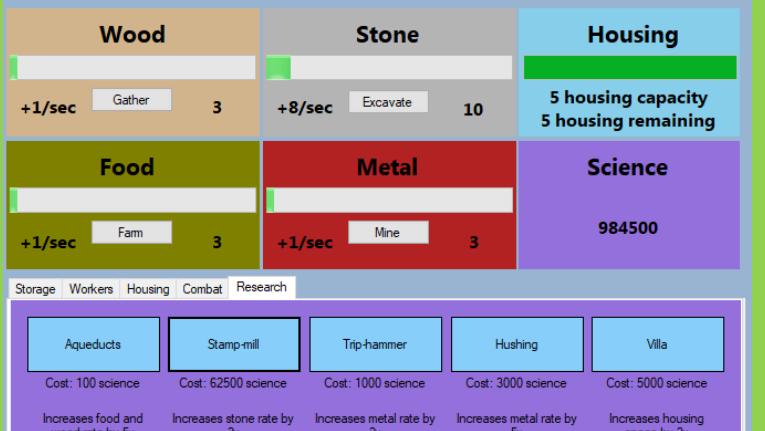
```
// Hushing
case 3:
    GlobalData.resourcesData[1][3] *= 5;
    break;
// Villa
case 4:
    GlobalData.totalHousing *= 3;
    break;
}
}
```

For testing I've just set the amount of science to 1,000,000 (except for the ones where there is not enough science) because it is faster to test.

Input	Expected Output	Actual Output																	
Aqueducts upgrade bought	Science reduced by cost, food and wood rate go up by 5x, cost multiplied by 5x	<p>Wood</p> <table border="1"> <tr> <td>+5/sec</td> <td>Gather</td> <td>13</td> </tr> </table> <p>Stone</p> <table border="1"> <tr> <td>+1/sec</td> <td>Excavate</td> <td>5</td> </tr> </table> <p>Housing</p> <p>5 housing capacity 5 housing remaining</p> <p>Food</p> <table border="1"> <tr> <td>+5/sec</td> <td>Farm</td> <td>13</td> </tr> </table> <p>Metal</p> <table border="1"> <tr> <td>+1/sec</td> <td>Mine</td> <td>5</td> </tr> </table> <p>Science</p> <p>999900</p> <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr> <td>Aqueducts</td> <td>Stamp-mill</td> <td>Trip-hammer</td> <td>Hushing</td> <td>Villa</td> </tr> </table> <p>Cost: 500 science Cost: 500 science Cost: 1000 science Cost: 3000 science Cost: 5000 science</p> <p>Increases food and wood rate by 5x Increases stone rate by 2x Increases metal rate by 2x Increases metal rate by 5x Increases housing space by 3x</p>	+5/sec	Gather	13	+1/sec	Excavate	5	+5/sec	Farm	13	+1/sec	Mine	5	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa
+5/sec	Gather	13																	
+1/sec	Excavate	5																	
+5/sec	Farm	13																	
+1/sec	Mine	5																	
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa															
Stamp-mill upgrade bought	Science reduced by cost, stone rate goes up by 2x, cost multiplied by 5x	<p>Wood</p> <table border="1"> <tr> <td>+1/sec</td> <td>Gather</td> <td>3</td> </tr> </table> <p>Stone</p> <table border="1"> <tr> <td>+2/sec</td> <td>Excavate</td> <td>4</td> </tr> </table> <p>Housing</p> <p>5 housing capacity 5 housing remaining</p> <p>Food</p> <table border="1"> <tr> <td>+1/sec</td> <td>Farm</td> <td>3</td> </tr> </table> <p>Metal</p> <table border="1"> <tr> <td>+1/sec</td> <td>Mine</td> <td>3</td> </tr> </table> <p>Science</p> <p>999500</p> <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr> <td>Aqueducts</td> <td>Stamp-mill</td> <td>Trip-hammer</td> <td>Hushing</td> <td>Villa</td> </tr> </table> <p>Cost: 100 science Cost: 2500 science Cost: 1000 science Cost: 3000 science Cost: 5000 science</p> <p>Increases food and wood rate by 5x Increases stone rate by 2x Increases metal rate by 2x Increases metal rate by 5x Increases housing space by 3x</p>	+1/sec	Gather	3	+2/sec	Excavate	4	+1/sec	Farm	3	+1/sec	Mine	3	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa
+1/sec	Gather	3																	
+2/sec	Excavate	4																	
+1/sec	Farm	3																	
+1/sec	Mine	3																	
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa															

Trip-hammer upgrade bought	Science reduced by cost, metal rate goes up by 2x, cost multiplied by 5x	
Hushing upgrade bought	Science reduced by cost, metal rate goes up by 5x, cost multiplied by 5x	
Villa upgrade bought	Science reduced by cost, housing space goes up by 3x, cost multiplied by 5x	
Aqueducts upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	

Stamp-mill upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	<p>Wood</p> <p>+1/sec Gather 29</p> <p>Stone</p> <p>+1/sec Excavate 29</p> <p>Housing</p> <p>5 housing capacity 5 housing remaining</p> <p>Food</p> <p>+1/sec Farm 29</p> <p>You do not have enough science for this</p> <p>Science</p> <p>0</p> <p>Aqueducts Cost: 100 science Increases food and wood rate by 5x</p> <p>Stamp-mill Cost: 500 science Increases stone rate by 2x</p> <p>Trip-hammer Cost: 1000 science Increases metal rate by 2x</p> <p>Hushing Cost: 3000 science Increases metal rate by 5x</p> <p>Villa Cost: 5000 science Increases housing space by 3x</p>
Trip-hammer upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	<p>Stone</p> <p>+1/sec Excavate 50</p> <p>Housing</p> <p>5 housing capacity 5 housing remaining</p> <p>Food</p> <p>+1/sec Farm 50</p> <p>Metal</p> <p>+1/sec Mine 50</p> <p>Science</p> <p>0</p> <p>Aqueducts Cost: 100 science Increases food and wood rate by 5x</p> <p>Stamp-mill Cost: 500 science Increases stone rate by 2x</p> <p>Trip-hammer Cost: 1000 science Increases metal rate by 2x</p> <p>Hushing Cost: 3000 science Increases metal rate by 5x</p> <p>Villa Cost: 5000 science Increases housing space by 3x</p>
Hushing upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	<p>Stone</p> <p>+1/sec Excavate 70</p> <p>Housing</p> <p>5 housing capacity 5 housing remaining</p> <p>Food</p> <p>+1/sec Farm 70</p> <p>Metal</p> <p>+1/sec Mine 70</p> <p>Science</p> <p>0</p> <p>Aqueducts Cost: 100 science Increases food and wood rate by 5x</p> <p>Stamp-mill Cost: 500 science Increases stone rate by 2x</p> <p>Trip-hammer Cost: 1000 science Increases metal rate by 2x</p> <p>Hushing Cost: 3000 science Increases metal rate by 5x</p> <p>Villa Cost: 5000 science Increases housing space by 3x</p>

Villa upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	
Aqueducts research bought 3 times from base value of 100 science, all at +1/sec	Science reduced by 100, food and wood rate go up to +5/sec. Science reduced by 500, food and wood rate go up to +25/sec. Science reduced by 2500, food and wood rate go up to +125/sec.	
Stamp-mill research bought 3 times from base value of 500 science, all at +1/sec	Science reduced by 500, stone rate goes up to +2/sec. Science reduced by 2500, stone rate goes up to +4/sec. Science reduced by 12500, stone rate goes up to +8/sec.	
Trip-hammer research bought 3 times from base value of 1000 science, all at +1/sec	Science reduced by 1000, metal rate goes up to +2/sec. Science reduced by 5000, metal rate goes up to +4/sec. Science reduced by 25000, metal rate goes up to +8/sec.	

<p>Hushing research bought 3 times from base value of 3000 science, all at +1/sec</p> <p>Villa research bought 3 times from base value of 5000 science, housing at 5</p>	<p>Science reduced by 3000, metal rate goes up to +5/sec. Science reduced by 15000, metal rate goes up to +25/sec. Science reduced by 75000, metal rate goes up to +125/sec.</p> <p>Science reduced by 5000, housing space increased to 15. Science reduced by 25000, housing space increased to 45. Science reduced by 125000, housing space increased to 135.</p>	<table border="1"> <thead> <tr> <th>Storage</th> <th>Workers</th> <th>Housing</th> <th>Combat</th> <th>Research</th> </tr> </thead> <tbody> <tr> <td>Aqueducts</td> <td>Stamp-mill</td> <td>Trip-hammer</td> <td>Hushing</td> <td>Villa</td> </tr> <tr> <td>Cost: 100 science</td> <td>Cost: 500 science</td> <td>Cost: 1000 science</td> <td>Cost: 375000 science</td> <td>Cost: 5000 science</td> </tr> <tr> <td>Increases food and wood rate by 5x</td> <td>Increases stone rate by 2x</td> <td>Increases metal rate by 2x</td> <td>Increases metal rate by 5x</td> <td>Increases housing space by 3x</td> </tr> </tbody> </table>	Storage	Workers	Housing	Combat	Research	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 100 science	Cost: 500 science	Cost: 1000 science	Cost: 375000 science	Cost: 5000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x
Storage	Workers	Housing	Combat	Research																		
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa																		
Cost: 100 science	Cost: 500 science	Cost: 1000 science	Cost: 375000 science	Cost: 5000 science																		
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x																		
<table border="1"> <thead> <tr> <th>Storage</th> <th>Workers</th> <th>Housing</th> <th>Combat</th> <th>Research</th> </tr> </thead> <tbody> <tr> <td>Aqueducts</td> <td>Stamp-mill</td> <td>Trip-hammer</td> <td>Hushing</td> <td>Villa</td> </tr> <tr> <td>Cost: 100 science</td> <td>Cost: 500 science</td> <td>Cost: 1000 science</td> <td>Cost: 3000 science</td> <td>Cost: 625000 science</td> </tr> <tr> <td>Increases food and wood rate by 5x</td> <td>Increases stone rate by 2x</td> <td>Increases metal rate by 2x</td> <td>Increases metal rate by 5x</td> <td>Increases housing space by 3x</td> </tr> </tbody> </table>	Storage	Workers	Housing	Combat	Research	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 100 science	Cost: 500 science	Cost: 1000 science	Cost: 3000 science	Cost: 625000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x		
Storage	Workers	Housing	Combat	Research																		
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa																		
Cost: 100 science	Cost: 500 science	Cost: 1000 science	Cost: 3000 science	Cost: 625000 science																		
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x																		

Review 1

- Green – Feature finished (does not matter if there is a minor error or change)
- Orange – Feature about to be worked on next
- Red – Feature skipped/not going to be worked on

Criteria	How to evidence criteria being met	Section Code
Section A: Resources/upgrades system		
Resources go up over time	2 screenshots – first one taken before, second one taken after to show progress with resources	A1
Resource rate increased through upgrade	Screenshot of increased rate	A2
Resources reach “milestone” where the number shortens	Screenshot of 4,000 resource shown has 4k resource	A3 <i>Optional</i>
Button to manually increase rate of resource collection increased, button turns brown, all other buttons turn grey and toggle off, that resource rate increases by 10%	2 screenshots – first one taken before with one button pressed, second one taken after with another button pressed to show increased rate and only one rate bonus allowed at a time	A4 <i>Optional</i>
[Resource] storage upgrade is bought	Screenshots to show max [resource] capacity increased	A5
Worker for [resource] upgrade is bought, not enough housing space	Screenshot to show “Not enough housing space” in tutorial logs	A6
Worker for [resource] upgrade is bought, enough housing space	Screenshot to show increased [resource] rate	A7
Housing upgrade is bought	Screenshot to show housing number increased	A8
Science upgrade is bought, but not enough science points available	Screenshot to show “Not enough science to buy this” in tutorial logs	A9
Science upgrade for combat is bought, new combat upgrade button appears in combat tab	Screenshot to show new upgrade button	A10
Different upgrade types tabs selected	Screenshot to show only worker upgrades shown in workers tab	A11 <i>Optional</i>
Combat upgrade to increase block bought	Screenshot to show troops’ increased block	A12
Enough resources are available to buy upgrade	Screenshot to show upgrade button turning to “clickable” state	A13 <i>Optional</i>
Section B: Save/load system		
Game is auto-saved every 2 minutes	3 screenshots – first one taken when auto-save happens, second one taken 2 minutes later when second auto-save happens, third taken of auto-save file changed (see time stamp in file)	B1

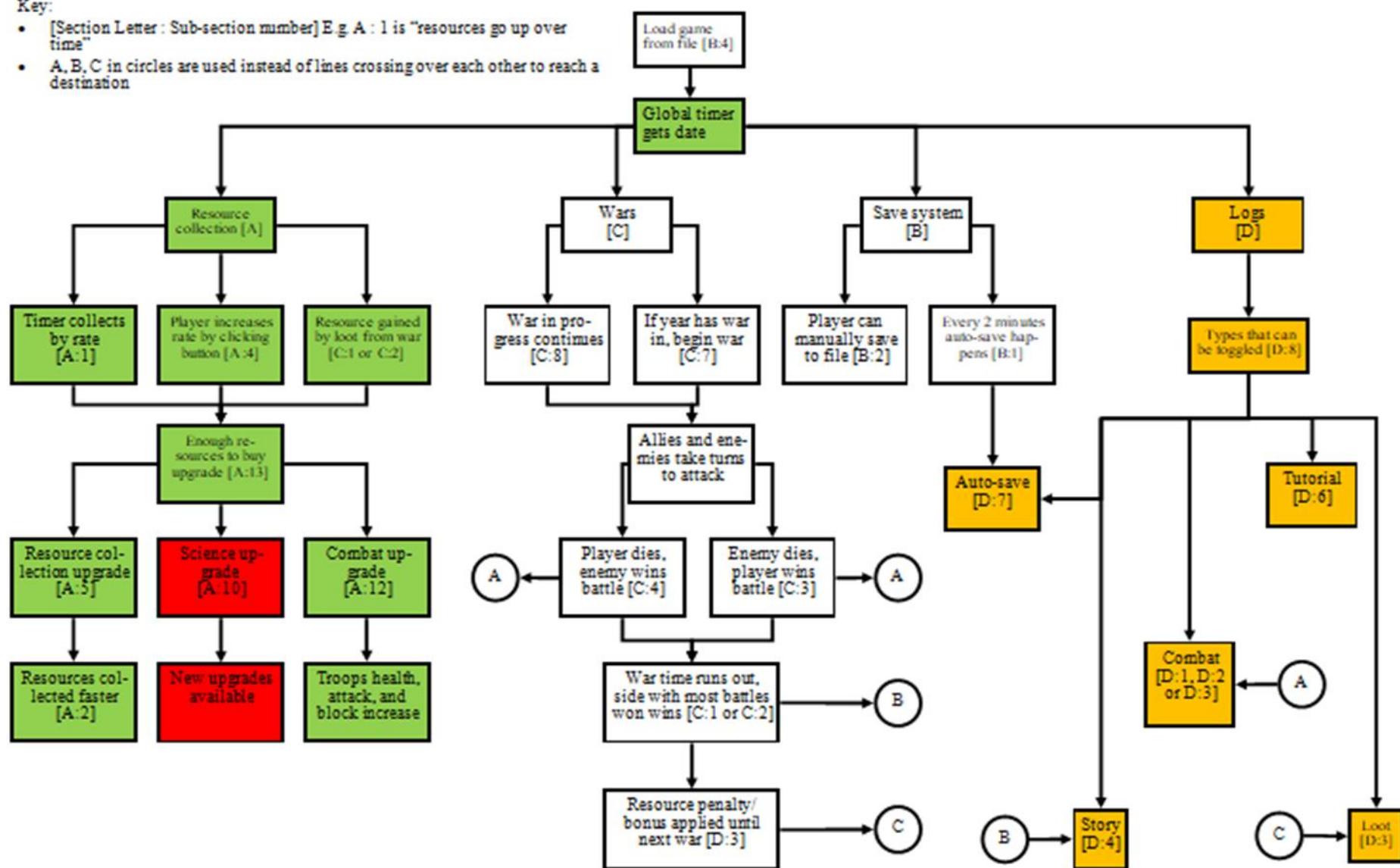
Player manually presses the save game button	Screenshot of new save file created	B2
Game closed	Screenshot of auto-save file changed	B3
Game loaded up, player selects save file to load, message box tells player how many resources were made when they were offline	Screenshot of before game closed, screenshot of after game reopened and save file loaded, screenshot of message box	B4
Section C: Combat/war system		
War is won by player	Screenshot of resources rate gain, and screenshot of grid reset	C1
War is lost by player	Screenshot of resources rate loss, and screenshot of grid reset	C2
Battle is won by player	Screenshot of player gaining green tile from enemy's red tile	C3
Battle is lost by player	Screenshot of enemy gaining red tile from player's green tile	C4
Player's army with 170 health and 50 block gets hit by 100 attack, so health goes down to 120	Screenshot before and after turn to show health/block/attack works	C5
Number of troops increased, total army health increases by (current upgrade of health) * number of new troops	Screenshot to show the stats of army increased	C6
New war started when right year reached, enemies have higher health/attack/block than in previous war	Screenshot to show stats in old war, and then stats in new	C7
Game loaded up, war in progress continues as before	Screenshot showing war before game close, after game close	C8
Section D: Logs system		
A battle is lost or won, show the message in the combat logs	Screenshot of combat logs	D1
A war is lost or won, show the statistics in the combat logs	Screenshot of combat logs	D2 <i>Optional</i>
A war is lost or won, logs show resource penalty/bonus until next war	Screenshot of loot logs	D3
A tech upgrade is made, and story progresses, show message in logs	Screenshot of story logs	D4
A new year begins, print new year in logs	Screenshot of story logs	D5
Player battles for first time, buys first upgrade or reaches first resource milestone, print	Screenshot of tutorial logs	D6

various tips or explanations in tutorial logs		
Game is auto-saved or manually saved by user, show message in auto-saves logs	Screenshot of auto-save logs	D7
A log type button is clicked to toggle it off, button changes from green to red and those logs stop showing	Screenshot of log toggle buttons and logs to show right logs toggled off	D8
The clear logs button is clicked, all logs toggled on cleared	Screenshot of nothing in logs	D9
Save logs to file button is clicked, all logs saved to a text file	Screenshot of file it saved to	D10 <i>Optional</i>
Section E: Usability		
The game balanced so that it can't be progressed really quickly (amount upgrades cost, how much the affect various parts of the game etc.)	Screenshots of player being able to keep up with enemies' combat stats, to show balanced progression through time	E1 <i>Optional</i>
Intuitive and easy to use menu	All of the game menus can be traversed within 2 clicks of the main game screen. Screenshots showing each menu	E2
Game does not have low framerate	Screenshot of CPU load reduction when game closed	E3
Big buttons, some colour coding to show toggles and being able to be clicked	Screenshots of buttons changing colour/position when various buttons pressed	E4 <i>Optional</i>

I have managed to implement A4, A11, A13 & E4 – all optional success criteria, however I have not been able to implement the milestone system A3, or the showing of new upgrades based off certain events A10.

Key:

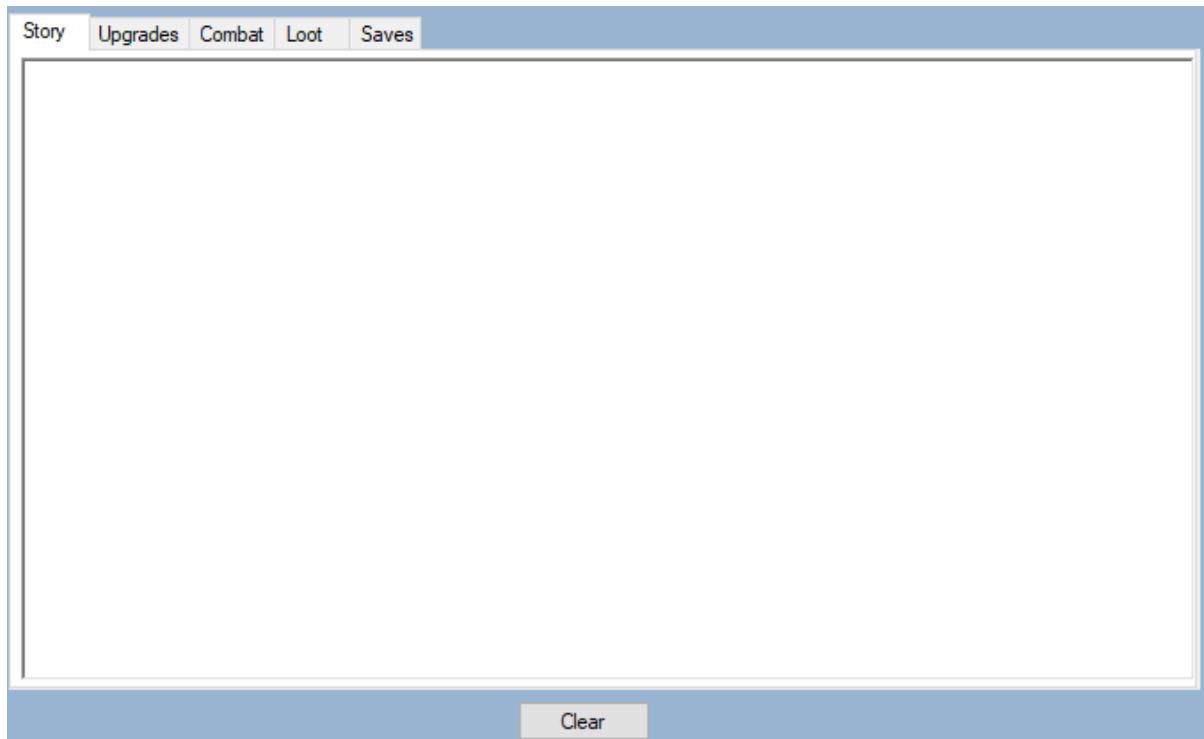
- [Section Letter : Sub-section number] E.g A : 1 is "resources go up over time"
- A, B, C in circles are used instead of lines crossing over each other to reach a destination



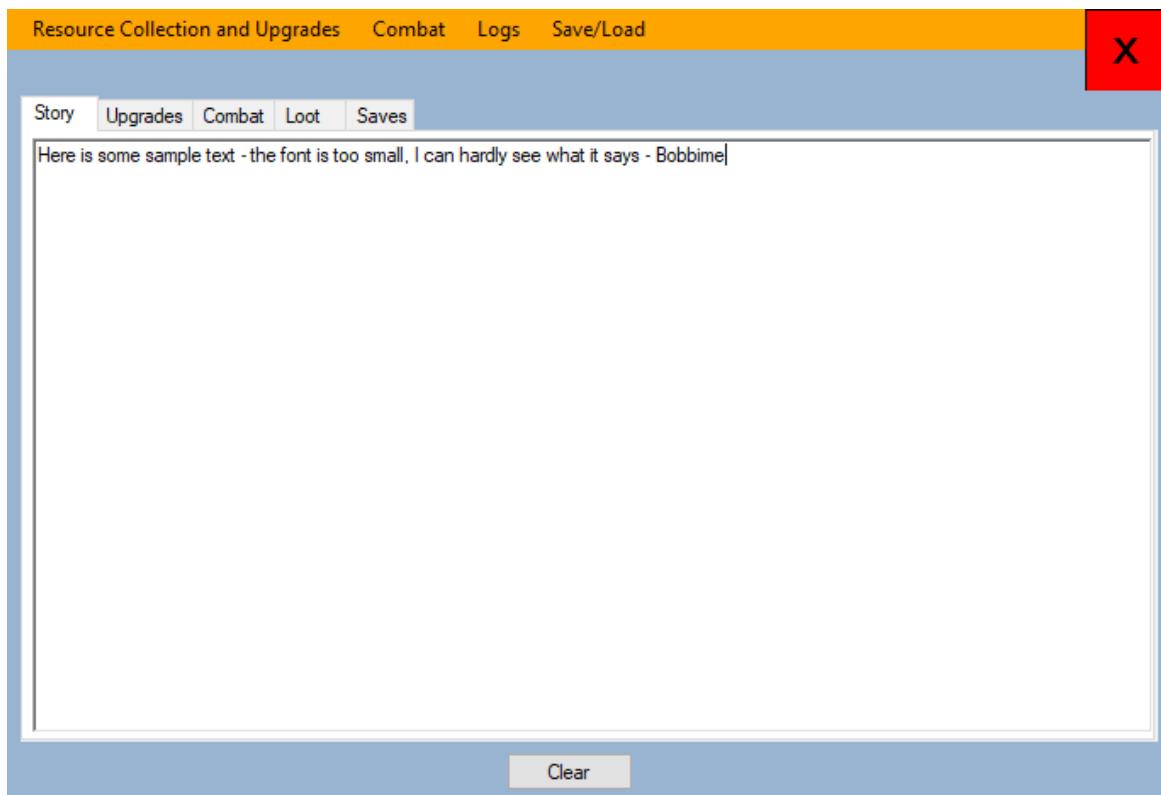
Logs GUI [D]

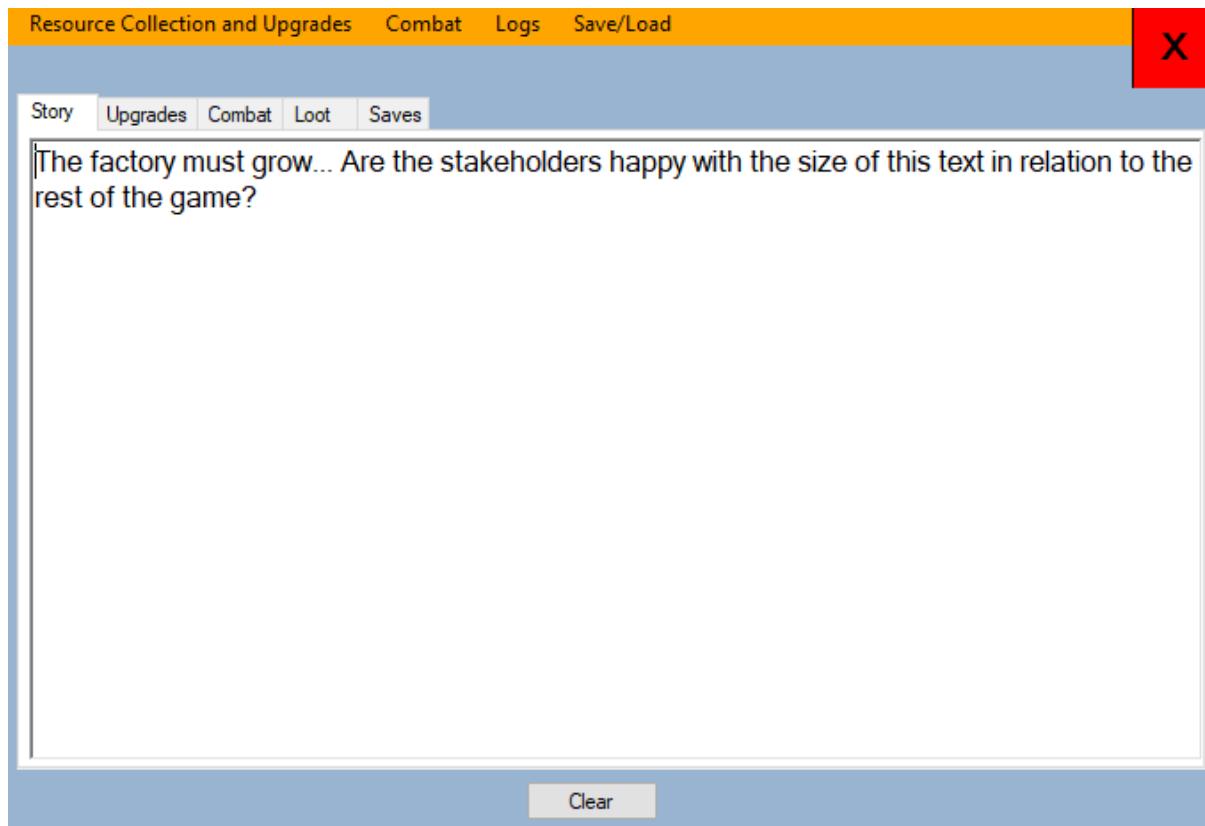
V1 27/10/19

Here is the UI for the logs taken directly from the design:



It sits directly on top of the resource collection and upgrades part (in the designer) so it looks like it is in the same location when the menu strip button is pressed to show it. Below is an image of sample text. It is far too small, so I will increase the font.





I sent the game to my stakeholders for them to test the font size on their computers, because they all have unique screen resolutions, potentially affecting the readability of the text. They all said the text is easy to read even from far back.

They also approve of the design. It is simple, and the clear button function for the selected tab is good.

Logs Code [D]

V1 01/11/19

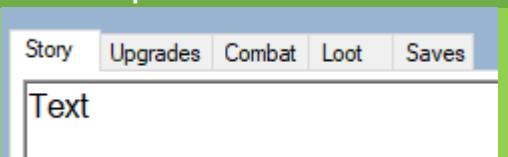
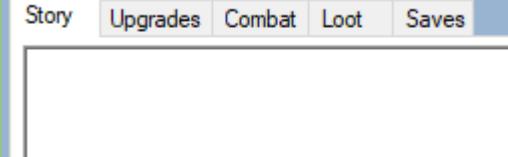
First is the clear button code.

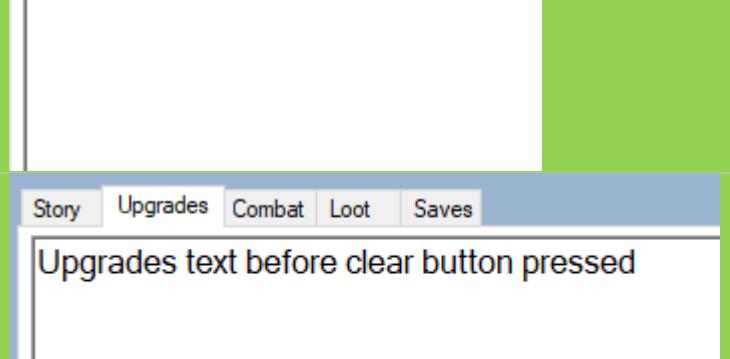
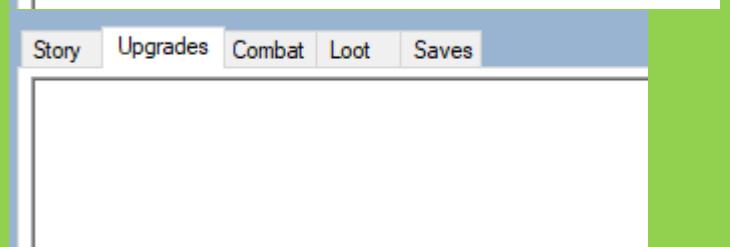
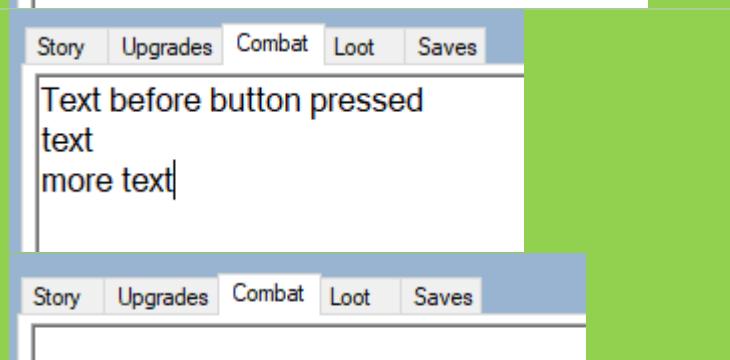
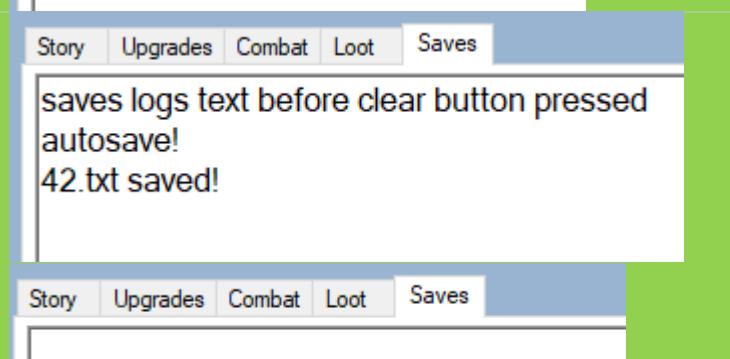
```

void clearText() {
    // Get index of selected tab
    int selectedTab = tabControl2.SelectedIndex;
    switch (selectedTab) {
        // Story
        case 0:
            rtxtStoryLogs.Text = "";
            break;
        // Upgrades
        case 1:
            rtxtUpgrades.Text = "";
            break;
        // Combat
        case 2:
            rtxtCombat.Text = "";
            break;
        // Loot
        case 3:
            rtxtLoot.Text = "";
            break;
        // Saves
        case 4:
            rtxtSaves.Text = "";
            break;
    }
}

private void btnClearLogs_Click(object sender, EventArgs e) {
    clearText();
}

```

Input	Expected Output	Actual Output
Some text is in the story logs, clear button pressed	Only text in the story logs is cleared	
Some text is in the loot logs, clear button pressed	Only text in the loot logs is cleared	

			
Some text is in the upgrades collection logs, clear button pressed	Only text in the upgrades collection logs is cleared		
Some text is in the combat logs, clear button pressed	Only text in the combat logs is cleared		
Some text is in the saves logs, clear button pressed	Only text in the saves logs is cleared		

I have decided that instead of having the logs event functions in the logs.cs file, I want to just add to the text straight from the other files, which would save having to essentially transfer the exact same

code over into another area but with more complications in getting it over, using a bunch of parameters in the function. E.g. When an upgrade cannot be bought, I want to just add the relevant text straight to the upgrades logs, rather than having to assign my wanted string as a parameter of a function, call a certain event and do a bunch of other complicated stuff which is completely unnecessary if I just do it on the spot.

So now I will add the code from each section that I have placeholders in:

Upgrades Logs

```
rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough wood for this upgrade");
rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough food for this upgrade");
rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": Troop number increased by 1, total is " + GlobalData.combatData[3][0].ToString());
rtxtCombat.Text += ("\n" + GlobalData.currentTime + " Troop health increased by 100, total is " + GlobalData.combatData[0][0].ToString());
rtxtCombat.Text += ("\n" + GlobalData.currentTime + " Troop block increased by 250, total is " + GlobalData.combatData[1][0].ToString());
rtxtStoryLogs.Text += ("\n" + "You have learned the art of aqueducts! These towering structures made of stone automate the carrying of water into your farms.");
rtxtStoryLogs.Text += ("\n" + "You have created the blueprints for stamp mills! These crush metal ore by pounding them to bits, making it faster to extract metal!");
rtxtStoryLogs.Text += ("\n" + "You have invented the trip-hammer! This will help you to shape your metals into more usable shapes!");
rtxtStoryLogs.Text += ("\n" + "You have stolen the secret of hushing from another civilisation! It allows you to use floods of water to reveal mineral veins.");
rtxtStoryLogs.Text += ("\n" + "You have earned yourself enough to construct a villa! This magnificent structure increases your housing space by 3 times!");
```

I can always add more story logs after certain wars etc, once the main structure of the game is done, to make the game more interesting.

```
rtxtStoryLogs.Text = "You feel an urge to grow the factory...";
```

The GlobalData.currentTime variable is a string that is the current time in UTC, taken from the system clock. The variable is updated every tick in the global timer:

```
public static string currentTime;
void currentTime() {
    GlobalData.currentTime = DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss");
}
void GlobalTimerTick(object sender, EventArgs e) {
    //Code
    currentTime();
}
```

V2 21/12/19

Now that most of the game has been developed, I will put in a bunch of the logs parts in order to meet various success criteria that required the existence of the logs.

Combat

```
rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": New war called " + GlobalData.warNames[i] + " started!");  
rtxtCombat.Text = ("\nYou have lost the battle! You are one step closer to losing the war.");  
rtxtCombat.Text = ("\nYou have won the battle! You are one step closer to winning the war.");  
rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": Draw");  
rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have lost the war, the enemy will get much stronger :(");  
rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have won the war! The enemy will not get that much stronger!");  
rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have gained " + (MULTIPLIERDRAW * GlobalData.warNumber).ToString() + " of each resource + " + (MULTIPLIERWAR * GlobalData.warNumber).ToString() + " science!");  
rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have lost " + randomNum.ToString() + " troops to the enemy, but gained " + (MULTIPLIERLOSS * GlobalData.warNumber).ToString() + " of each resource + " + (MULTIPLIERWAR * GlobalData.warNumber).ToString() + " science!");  
rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have gained half of the enemy's troops, and gained " + (MULTIPLIERWIN * GlobalData.warNumber).ToString() + " of each resource + " + (MULTIPLIERWARWIN * GlobalData.warNumber).ToString() + " science!");
```

File Handling

```
rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": The game has been autosaved!");  
rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": " + selectedFile + ".txt has been loaded.");  
rtxtSaves.Text += ("\n" + ex.Message);  
rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": File has been saved to " + selectedFile);  
rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": " + selectedFile + " has been deleted permanently.");
```

Review 2

- Green – Feature finished (does not matter if there is a minor error or change)
- Orange – Feature about to be worked on next
- Red – Feature skipped/not going to be worked on

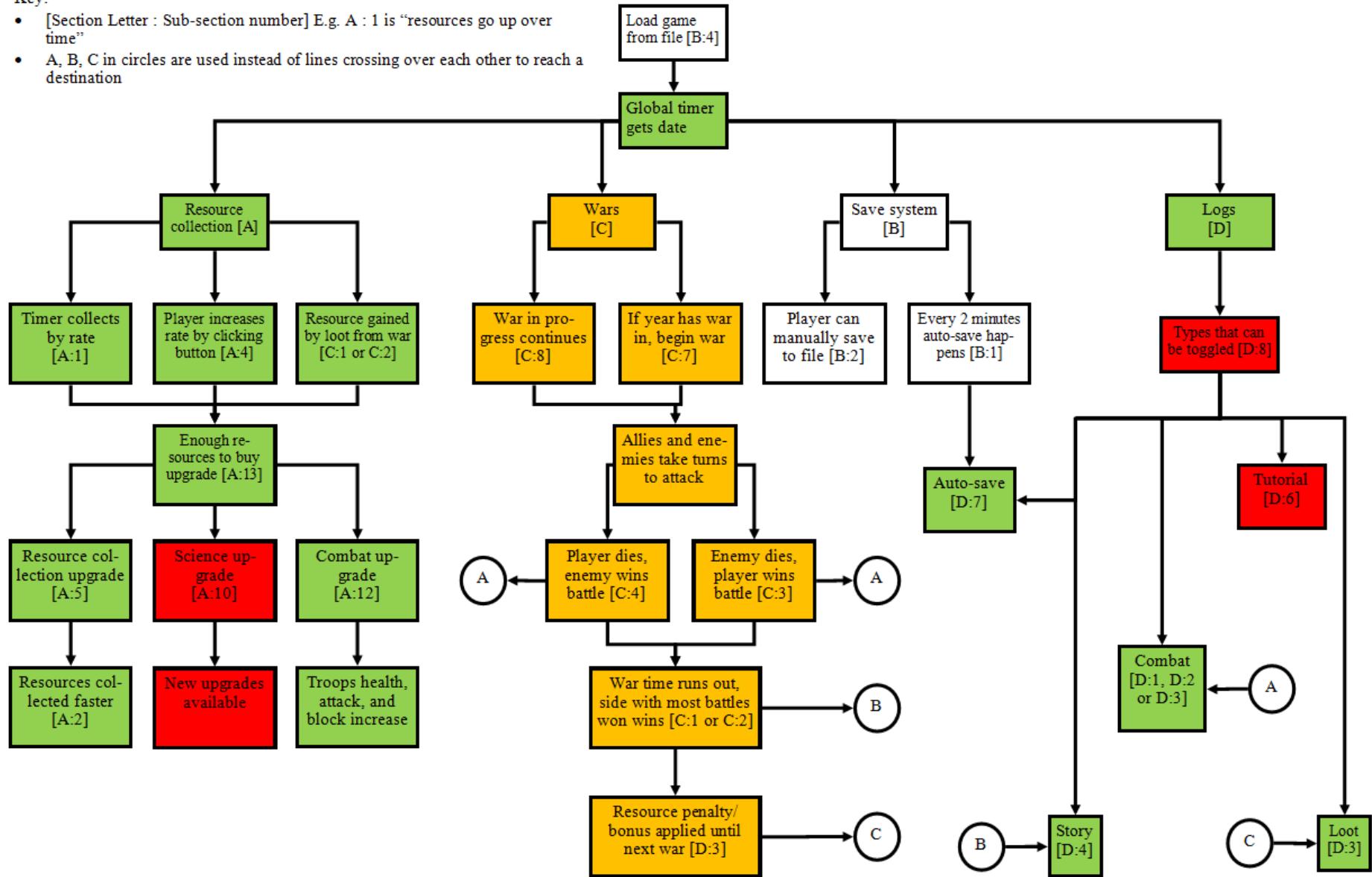
Criteria	How to evidence criteria being met	Section Code
Section A: Resources/upgrades system		
Resources go up over time	2 screenshots – first one taken before, second one taken after to show progress with resources	A1
Resource rate increased through upgrade	Screenshot of increased rate	A2
Resources reach “milestone” where the number shortens	Screenshot of 4,000 resource shown has 4k resource	A3 <i>Optional</i>
Button to manually increase rate of resource collection increased, button turns brown, all other buttons turn grey and toggle off, that resource rate increases by 10%	2 screenshots – first one taken before with one button pressed, second one taken after with another button pressed to show increased rate and only one rate bonus allowed at a time	A4 <i>Optional</i>
[Resource] storage upgrade is bought	Screenshots to show max [resource] capacity increased	A5
Worker for [resource] upgrade is bought, not enough housing space	Screenshot to show “Not enough housing space” in tutorial logs	A6
Worker for [resource] upgrade is bought, enough housing space	Screenshot to show increased [resource] rate	A7
Housing upgrade is bought	Screenshot to show housing number increased	A8
Science upgrade is bought, but not enough science points available	Screenshot to show “Not enough science to buy this” in tutorial logs	A9
Science upgrade for combat is bought, new combat upgrade button appears in combat tab	Screenshot to show new upgrade button	A10
Different upgrade types tabs selected	Screenshot to show only worker upgrades shown in workers tab	A11 <i>Optional</i>
Combat upgrade to increase block bought	Screenshot to show troops’ increased block	A12
Enough resources are available to buy upgrade	Screenshot to show upgrade button turning to “clickable” state	A13 <i>Optional</i>
Section B: Save/load system		
Game is auto-saved every 2 minutes	3 screenshots – first one taken when auto-save happens, second one taken 2 minutes later when second auto-save happens, third taken of auto-save file changed (see time stamp in file)	B1

Player manually presses the save game button	Screenshot of new save file created	B2
Game closed	Screenshot of auto-save file changed	B3
Game loaded up, player selects save file to load, message box tells player how many resources were made when they were offline	Screenshot of before game closed, screenshot of after game reopened and save file loaded, screenshot of message box	B4
Section C: Combat/war system		
War is won by player	Screenshot of resources rate gain, and screenshot of grid reset	C1
War is lost by player	Screenshot of resources rate loss, and screenshot of grid reset	C2
Battle is won by player	Screenshot of player gaining green tile from enemy's red tile	C3
Battle is lost by player	Screenshot of enemy gaining red tile from player's green tile	C4
Player's army with 170 health and 50 block gets hit by 100 attack, so health goes down to 120	Screenshot before and after turn to show health/block/attack works	C5
Number of troops increased, total army health increases by (current upgrade of health) * number of new troops	Screenshot to show the stats of army increased	C6
New war started when right year reached, enemies have higher health/attack/block than in previous war	Screenshot to show stats in old war, and then stats in new	C7
Game loaded up, war in progress continues as before	Screenshot showing war before game close, after game close	C8
Section D: Logs system		
A battle is lost or won, show the message in the combat logs	Screenshot of combat logs	D1
A war is lost or won, show the statistics in the combat logs	Screenshot of combat logs	D2 <i>Optional</i>
A war is lost or won, logs show resource penalty/bonus until next war	Screenshot of loot logs	D3
A tech upgrade is made, and story progresses, show message in logs	Screenshot of story logs	D4
A new year begins, print new year in logs	Screenshot of story logs	D5
Player battles for first time, buys first upgrade or reaches first resource milestone, print	Screenshot of tutorial logs	D6

various tips or explanations in tutorial logs		
Game is auto-saved or manually saved by user, show message in auto-saves logs	Screenshot of auto-save logs	D7
A log type button is clicked to toggle it off, button changes from green to red and those logs stop showing	Screenshot of log toggle buttons and logs to show right logs toggled off	D8
The clear logs button is clicked, all logs toggled on cleared	Screenshot of nothing in logs	D9
Save logs to file button is clicked, all logs saved to a text file	Screenshot of file it saved to	D10 <i>Optional</i>
Section E: Usability		
The game balanced so that it can't be progressed really quickly (amount upgrades cost, how much the affect various parts of the game etc.)	Screenshots of player being able to keep up with enemies' combat stats, to show balanced progression through time	E1 <i>Optional</i>
Intuitive and easy to use menu	All of the game menus can be traversed within 2 clicks of the main game screen. Screenshots showing each menu	E2
Game does not have low framerate	Screenshot of CPU load reduction when game closed	E3
Big buttons, some colour coding to show toggles and being able to be clicked	Screenshots of buttons changing colour/position when various buttons pressed	E4 <i>Optional</i>

Key:

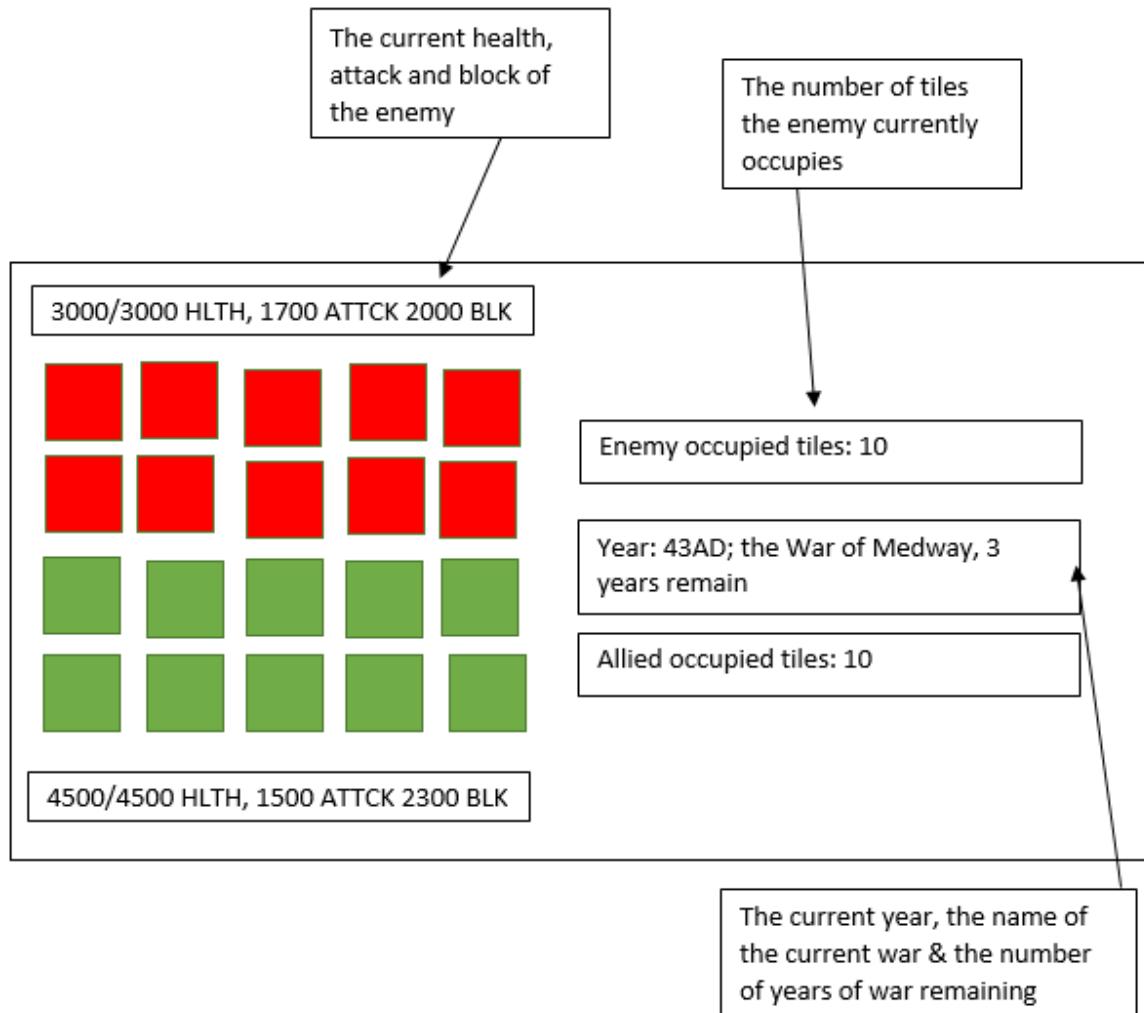
- [Section Letter : Sub-section number] E.g. A : 1 is “resources go up over time”
- A, B, C in circles are used instead of lines crossing over each other to reach a destination

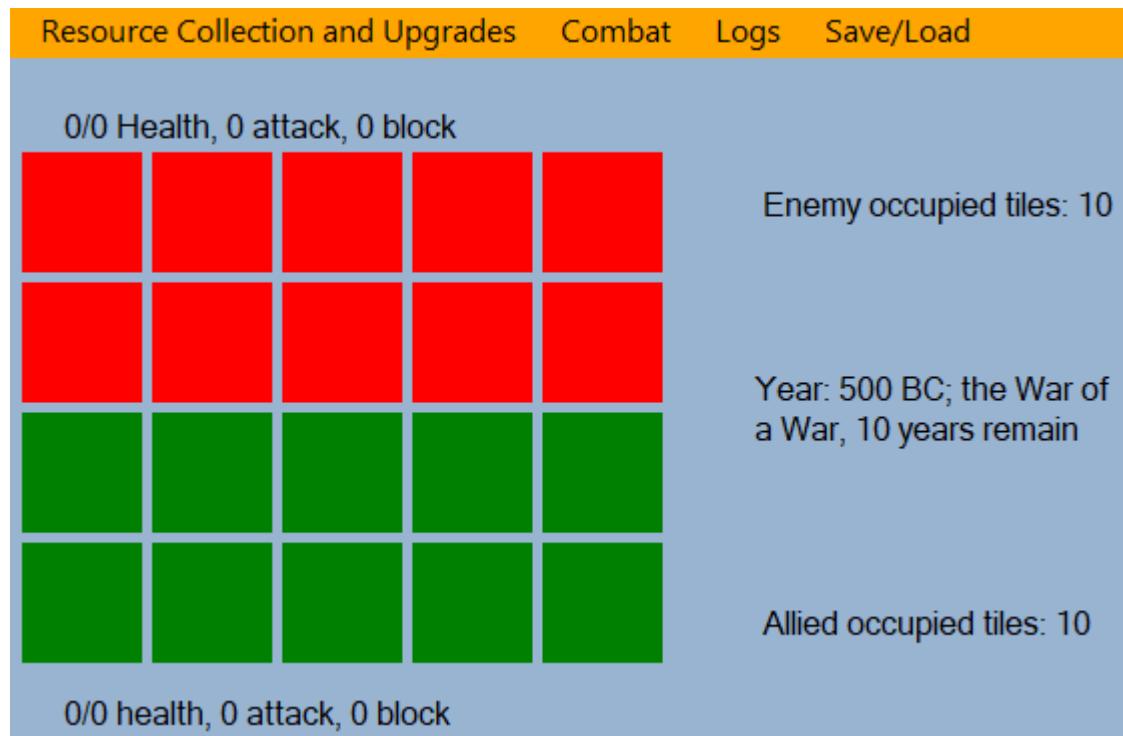


Combat GUI [C]

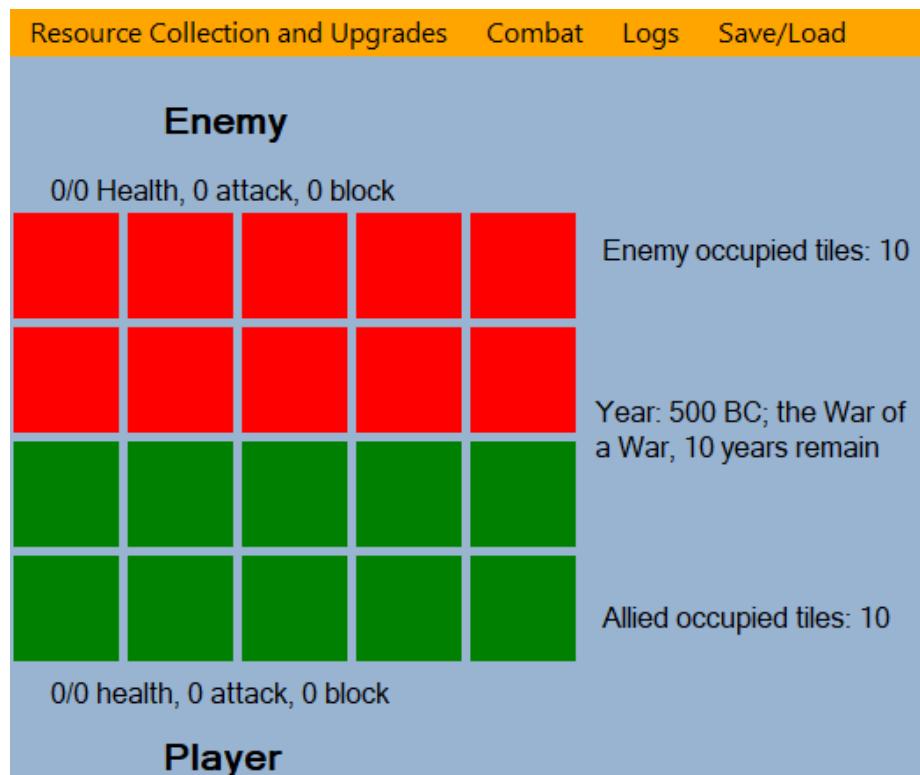
V1 02/11/19

Again, here is the GUI I built in-game that is exactly from the designs:



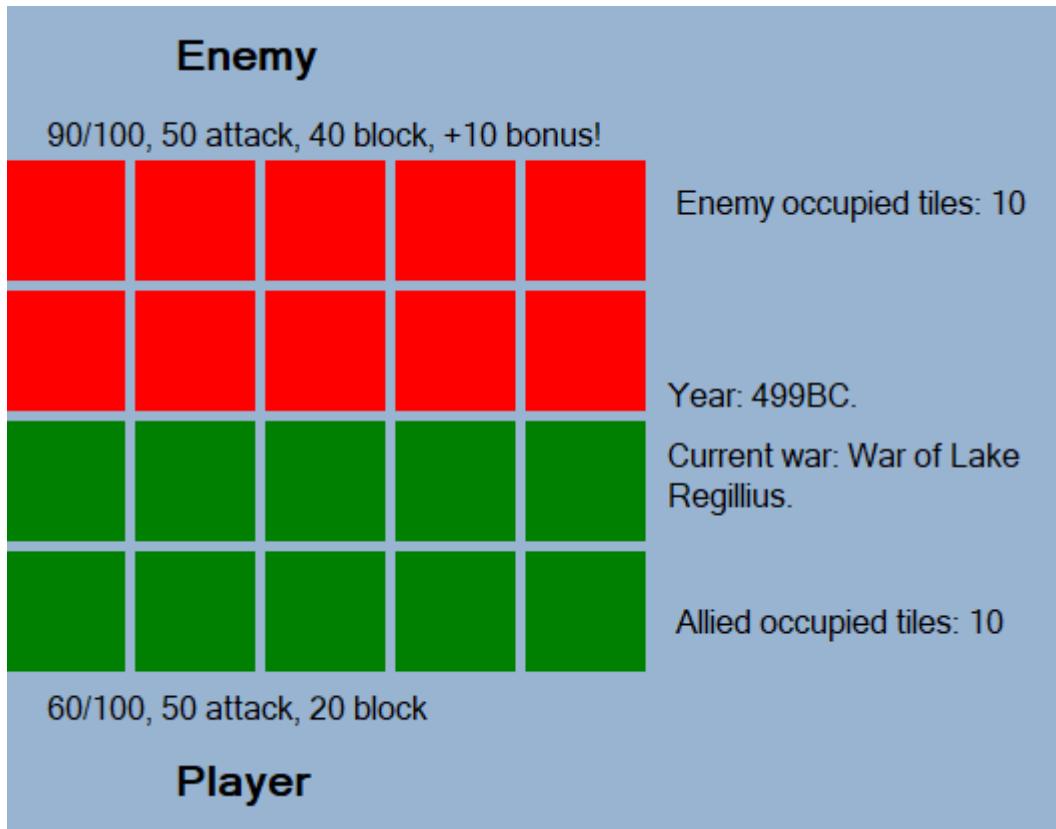


My stakeholders were unsure of which side was which, so I added labels on the top and bottom showing which side is which:



V2 27/11/19

When working general program testing, I noticed that I needed to make the label that shows the current year and war update constantly, rather than before when it only updated when a new war started. However, this caused the current war text to disappear, so to keep things simple I have opted to just split the label into 2: One showing the current year, and the next showing the current war. So now it looks like this:



Combat Code [C]

V1 07/11/19-18/11/19

The first thing I need to code is a year counter so that I can display it on the label that shows the year and the current war. This was not in my design because I didn't realise I needed it. I put it in my resourceCollection.cs file underneath the global timer tick function so that I could easily access it from directly underneath – the yearCounter() function is also run every tick from the global timer.

```
void yearCounter() {
    // If tick counter has reached a multiple of 60
    if ((GlobalData.tickCounter % 60) == 0) {
        // Increase the year counter
        GlobalData.year = 1;
        // If the year is negative
        if (GlobalData.year < 0) {
            // Get the positive year version
            GlobalData.actualYear = (GlobalData.year * 2);
            // Set the era to BC
            GlobalData.era = "BC";
        } else {
            GlobalData.actualYear = GlobalData.year;
            // Set the era to AD
            GlobalData.era = "AD";
        }
    }
}
```

There are no problems with this.

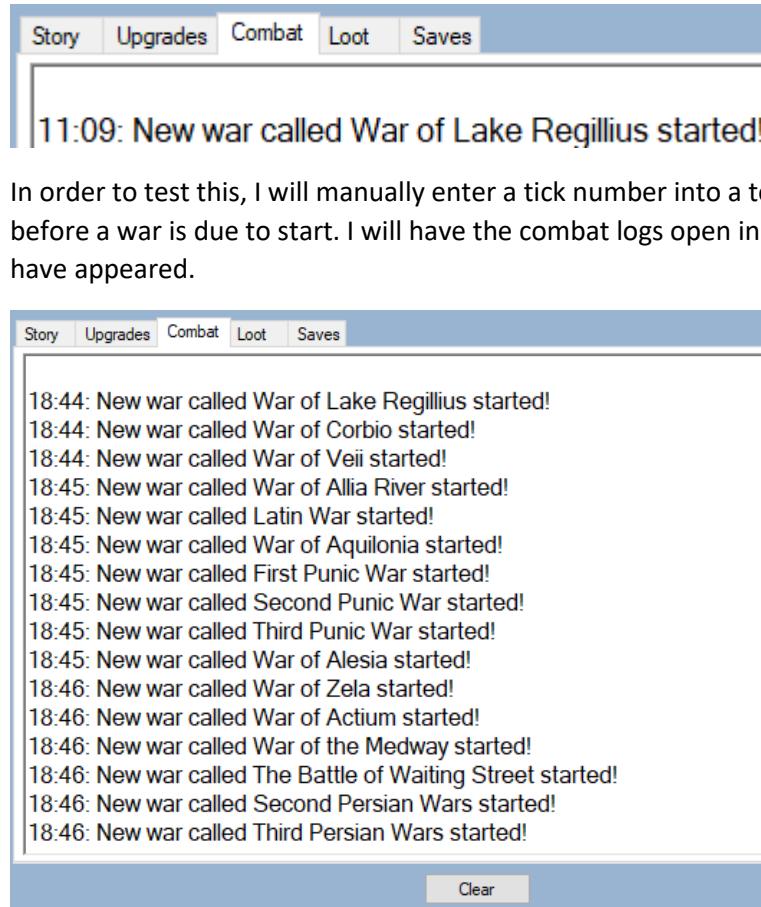
Next, the startWar() function which is called every tick in order to check if it is the correct year to start a war.

```
void startWar() {
    // Loop number of wars
    for (int i = 0; i < GlobalData.warTimes.Length; i++) {
        // If the current tick is equivalent to the current war time
        if (GlobalData.tickCounter == GlobalData.warTimes[i]) {
            // New war starts
            GlobalData.currentWar = GlobalData.warTimes[i];
            GlobalData.warNumber++;
            // Print this in logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": New war called " + GlobalData.warNames[i] + " started!");
            lblWarDescription.Text = ("Year: " +
GlobalData.actualYear.ToString() + GlobalData.era + ", " + GlobalData.warNames[i] +
".");
            // Reset the board
            resetBoard();
            // Start the combat timer
            combatTimer.Start();
        }
    }
}
```

The lblWarDescription looks like this after it has been changed by the function:

Year: 500BC, War of Lake Regillius.

The logs part looks like this:



```
public static int[] warTimes = new int[]{  

    300, 3240, 6000, 6800, 9600, 12300,  

    14160, 16920, 21000, 22200, 27540, 28200,  

    28920, 32580, 33660, 39660, 47760, 50640  

}; // Measured in tick numbers (see tickCounter), each war lasts 10 minutes (600)  

public static int currentWar = 0;  

public static int warNumber = 0;  

public static int year = -500;  

public static int actualYear = 500;  

public static string era = "BC";  

public static string[] warNames = new string[18] {  

    "War of Lake Regillius", "War of Corbio", "War of Veii", "War of Allia  

    River", "Latin War", "War of Aquilonia",  

    "First Punic War", "Second Punic War", "Third Punic War", "Achaean War",  

    "War of Alesia", "War of Zela",  

    "War of Actium", "War of the Medway", "The Battle of Waiting Street", "First  

    Persian Wars", "Second Persian Wars", "Third Persian Wars"  

};
```

Above are the variables, so you can see which wars are which.

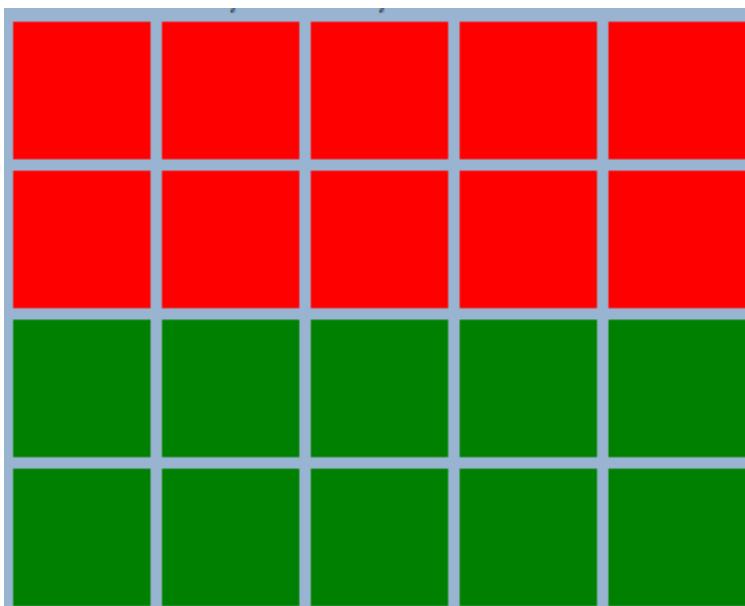
My stakeholders have noted that perhaps these logs should go into the story section instead, because it goes along with that is happening as the years go by, which is a fair point. So I have now moved these logs to the story section, but the test was a success.

The reset board function is called when a new war starts. This is because the board would have been previously shown from the previous wars and the values will have needed to be reset.

```
void resetBoard() {
    // Setup the graphics and brushes
    Bitmap bitmap = new Bitmap(1000, 1000);
    Graphics GFX = Graphics.FromImage(bitmap);
    SolidBrush greenBrush = new SolidBrush(Color.Green);
    SolidBrush redBrush = new SolidBrush(Color.Red);

    // Loop through each column (all are same)
    for (int x = 0; x < columns; x++) {
        // Loop through first 2 rows
        for (int y = 0; y < rows; y++) {
            // Paint the red tiles
            GFX.FillRectangle(redBrush, x * 65, y * 65, 60, 60);
        }
        // Loop through last 2 rows
        for (int y = 2; y < rows; y++) {
            // Paint the green tiles
            GFX.FillRectangle(greenBrush, x * 65, y * 65, 60, 60);
        }
    }
    // Show the image in the picture box
    pbxGrid.Image = bitmap;
}
```

This function draws this:



Finally, the combat timer is turned on. This ticks every 1000ms (1 second), at the same time as the global timer. This only runs when a war is on.

```
private void combatTimer_Tick(object sender, EventArgs e) {
    calculateAttack();
    winWarCheck();
}
```

```
void calculateAttack() {
    // If it is the player's turn
    if (playerTurn) {
        playerTurn = false;
        // Work out how much damage the player does
        int blockedPlayerDamage = playerDamage - enemyBlock;
        newEnemyHealth = enemyHealth - blockedPlayerDamage;
        // 0/0 Health, 0 attack, 0 block
        lblEnemyStats.Text = String.Format("{0}/{1}, {2} attack, {3} block",
newEnemyHealth, enemyHealth, enemyDamage, enemyBlock);
        lblPlayerStats.Text = String.Format("{0}/{1}, {2} attack, {3} block",
newPlayerHealth, playerHealth, playerDamage, playerBlock);

        // If the enemy health is 0 or less
        int checkSquare = grid[0, 0];
        if (newEnemyHealth <= 0) {
            // Give the player a tile
            playerTiles += 1;
            int enemyTiles = 20 - playerTiles;
            lblEnemyTiles.Text = ("Enemy tiles: " + enemyTiles);
            lblPlayerTiles.Text = ("Allied tiles: " + playerTiles);
            // Loop through each column
            for (int x = 0; x < columns; x++) {
                // Loop through each row
                for (int y = 0; y < rows; y++) {
                    // If the grid at the current point is not the
                    // same as the top left grid point
                    if (grid[x, y] != checkSquare) {
                        // Give the next to tile the enemy
                        grid[y + 1, x] = 1;
                    } else {
                }
            }
        }
    }
}
```

```
        // Give the next tile to the player
        grid[y + 1, x] = 0;
    }
}
updateBoard();
// Reset player and enemy health back to normal
newEnemyHealth = enemyHealth;
newPlayerHealth = playerHealth;
}
} else {
    playerTurn = true;
    // Work out how much damage the enemy does
    int blockedEnemyDamage = enemyDamage - playerBlock;
    newPlayerHealth = playerHealth - blockedEnemyDamage;
    // 0/0 Health, 0 attack, 0 block
    lblEnemyStats.Text = String.Format("{0}/{1}, {2} attack, {3} block",
newEnemyHealth, enemyHealth, enemyDamage, enemyBlock);
    lblPlayerStats.Text = String.Format("{0}/{1}, {2} attack, {3} block",
newPlayerHealth, playerHealth, playerDamage, playerBlock);

    // If the player health is 0 or less
    int checkSquare = grid[0, 0];
    if (newPlayerHealth <= 0) {
        // Give the enemy the tile
        playerTiles -= 1;
        int enemyTiles = 20 - playerTiles;
        lblEnemyTiles.Text = ("Enemy tiles: " + enemyTiles);
        lblPlayerTiles.Text = ("Allied tiles: " + playerTiles);
        // Loop through each column
        for (int x = 0; x < columns; x++) {
            // Loop through each row
            for (int y = 0; y < rows; y++) {
                // If the grid at the current point is not the
                // same as the top left grid point
                if (grid[x, y] != checkSquare) {
                    // Give the next tile to the enemy
                    grid[y + 1, x] = 1;
                } else {
                    // Give the next tile to the player
                    grid[y + 1, x] = 0;
                }
            }
        }
        updateBoard();
        // Reset player and enemy health to normal
        newPlayerHealth = playerHealth;
        newEnemyHealth = enemyHealth;
    }
}
}
```

I will show all the variables I have next so that you can see which variable does what:

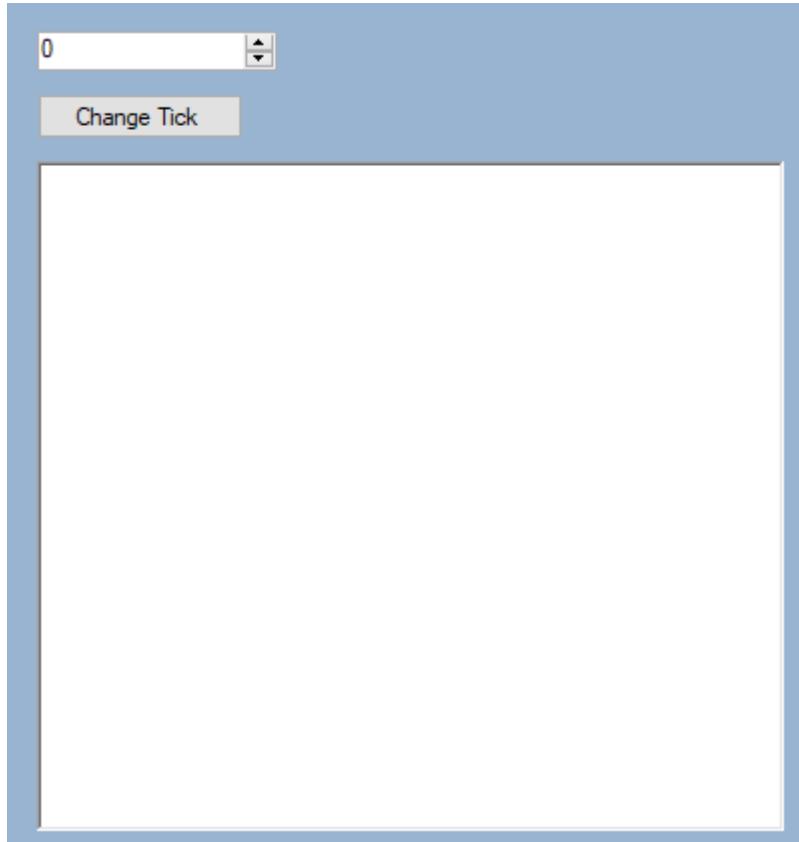
```
int playerHealth = GlobalData.combatData[0][0] * GlobalData.combatData[3][0];
int playerBlock = GlobalData.combatData[1][0] * GlobalData.combatData[3][0];
int playerDamage = GlobalData.combatData[2][0] * GlobalData.combatData[3][0];

int enemyHealth = GlobalData.combatData[0][1] * GlobalData.combatData[3][1];
int enemyBlock = GlobalData.combatData[1][1] * GlobalData.combatData[3][1];
int enemyDamage = GlobalData.combatData[2][1] * GlobalData.combatData[3][1];

int newEnemyHealth = 100;
int newPlayerHealth = 30;

bool playerTurn = true;
Random rng = new Random();
int playerTiles = 10;
int offset = 0;
bool allEqual = false;
const int rows = 4;
const int columns = 5;
int[,] grid = new int[4, 5]
{
    {0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0},
    {1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1}
}; //Where 0 is red and 1 is green (enemy and player occupied respectively)
```

To test this function without the other parts (like updateBoard() and warWinCheck()), I will comment out the function calls, and use a temporary text box to capture all of the values that affect other parts of the program or outputs:

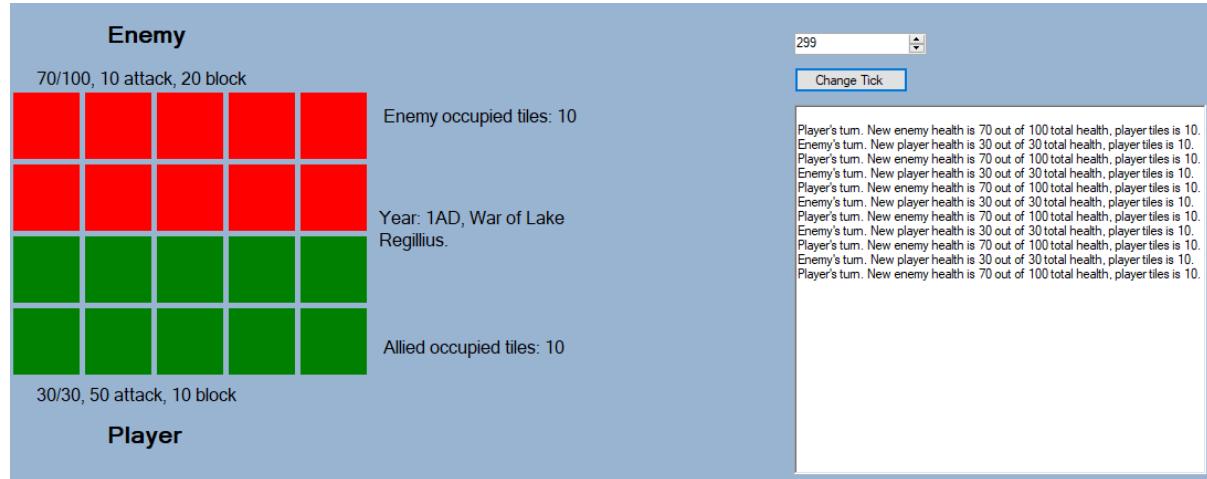


```

rtxtTempOutput.Text += String.Format("\nPlayer's turn. New enemy health is {0} out
of {1} total health, player tiles is {2}.", newEnemyHealth, enemyHealth,
playerTiles);

rtxtTempOutput.Text += String.Format("\nEnemy's turn. New player health is {0} out
of {1} total health, player tiles is {2}.", newPlayerHealth, playerHealth,
playerTiles);

```



The first obvious issue to notice here is that the player and enemy's health reset back to their max after every other turn. This is because I set the new health to equal the max health at the start of every turn, which of course resets it.

In order to fix this, I declare the local stats variables when the game first runs, and assign their new values from the GlobalData variables every time a war starts. Then, instead of setting the new enemyHealth value to the max every turn, it does the calculations based off its previous amount (unless it has been reset if a battle is won/lost).

```

// Run when game first loaded
int maxPlayerHealth;
int playerBlock;
int playerDamage;
int maxEnemyHealth;
int enemyBlock;
int enemyDamage;

// Inside startWar()
maxPlayerHealth = GlobalData.combatData[0][0] * GlobalData.combatData[3][0];
playerBlock = GlobalData.combatData[1][0] * GlobalData.combatData[3][0];
playerDamage = GlobalData.combatData[2][0] * GlobalData.combatData[3][0];
maxEnemyHealth = GlobalData.combatData[0][1] * GlobalData.combatData[3][1];
enemyBlock = GlobalData.combatData[1][1] * GlobalData.combatData[3][1];
enemyDamage = GlobalData.combatData[2][1] * GlobalData.combatData[3][1];

// Inside calculateAttack() in the player turn section
int blockedPlayerDamage = playerDamage - enemyBlock;
enemyHealth -= blockedPlayerDamage;

enemyHealth = maxEnemyHealth;
playerHealth = maxPlayerHealth;

```

The second noticeable issue is that for some reason, the year is set to 1 AD instead of 500 BC when the ticks are skipped, but this will not happen when the ticks are incremented by 1 instead of jumping by 300, as we have tested and seen when I first implemented it.

The final issue is less noticeable from screenshots – I don't like how each turn lasts 1 second, as it feels far too short. To solve this, I could increase the delay of the combat timer to 5000 or 1000ms for 5 or 10 seconds between each turn respectively. However, this would then affect how often I call the winWarCheck() function, which may cause the second

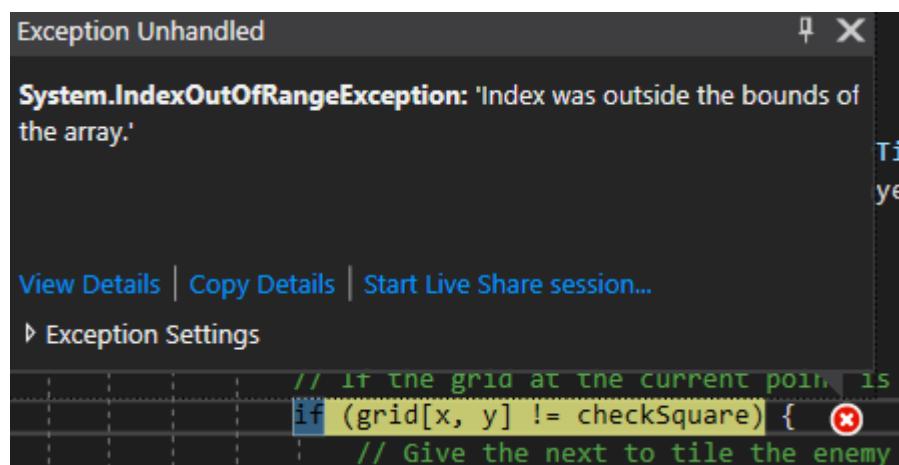
```
if (allEqual || (GlobalData.tickCounter + 600) >= (GlobalData.currentWar + 600))
```

condition to not ever be called. This is because if the tick counter is for example 343 when the war starts, and the winWarCheck() is subsequently called every 10 seconds, the value of GlobalData.tickCounter + 600 will not be larger than the GlobalData.currentWar + 600 value at the correct time. Therefore, to get around this I will just have a new variable in the combat timer, which resets back to 0 when a new war starts.

```
warTicks += combatTimer.Interval / 1000; // Inside combat timer  
warTicks = 0; // Inside starWar()  
if (allEqual || warTicks == 600) { // Inside winWarCheck()
```

I let my stakeholders try 10 seconds, but they said it was too long, so I changed it to 5 seconds – this is still okay because 5 is a multiple for 600 so warTicks will still reach 600 at some point (if allowed to).

Now that I have made these changes, I have ran the program to check for any crashes.



However, after only a small number of turns, the game crashes due to this line. The reason was because the x and y were swapped around when calling elements from the grid array, the for loop was cooling elements higher than those that existed so caused an outside the bounds of array crash. This was of course easily fixed by swapping the x and y.

Next, I will build the winWarCheck() and updateBoard() systems, again based off my designs. The updateBoard() procedure will use the array of green and red brushes based on the grid array to show the correct tiles.

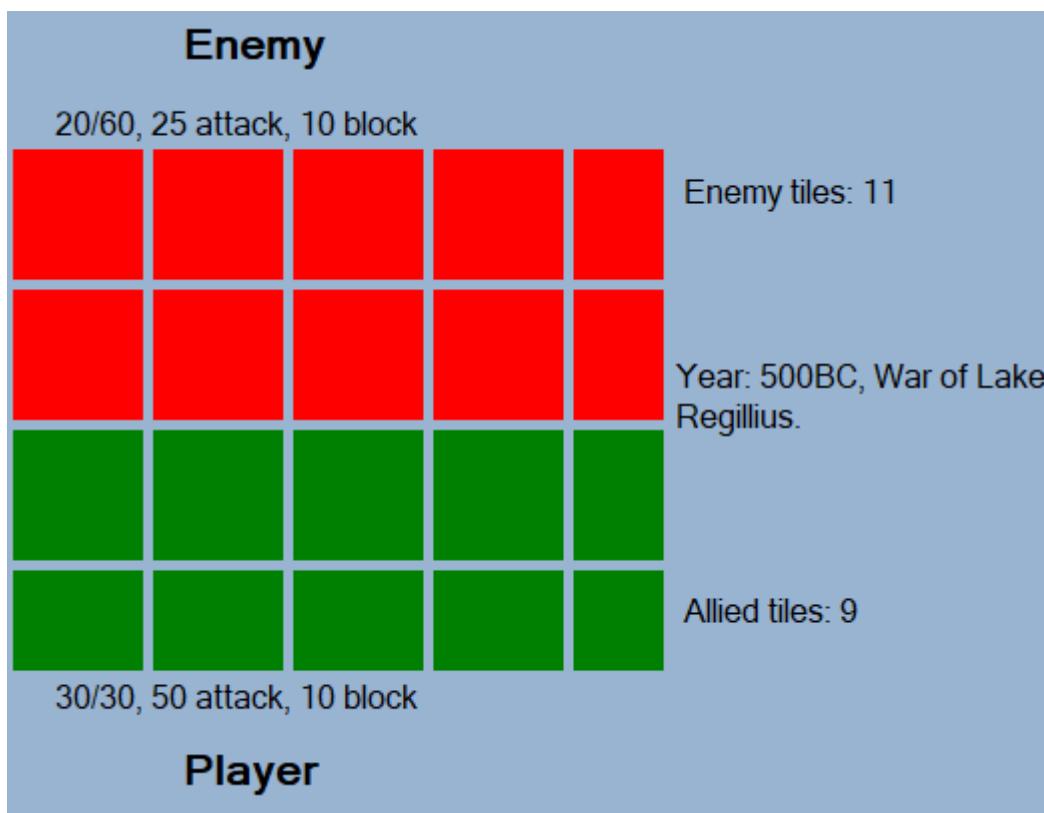
```

void updateBoard() {
    // Setup graphics and brushes
    Bitmap bitmap = new Bitmap(1000, 1000);
    Graphics GFX = Graphics.FromImage(bitmap);
    SolidBrush[] brushes = { new SolidBrush(Color.Red), new
SolidBrush(Color.Green) };
    int checksquare = grid[0, 0];
    const int gridSize = 70;

    // Loop through each column
    for (int x = 0; x < columns; x++) {
        // Loop through each row
        for (int y = 0; y < rows; y++) {
            // Redraw the grid according to what the values in the grid
            variable are
            GFX.FillRectangle(brushes[grid[y, x]], Rectangle.FromLTRB(x *
(gridSize), y * (gridSize), x * (gridSize) + 65, y * (gridSize) + 65));
            // If the current grid square is equal to the top left square
            if (grid[y, x] != checksquare) {
                // A side has won, as they own all squares
                allEqual = true;
            }
        }
    }
    pbxGrid.Image = bitmap;
}

```

Just to test it definitely works before I go onto the main testing, I ran the program where a battle is won quickly and thus I can check if the system works. It does not.



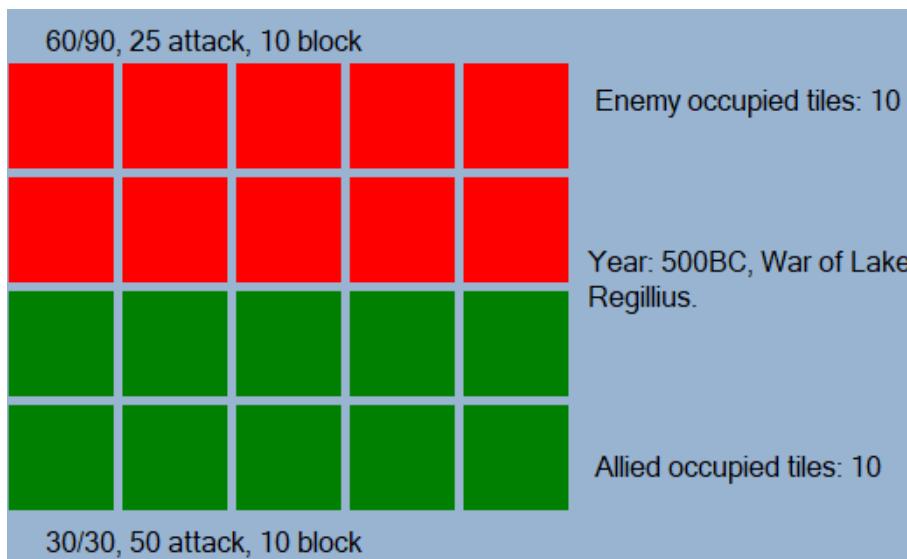
As you can see, the updated board is cut off and broken, thus something hasn't worked correctly when showing it. The issue is that in the `resetBoard()` sub, each tile size is 65x65, but in `updateBoard()` I currently have it set to 70 (on the constant `gridSize`). However, this then lead to the gaps between each tile disappearing, because they were set at 0 apart. The fix for this was to just change this line:

```
GFX.FillRectangle(brushes[grid[y, x]], Rectangle.FromLTRB(x * (gridSize), y * (gridSize), x * (gridSize) + 65, y * (gridSize) + 65));
```

To this:

```
GFX.FillRectangle(brushes[grid[y, x]], Rectangle.FromLTRB(x * (gridSize), y * (gridSize), x * (gridSize) + 60, y * (gridSize) + 60));
```

As you can see, all I had to do was change the bottom corners' positions to be 5 less than before. So now that a war starts and a turn has been made (thus calling the `updateBoard()` sub), the grid looks the same as the reset grid at the start of each war.



And if I manually change the grid 2d array so that the player has, say, 3 of the enemy's tiles, it now updates correctly.



However, there is now another issue that I have noticed. If I reset the grid 2d array back to default, run a war and a side wins, the board doesn't update correctly. In fact, it does not change at all. Now that I know the issue is not the updateBoard() sub, it must be the grid array not being updated correctly in calculateAttack(). More specifically, the part that must be wrong is this.

```
// Loop through each column
for (int x = 0; x < columns; x++) {
    // Loop through each row
    for (int y = 0; y < rows; y++) {
        // If the grid at the current point is not the same as the top left
        // grid point
        if (grid[y, x] != checkSquare) {
            // Give the next tile to the enemy
            grid[y, x] = 1;
        } else {
            // Give the next tile to the player
            grid[y, x] = 0;
        }
    }
}
```

It turns out that, of course, this never actually checks if the next tile is owned by the player or enemy, so actually does nothing (as seen). Here is the fix (I also made player and enemy tile constants to avoid the use of magic numbers):

```
int i = 19;
// While the current tile is owned by the player
while(grid[i / 5, i % 5] == PLAYER) {
    i--;
}
// Next tile must be next in line to be claimed, thus set this to player
grid[i / 5, i % 5] = PLAYER;
```

```
int i = 0;
// While the current tile is owned by the enemy
while (grid[i / 5, i % 5] == ENEMY) {
    i++;
}
// Next tile must be next in line to be claimed, thus set this to enemy
grid[i / 5, i % 5] = ENEMY;
```

So now, when a tile has been lost or gained by the player, the grid 2d array now updates correctly.

The next part to work on is winWarCheck(). Here is the code for it:

```
void winWarCheck() {
    // A side wins all tiles, or the war timer has run out
    if (allEqual || warTicks == 600) {
        // Loop through all columns
        for (int x = 0; x < columns; x++) {
            // Loop through all rows
            for (int y = 0; y < rows; y++) {
                // Getting the value if then offset to see who has all
                tiles
                offset += 1 - (2 * grid[y, x]);
            }
        }

        // If both sides have the same number of tiles
        if (offset == 0) {
            // Work out new enemy stats
            int randomNum = rng.Next(3, 5);
            for (int i = 0; i < 3; i++) {
                GlobalData.combatData[i][1] *= randomNum;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Print draw into logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": Draw");

            // Work out how much loot player receives
            calculateLoot(0);
            // If the enemy has more tiles than the player
        } else if (offset > 0) {
            // Work out new enemy stats
            int randomNum = rng.Next(3, 7);
            for (int i = 0; i < 3; i++) {
                GlobalData.combatData[i][1] *= randomNum;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Print enemy wins into logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have lost the war, the enemy will get much stronger :(");

            // Work out how much loot player receives
            calculateLoot(1);
            // If the player has more tiles than the enemy
        } else {
            // Work out new enemy stats
            int randomNum = rng.Next(3, 5);
            for (int i = 0; i < 3; i++) {
                GlobalData.combatData[i][1] *= randomNum;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Print player wins into logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have won the war! The enemy will not get that much stronger!");

            // Work out how much loot player receives
            calculateLoot(2);
        }
    }
    // Stop the combat timer until new war starts
    combatTimer.Stop();
}
```

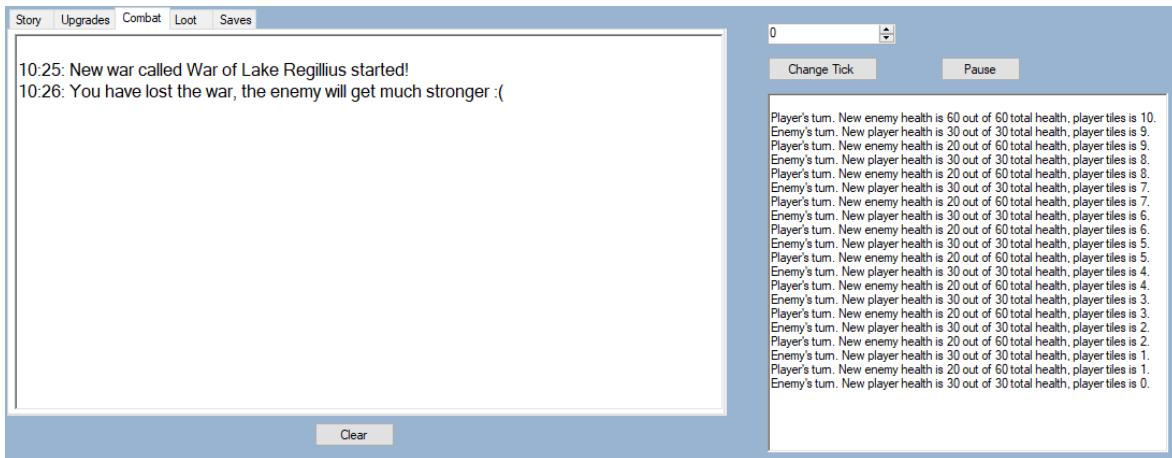
As you can see, there is already a bit of an issue here. Offset is a variable that works out how many tiles each side owns. Then, the value of offset is compared to and an outcome is taken based off the result of the comparisons. The issue here is that there will ALWAYS be an outcome. One statement checks if offset is equal to 0, one checks if it is more than 0, and another checks if it is less than 0. Of course, every number falls into one of those categories. So something isn't right there.

Whilst in the process of trying to work out how to fix this, I realised something – I have been overcomplicating the system far too much. I don't need to calculate the tile offset – and that doesn't work well when there is a draw. I realised I can just use the playerTiles variable to work out if I have enough tiles to complete a war (0 or 20 for enemy or player win respectively) and then 3 if statements to check which side has won or if it is a draw. I could also then remove this part from updateBoard():

```
if (grid[y, x] != checksquare) {  
    // A side has won, as they own all squares  
    allEqual = true;  
}
```

Now, winWarCheck() looks like this:

```
void winWarCheck() {
    // A side wins all tiles, or the war timer has run out
    if (playerTiles == 0 || playerTiles == 20 || warTicks == 600) {
        // If both sides have the same number of tiles
        if (playerTiles == 10) {
            // Work out new enemy stats
            int randomNum = rng.Next(3, 5);
            for (int i = 0; i < 3; i++) {
                GlobalData.combatData[i][1] *= randomNum;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Print draw into logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": Draw");
            // Work out how much loot player receives
            calculateLoot(0);
            // If the enemy has more tiles than the player
        } else if (playerTiles == 0) {
            // Work out new enemy stats
            int randomNum = rng.Next(3, 7);
            for (int i = 0; i < 3; i++) {
                GlobalData.combatData[i][1] *= randomNum;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Print enemy wins into logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have lost the war, the enemy will get much stronger :(");
            // Work out how much loot player receives
            calculateLoot(1);
            // If the player has more tiles than the enemy
        } else if (playerTiles == 20) {
            // Work out new enemy stats
            int randomNum = rng.Next(3, 5);
            for (int i = 0; i < 3; i++) {
                GlobalData.combatData[i][1] *= randomNum;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Print player wins into logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have won the war! The enemy will not get that much stronger!");
            // Work out how much loot player receives
            calculateLoot(2);
        }
        // Stop the combat timer until new war starts
        combatTimer.Stop();
    }
}
```



This solution actually works well, as you can see from the temporary output and the logs.

The penultimate part to address is calculateLoot, which takes the outcome from winWarCheck().

```

void calculateLoot(int outcome) {
    switch (outcome) {
        case 0:
            // Gets 200* war number for each resource
            const int multiplierDraw = 200;
            for (int i = 0; i < 4; i++) {
                GlobalData.resourcesData[0][i] += multiplierDraw *
GlobalData.warNumber;
            }
            // Gets 1000 * war number of science
            const int multiplierWar = 1000;
            GlobalData.scienceData += multiplierWar *
GlobalData.warNumber;
            // Print values to logs
            rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have gained " + (multiplierDraw * GlobalData.warNumber).ToString() + " of
each resource " + (multiplierWar * GlobalData.warNumber).ToString() + "
science!");
            break;
        case 1:
            // Loses 5-10 troops to the enemy
            int randomNum = rng.Next(5, 10);
            GlobalData.combatData[3][0] -= randomNum;
            if (GlobalData.combatData[3][0] < 1) {
                GlobalData.combatData[3][0] = 1;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Gets 500* war number of each resource
            const int multiplierLoss = 100;
            for (int i = 0; i < 4; i++) {
                GlobalData.resourcesData[0][i] += multiplierLoss *
GlobalData.warNumber;
            }
            // Gets 1000 * war number of science
            GlobalData.scienceData += multiplierWar *
GlobalData.warNumber;
            // Print values to logs
            rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have lost " + randomNum.ToString() + " troops to the enemy, but gained " +
(multiplierLoss * GlobalData.warNumber).ToString() + " of each resource " +
(multiplierWar * GlobalData.warNumber).ToString() + " science!");
            break;
    }
}

```

```

        case 2:
            // Gets half number of troops enemy had
            int noEnemyTroops = GlobalData.combatData[3][1];
            // If it is odd, take 1 then half it
            if (noEnemyTroops % 2 != 0) {
                noEnemyTroops -= 1;
            }
            GlobalData.combatData[3][0] += (noEnemyTroops / 2);
            // Gets 500* war number of each resource
            const int multiplierWin = 500;
            for (int i = 0; i < 4; i++) {
                GlobalData.resourcesData[0][i] += multiplierWin *
GlobalData.warNumber;
            }
            // Gets 5000 * war number of science
            const int multiplierWarWin = 5000;
            GlobalData.scienceData += multiplierWarWin *
GlobalData.warNumber;
            // Print values to logs
            rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() +
": You have gained half of the enemy's troops, and gained " + (multiplierWin *
GlobalData.warNumber).ToString() + " of each resource + " + (multiplierWarWin *
GlobalData.warNumber).ToString() + " science!");
            break;
        }
    }
}

```

Each outcome – win, loss or draw – is based off the loot that is in the design:

1. Loot (if they win)
 - a. Gets half of the number of troops as the enemy had along with more housing space to accommodate them
 - b. Gets 500 * war number of each wood, stone, metal & food.
2. Loot (if they lose)
 - a. Loses between 5 and 10 troops to the enemy, but keeps the housing space
 - b. Gets 100 * war number of each wood, stone, metal & food.
3. Loot (if they draw)
 - a. Keeps all troops
 - b. Gains 200 * war number of each wood, stone, metal & food
4. Science
 - a. There are only a limited number of wars, so therefore enough science should be awarded every war to allow the upgrades to be bought multiple times
 - b. Therefore the player gains 5000 science * war number per win, but only 1000 science * war number on loss or draw. This is enough to buy the villa upgrade for the first war, but the player may decide to upgrade the first couple of upgrades a few times in order to increase their resource rates.

Again, this is all subject to change through the development of the program if I feel that the values should be changed.

As I stated at the bottom of that bit, this parts are subject to change, when I get onto balancing the program.

The final part to address, is the calculating of the enemy stats at the start of each war. I did this based off the design, using some random number generators (RNG) to make it a bit more interesting. Remember that all the values are subject to change after balancing.

```
void calculateNewEnemyStats() {
    // Enemy health 100-200% of player's (1x-2x)
    int randomHealth = rng.Next(1, 3); // MaxVal is exclusive
    GlobalData.combatData[0][1] = GlobalData.combatData[0][0] * randomHealth;
    // Enemy block 100-200% of player's (1x-2x)
    int randomBlock = rng.Next(1, 3);
    GlobalData.combatData[1][1] = GlobalData.combatData[1][0] * randomBlock;
    // Enemy damage 50-100% of player's (0.5x-1x)
    int randomAttack = rng.Next(1, 3);
    // Get around having to use random doubles
    if (randomAttack == 1) {
        GlobalData.combatData[2][1] = GlobalData.combatData[2][0] / 2;
    } else {
        GlobalData.combatData[2][1] = GlobalData.combatData[2][0];
    }
}
```

Just before I have began to do testing, I realised that the outcome of each battle is EXACTLY the same, because the stats are all static. This makes the game very boring, because there is no back and forth of the player and enemy occupation. Therefore, I will add some RNG into the battles. This has been solved by adding a randomly generated “bonus damage” during each turn, which is added onto player or enemy damage. E.g. on the player side it is done by:

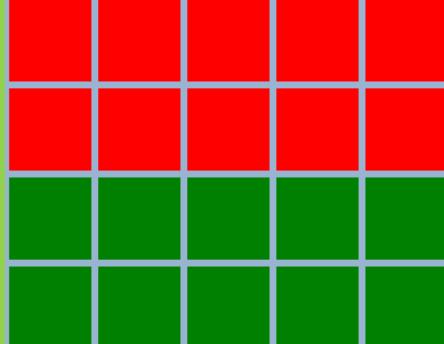
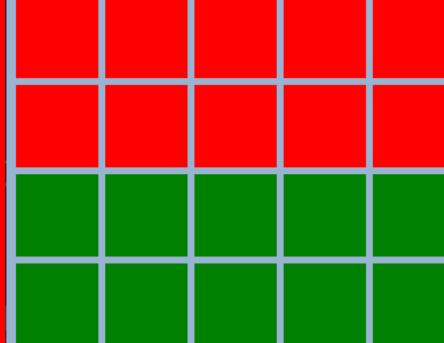
```
// Add some random into the damage, using a bonus damage
int bonusDamageType = rng.Next(1, 3);
int bonusDamage = 0;
switch (bonusDamageType) {
    case 1:
        bonusDamage = GlobalData.combatData[2][0] / 5;
        break;
    case 2:
        bonusDamage = GlobalData.combatData[2][0] / 2;
        break;
}
// Work out how much damage the player does
int blockedPlayerDamage = (playerDamage + bonusDamage) - enemyBlock;
enemyHealth -= blockedPlayerDamage;
```

But then, after watching my program run for a while, I realised I needed to balance it a bit to make it:

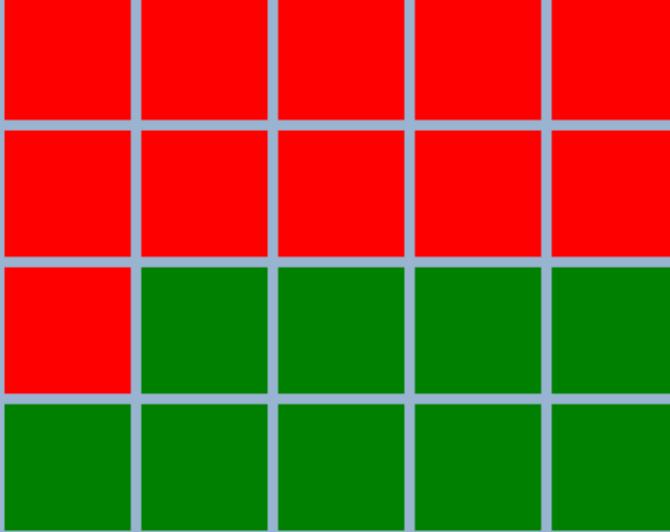
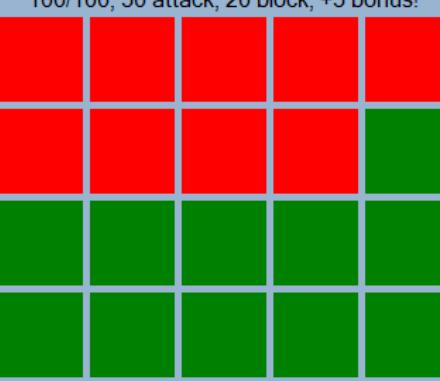
1. More fair for both sides – player doesn't just gain ALL tiles consecutively or the other way round for losing them
2. Not be in a constant stalemate – tiles going 10, 9, 10, 9, 10, 9 etc

The enemy turn bit is a bit different, from the balancing side. I just have the bonus damage being slightly higher, at `GlobalData.combatData[2][0] / 10` and `/ 5`.

Input	Expected Output	Actual Output
300 seconds (ticks) have passed since the game first loaded up (not from save file)	A new war starts, the health, attack and block labels are set to the correct values, the year and war name is shown, the grid is in the default state, new war message printed into combat logs	<p style="text-align: center;">Enemy</p> <p>60/200, 50 attack, 40 block, +5 bonus!</p> <p>Enemy occupied tiles: 10</p> <p>Year: 499BC.</p> <p>Current war: War of Lake Regillius.</p> <p>25/100, 50 attack, 20 block</p> <p style="text-align: center;">Player</p>
New war starts. Player has 100 health, 50 attack, 20 block, enemy has 200 health, 25 attack, 40 block	First round should go like this: -Player does 10 damage to enemy -Enemy does 5 damage to player -Repeat until all tiles are won Player wins, loot given to player, board and labels reset, outcome printed to combat logs, loot printed to loot logs.	<p style="text-align: center;">Enemy</p> <p>60/100, 50 attack, 20 block</p> <p>Enemy tiles: 9</p> <p>Year: 499BC.</p> <p>Current war: War of Lake Regillius.</p> <p>60/100, 50 attack, 20 block, +10 bonus!</p> <p style="text-align: center;">Player</p>

<p>New war starts. Both player and enemy have exact same stats.</p>	<p>The war will be a stalemate, with no sides gaining any territory at all. The draw is called, player gets appropriate loot, board and labels reset, outcome printed to combat logs, loot printed to loot logs.</p>	<p>Enemy</p> <p>No war running</p>  <p>Enemy tiles: 10</p> <p>Year: 494BC.</p> <p>Current war: None - Check loot and combat logs.</p> <p>No war running</p> <p>Player</p>	
<p>New war starts. Player has the upper hand so beats the enemy.</p>	<p>The player receives winning loot. Board reset, labels reset, winning message and loot gained printed into loot logs.</p>	<p>Story Upgrades Combat Loot Saves</p> <p>11:37: You have gained half of the enemy's troops, and gained 500 of each resource + 5000 science!</p>	
<p>A war has ended, another few ticks pass until a new war begins again.</p>	<p>Everything resets properly, new war message printed into combat logs.</p>	<p>Enemy</p> <p>No war running</p>  <p>Enemy tiles: 10</p> <p>Year: 498BC.</p> <p>Current war: None - Check loot and combat logs.</p> <p>No war running</p> <p>Player</p> <p>16:17: New war called War of Lake Regillius started! 16:19: You have won the war! The enemy will not get that much stronger!</p>	



		<p>16:17: New war called War of Lake Regilius started! 16:19: You have won the war! The enemy will not get that much stronger! 16:20: New war called War of Corbio started! 16:20: You have won the war! The enemy will not get that much stronger! 16:21: New war called War of Corbio started! 16:21: You have won the war! The enemy will not get that much stronger! 16:21: New war called War of Corbio started! 16:22: You have won the war! The enemy will not get that much stronger!</p> <p>Rinse and repeat</p>	
Enemy wins tiles	Grid updated to show one more red tile	 <p>100/100, 50 attack, 20 block, +25 bonus!</p> <p>Player</p> <p>ENEMY</p> <p>100/100, 50 attack, 20 block, +5 bonus!</p> <p>Enemy tiles: 9</p> <p>Year: 499BC.</p> <p>Current war: War of Lake Regilius.</p> <p>Allied tiles: 11</p> <p>Player</p>	Enemy Year Current war Allied
Player wins tiles	Grid updated to show one more green tile	 <p>65/100, 50 attack, 20 block</p>	

From this testing, it looks as if there is an issue with the combat after only the first war. I think the issue is that when the board is reset, the actual grid variable itself is not reset. Therefore, because at the end of a war, the grid already has all 1s and all 0s (because one side has won a war), and a new war starts, the winWarCheck() function is run, finds that the grid is already full of one side, and shows a winner. Therefore, my fix for this is just in the resetBoard() function (each line in its own for loop of the function):

```
grid[y, x] = ENEMY;
grid[y, x] = PLAYER;
```

Retesting this part:

Input	Expected Output	Actual Output
A war has ended, another few ticks pass until a new war begins again.	Everything resets properly, new war message printed into combat logs.	12:08: You have won the war! The enemy will not get that much stronger! 12:09: New war called War of Corbio started! 12:11: You have lost the war, the enemy will get much stronger :(

V2 27/11/19

Whilst working on the saving to file code, during testing I realised that the year counting code does not actually work as intended. This is because the line:

```
GlobalData.year = 1;
```

Always set the value to 1, which is not increasing it at all. This meant that the outcome of the condition afterwards (if year is more than 0) would always be true, thus didn't work correctly. I was originally confused about this because the comment above had said to increase the year by 1, yet the code was setting it to 1 and the way it worked before neither way would work.

I have, however, come up with a solution.

```
void yearCounter() {
    // If tick counter has reached a multiple of 60
    if ((GlobalData.tickCounter % 60) == 0) {
        // Increment year by 1
        GlobalData.year++;
        // If the year is negative
        if (GlobalData.year < 0) {
            // Get the positive year version
            GlobalData.actualYear = (GlobalData.year * -2) +
GlobalData.year;
            // Set the era to BC
            GlobalData.era = "BC";
        } else {
            GlobalData.actualYear = GlobalData.year;
            // Set the era to AD
            GlobalData.era = "AD";
        }
    }
}
```

E.g. if the actual year needs to be -400BC, but the year can only be positive ints, it needs to find a way to somehow convert from positive to negative (because of the middle being 0). Thus the year is multiplied by -2 and added on by itself.

Review 3

- Green – Feature finished (does not matter if there is a minor error or change)
- Orange – Feature about to be worked on next
- Red – Feature skipped/not going to be worked on

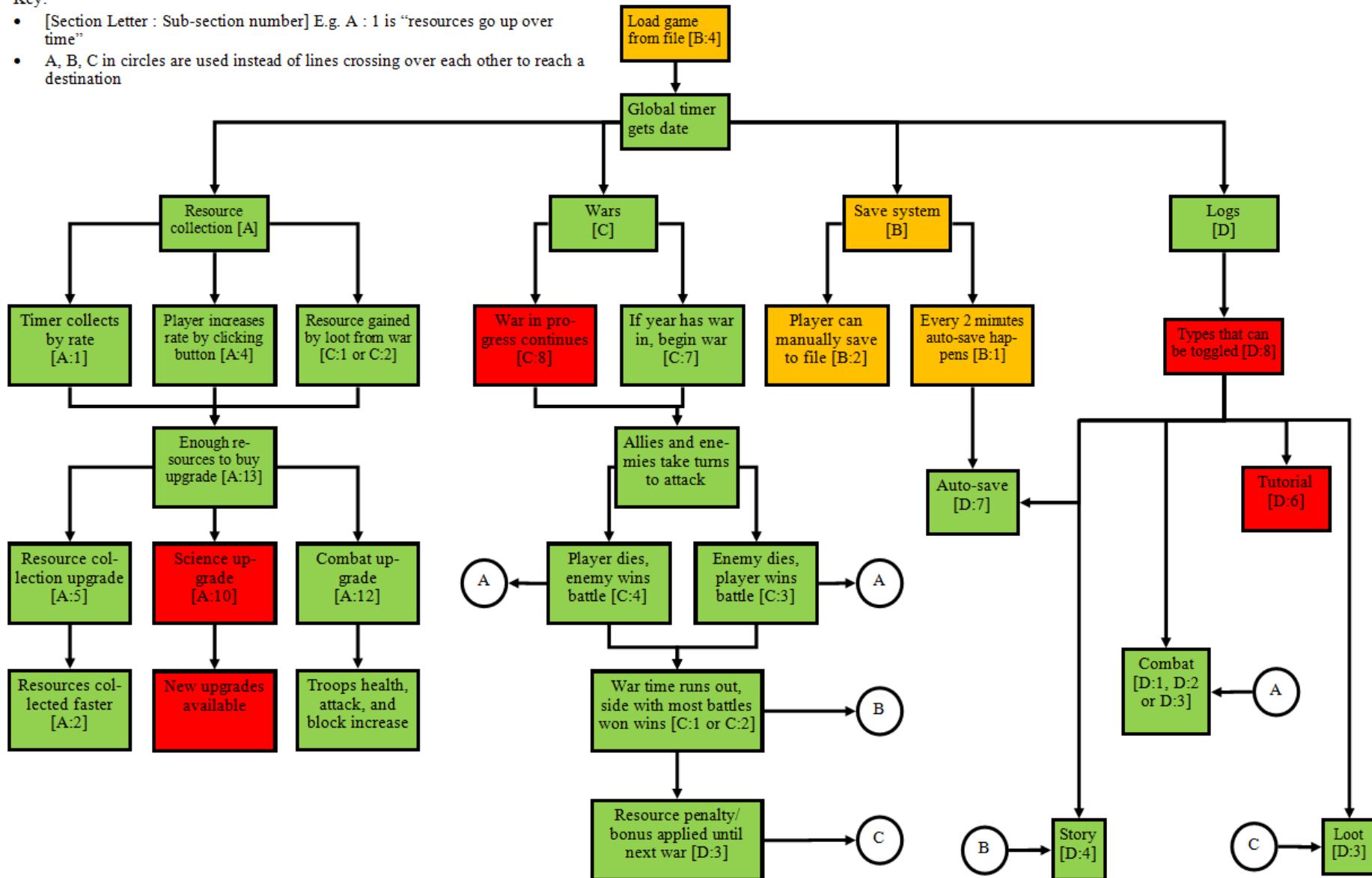
Criteria	How to evidence criteria being met	Section Code
Section A: Resources/upgrades system		
Resources go up over time	2 screenshots – first one taken before, second one taken after to show progress with resources	A1
Resource rate increased through upgrade	Screenshot of increased rate	A2
Resources reach “milestone” where the number shortens	Screenshot of 4,000 resource shown has 4k resource	A3 <i>Optional</i>
Button to manually increase rate of resource collection increased, button turns brown, all other buttons turn grey and toggle off, that resource rate increases by 10%	2 screenshots – first one taken before with one button pressed, second one taken after with another button pressed to show increased rate and only one rate bonus allowed at a time	A4 <i>Optional</i>
[Resource] storage upgrade is bought	Screenshots to show max [resource] capacity increased	A5
Worker for [resource] upgrade is bought, not enough housing space	Screenshot to show “Not enough housing space” in tutorial logs	A6
Worker for [resource] upgrade is bought, enough housing space	Screenshot to show increased [resource] rate	A7
Housing upgrade is bought	Screenshot to show housing number increased	A8
Science upgrade is bought, but not enough science points available	Screenshot to show “Not enough science to buy this” in tutorial logs	A9
Science upgrade for combat is bought, new combat upgrade button appears in combat tab	Screenshot to show new upgrade button	A10
Different upgrade types tabs selected	Screenshot to show only worker upgrades shown in workers tab	A11 <i>Optional</i>
Combat upgrade to increase block bought	Screenshot to show troops’ increased block	A12
Enough resources are available to buy upgrade	Screenshot to show upgrade button turning to “clickable” state	A13 <i>Optional</i>
Section B: Save/load system		
Game is auto-saved every 2 minutes	3 screenshots – first one taken when auto-save happens, second one taken 2 minutes later when second auto-save happens, third taken of auto-save file changed (see time stamp in file)	B1

Player manually presses the save game button	Screenshot of new save file created	B2
Game closed	Screenshot of auto-save file changed	B3
Game loaded up, player selects save file to load, message box tells player how many resources were made when they were offline	Screenshot of before game closed, screenshot of after game reopened and save file loaded, screenshot of message box	B4
Section C: Combat/war system		
War is won by player	Screenshot of resources rate gain, and screenshot of grid reset	C1
War is lost by player	Screenshot of resources rate loss, and screenshot of grid reset	C2
Battle is won by player	Screenshot of player gaining green tile from enemy's red tile	C3
Battle is lost by player	Screenshot of enemy gaining red tile from player's green tile	C4
Player's army with 170 health and 50 block gets hit by 100 attack, so health goes down to 120	Screenshot before and after turn to show health/block/attack works	C5
Number of troops increased, total army health increases by (current upgrade of health) * number of new troops	Screenshot to show the stats of army increased	C6
New war started when right year reached, enemies have higher health/attack/block than in previous war	Screenshot to show stats in old war, and then stats in new	C7
Game loaded up, war in progress continues as before	Screenshot showing war before game close, after game close	C8
Section D: Logs system		
A battle is lost or won, show the message in the combat logs	Screenshot of combat logs	D1
A war is lost or won, show the statistics in the combat logs	Screenshot of combat logs	D2 <i>Optional</i>
A war is lost or won, logs show resource penalty/bonus until next war	Screenshot of loot logs	D3
A tech upgrade is made, and story progresses, show message in logs	Screenshot of story logs	D4
A new year begins, print new year in logs	Screenshot of story logs	D5
Player battles for first time, buys first upgrade or reaches first resource milestone, print	Screenshot of tutorial logs	D6

various tips or explanations in tutorial logs		
Game is auto-saved or manually saved by user, show message in auto-saves logs	Screenshot of auto-save logs	D7
A log type button is clicked to toggle it off, button changes from green to red and those logs stop showing	Screenshot of log toggle buttons and logs to show right logs toggled off	D8
The clear logs button is clicked, all logs toggled on cleared	Screenshot of nothing in logs	D9
Save logs to file button is clicked, all logs saved to a text file	Screenshot of file it saved to	D10 <i>Optional</i>
Section E: Usability		
The game balanced so that it can't be progressed really quickly (amount upgrades cost, how much the affect various parts of the game etc.)	Screenshots of player being able to keep up with enemies' combat stats, to show balanced progression through time	E1 <i>Optional</i>
Intuitive and easy to use menu	All of the game menus can be traversed within 2 clicks of the main game screen. Screenshots showing each menu	E2
Game does not have low framerate	Screenshot of CPU load reduction when game closed	E3
Big buttons, some colour coding to show toggles and being able to be clicked	Screenshots of buttons changing colour/position when various buttons pressed	E4 <i>Optional</i>

Key:

- [Section Letter : Sub-section number] E.g. A : 1 is “resources go up over time”
- A, B, C in circles are used instead of lines crossing over each other to reach a destination



Save/Load GUI [B]

V1 19/11/19

I will again begin with designing it directly off my GUI designs and see how it goes.

Save Current	Delete Current
Load Game	
Autosave Option	Autosave interval: 5 minutes
The current time (UTC) is 18:00	

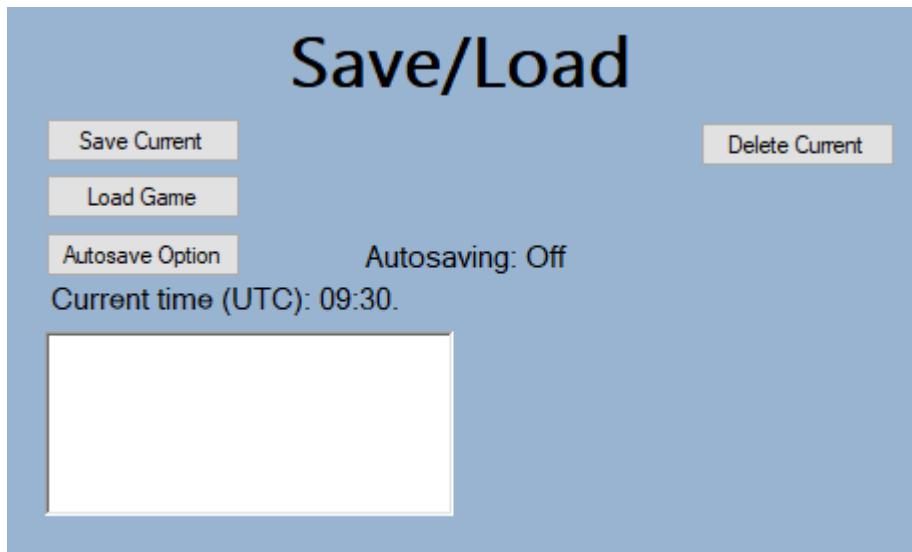
newGame1.txt myGame.txt thebestsave.txt anewworld.txt
Submit

Save Current	Delete Current		
Load Game	<table border="1"><tr><td>Yes</td></tr><tr><td>No</td></tr></table>	Yes	No
Yes			
No			
Autosave Option			
Current time (UTC) is: 18:00			
newgame.txt autosave.txt badgame.txt hopefullyiwininthistime.txt bobbyiscool.txt			

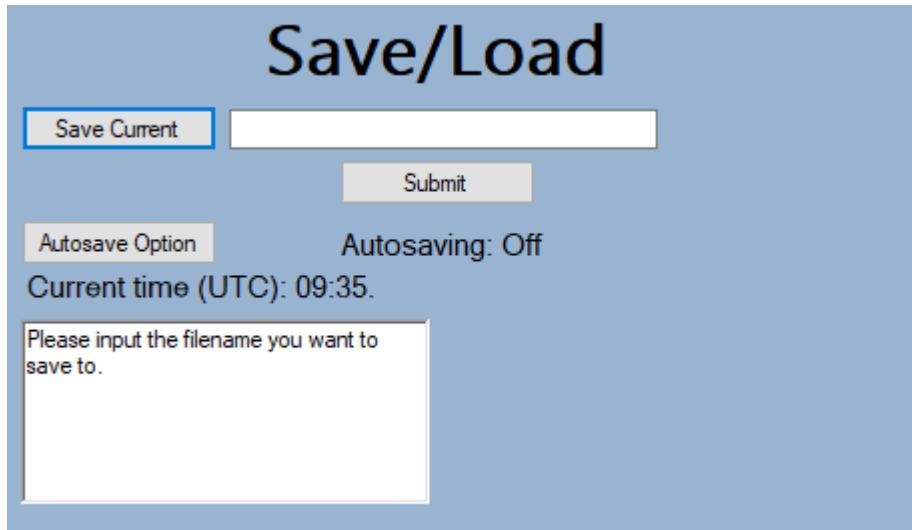
This layout, however, has caused a few issues:

1. There is no easy output box for any messages that the user needs to see quickly, e.g. if they definitely want to delete a file, or telling them to select a file to load from when they click load game
2. If, for example, the load game button is pressed and the user selects the file they want to load, how does the user “submit” the file they want to load? I don’t want to use a `listbox.SelectedText` event because the user might want to switch or might accidentally select the wrong thing

3. When the save game button is pressed, there is no place to input a filename of their choice
Therefore, I have come up with this GUI to fill these requirements.

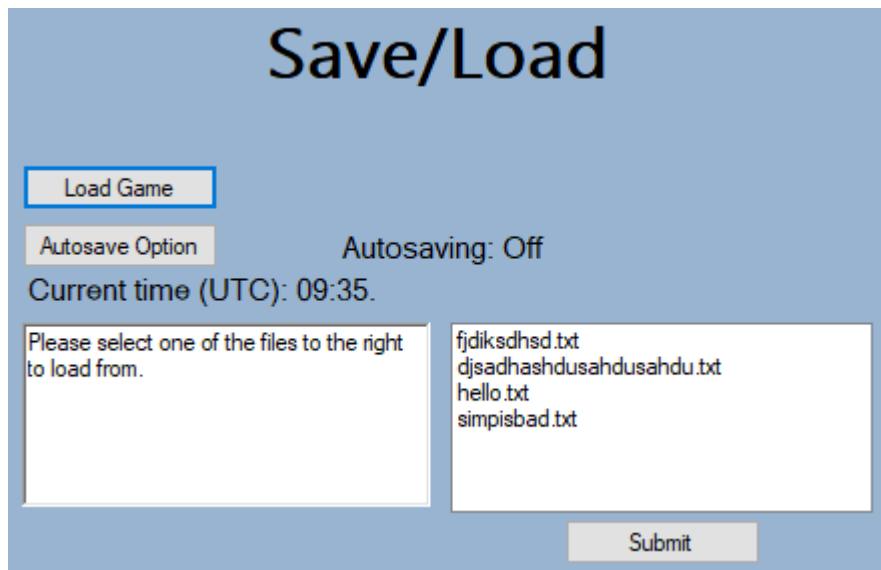


Starting menu



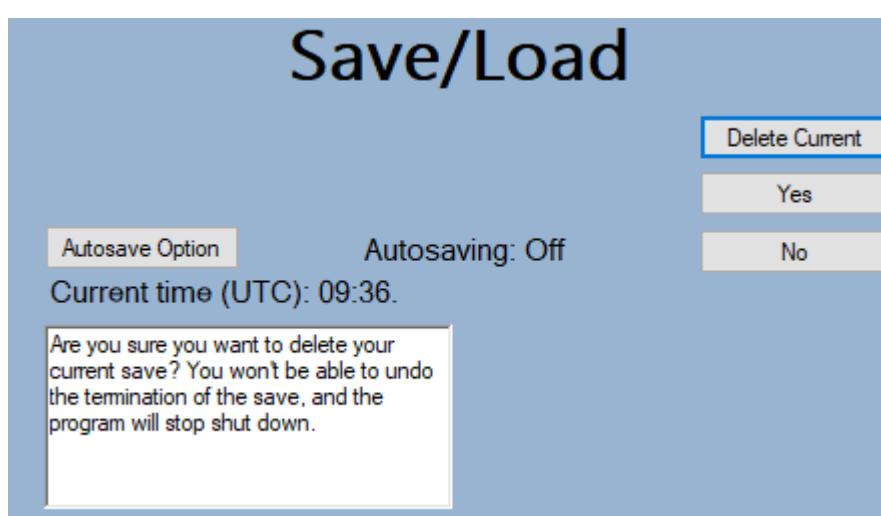
Save current button

pressed



pressed

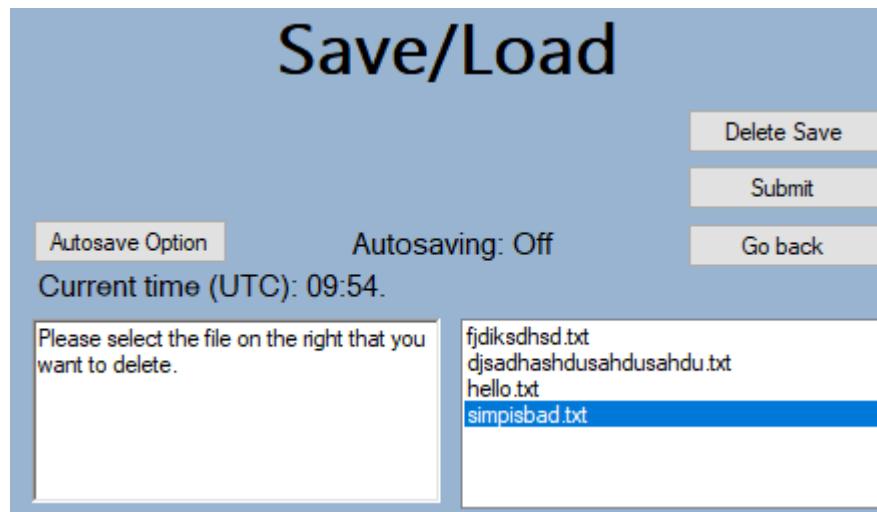
Load game button



pressed

Delete current button

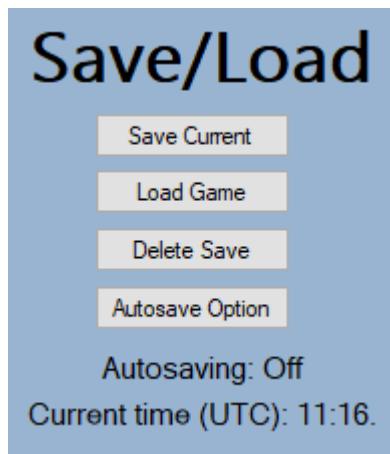
My stakeholders seem happy about this... bar one thing. One of them has asked what the “delete current” button means. Is it meant to delete the current save by just closing the program thus losing all its data? That doesn’t really make sense, because the user could do that themselves. Therefore, they asked if I could change the delete button to allow the user to select a file to delete and the program will remove that file. This makes a lot more sense, so when the user presses the delete button now, the following happens:



The “Go Back” button just returns the screen to default and the “Submit” button “locks in” the user’s choice and deletes the save.

V2 15/12/19

Because the stakeholders wanted a file dialog box to save, load and delete files, I can now improve the GUI greatly by removing the parts that had to be hidden/shown at various different points, and were in general a bit of a mess. This is the new GUI:



It is very simple; top 3 buttons open the dialog box, autosave option changes the label below it and the current time label updates on the minute change.

Save/Load Code [B]

Buttons

V1 21/11/19

I have already coded the buttons and controls on the GUI so that they show and hide based on different events. I have 4 “sections” of buttons:

1. Saving:

```
private void btnSave_Click(object sender, EventArgs e) {
    rtxtSavingOutput.Text = ("Please input the filename you want to save to.");
    txtSaveName.Show();
    btnSubmitFileName.Show();
    btnLoad.Hide();
    btnDelete.Hide();
}

private void btnSubmitFileName_Click(object sender, EventArgs e) {
    saveToFile(txtSaveName.Text);
    txtSaveName.Hide();
    btnSubmitFileName.Hide();
    btnLoad.Show();
    btnDelete.Show();
}
```

2. Loading:

```
private void btnLoad_Click(object sender, EventArgs e) {
    rtxtSavingOutput.Text = ("Please select one of the files to the right to
load from.");
    lbSelectFile.Show();
    btnSave.Hide();
    btnDelete.Hide();
    displayFiles();
    btnSubmitSelectedFile.Show();
}

private void btnSubmitSelectedFile_Click(object sender, EventArgs e) {
    loadFromFile(lbSelectFile.SelectedItem.ToString());
    btnSubmitSelectedFile.Hide();
    lbSelectFile.Hide();
    btnSave.Show();
    btnDelete.Show();
}
```

3. Deleting:

```
private void btnDelete_Click(object sender, EventArgs e) {
    rtxtSavingOutput.Text = ("Please select the file on the right that you want
to delete.");
    lbSelectFile.Show();
    pnlDeleteSure.Show();
    btnLoad.Hide();
    btnSave.Hide();
    displayFiles();
}
```

```

private void btnDeleteYes_Click(object sender, EventArgs e) {
    deleteSave(lbSelectFile.SelectedItem.ToString());
    rtxtSavingOutput.Text = ("Save deleted.");
    pnlDeleteSure.Hide();
    btnLoad.Show();
    btnSave.Show();
    btnSubmitSelectedFile.Hide();
    lbSelectFile.Hide();
}

private void btnDeleteNo_Click(object sender, EventArgs e) {
    rtxtSavingOutput.Text = ("Save saved.");
    pnlDeleteSure.Hide();
    btnSubmitSelectedFile.Hide();
    lbSelectFile.Hide();
    btnLoad.Show();
    btnSave.Show();
}

```

4. Autosaving

```

private void btnAutosaveOption_Click(object sender, EventArgs e) {
    if (isAutosaveToggled) {
        autosaveTimer.Stop();
        isAutosaveToggled = false;
        rtxtSavingOutput.Text = ("Autosave is off - be careful!");
        lblAutosaveStatus.Text = ("Autosaving off.");
    } else {
        autosaveTimer.Start();
        isAutosaveToggled = true;
        rtxtSavingOutput.Text = ("Autosave is on, interval is every 5
minutes.");
        lblAutosaveStatus.Text = ("Autosave interval: 5 minutes");
    }
}

```

V2 15/12/19

Now that I have removed a lot of buttons and controls to replace them with the dialog box, the buttons code is very simple and not at all confusing anymore.

```

private void btnSave_Click(object sender, EventArgs e) {
    saveToFile(false);
}

private void btnLoad_Click(object sender, EventArgs e) {
    loadFromFile();
}

private void btnAutosaveOption_Click(object sender, EventArgs e) {
    if (isAutosaveToggled) {
        autosaveTimer.Stop();
        isAutosaveToggled = false;
        lblAutosaveStatus.Text = ("Autosaving off.");
    } else {
        autosaveTimer.Start();
        isAutosaveToggled = true;
        lblAutosaveStatus.Text = ("Autosave interval: 5 minutes");
    }
}

```

```
private void btnDelete_Click(object sender, EventArgs e) {
    deleteSave();
}
```

Display Files

V1 23/11/19

```
/// <summary>
/// Gets and displays the text files in a list box
/// </summary>
void displayFiles() {
    // Get files from local directory
    DirectoryInfo localFolder = new
    DirectoryInfo(System.IO.Path.GetDirectoryName(System.Windows.Forms.Application.Exec
utablePath));
    // Finds all text files in the directory
    FileInfo[] files = localFolder.GetFiles("*.txt");
    foreach (FileInfo f in files) {
        lbSelectFile.Text += ("\n" + f.Name);
    }
}
```

V2 15/12/19

This function no longer exists, because the file dialogs do everything for it.

Delete Files

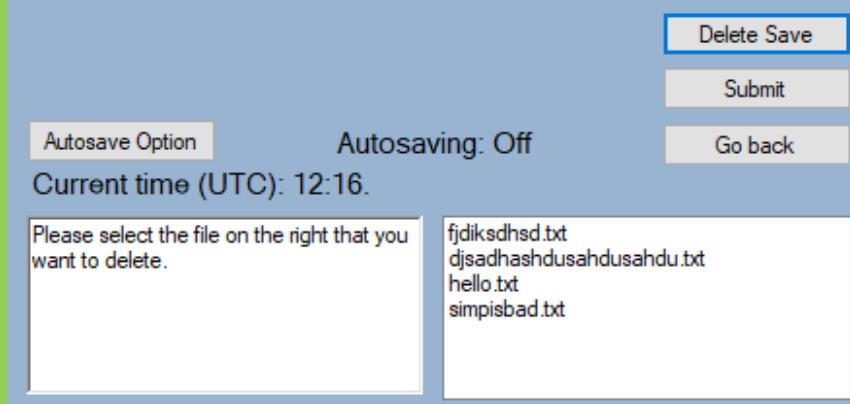
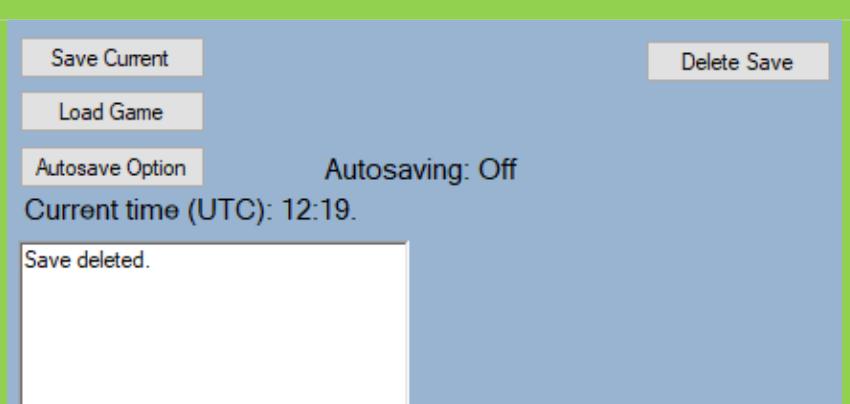
V1 26/11/19

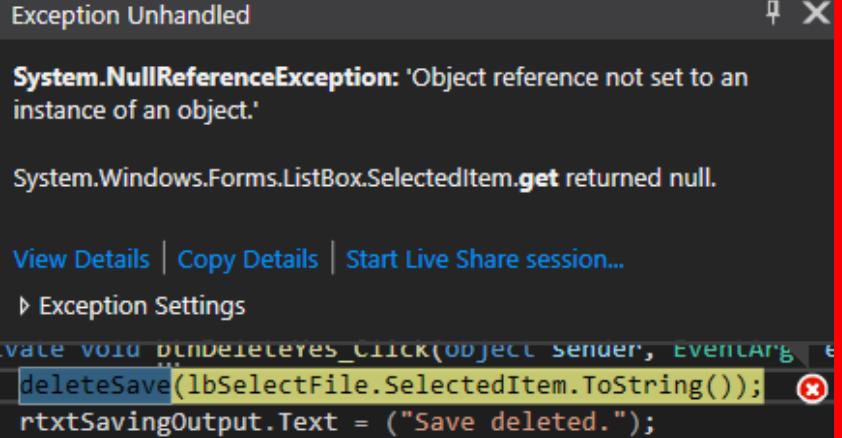
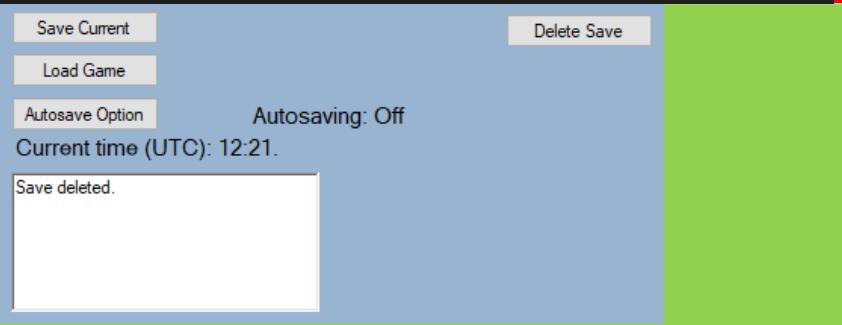
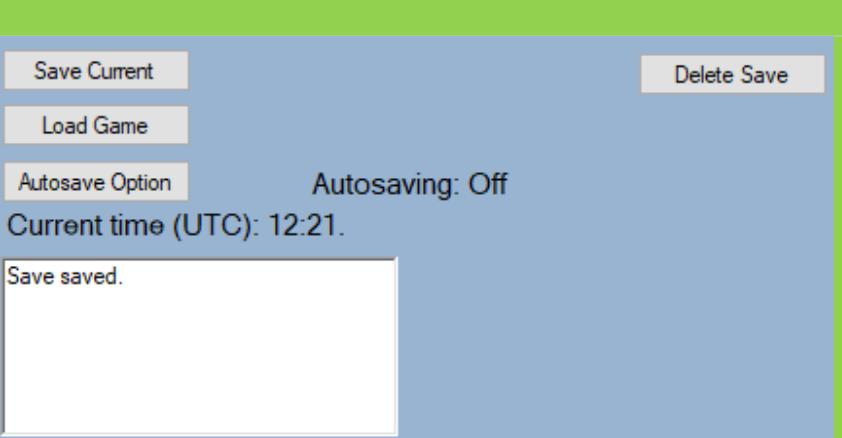
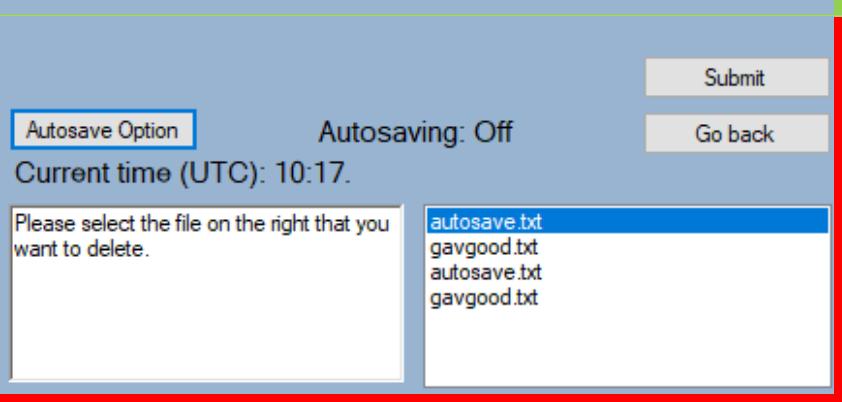
```

void deleteSave(string fileName) {
    // Get files from local directory
    DirectoryInfo localFolder = new
DirectoryInfo(System.IO.Path.GetDirectoryName(System.Windows.Forms.Application.Exec
utablePath));
    // Finds all text files in the directory
    FileInfo[] files = localFolder.GetFiles("*.txt");
    foreach (FileInfo f in files) {
        // If selected file = current file
        if (fileName + ".txt" == f.Name) {
            // Delete file
            f.Delete();
        }
    }
}

```

I can quickly test this:

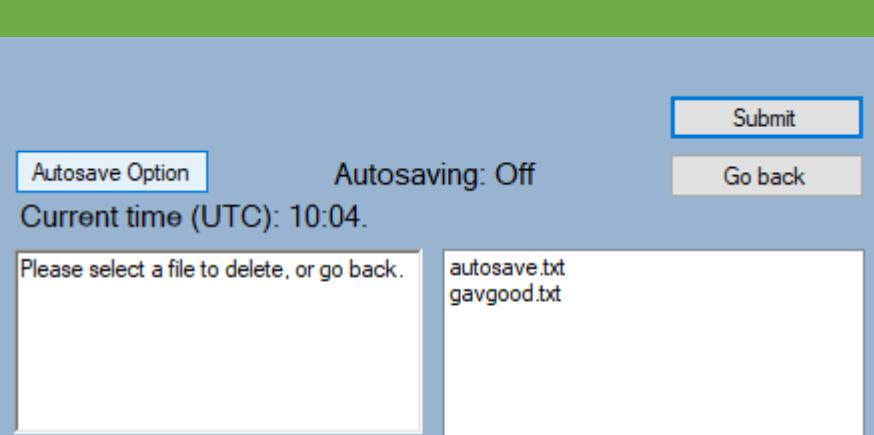
Input	Expected Output	Actual Output
The delete game button is pressed	The select file list box, submit button show	
A file is selected, submit button is pressed	The “are you sure you want to delete this” panel show (with yes and no buttons inside)	

A file is not selected, submit button is pressed	Output in box that a file must be selected to delete	
The yes button is pressed	The selected file is deleted permanently , output this to user in logs	
The no button is pressed	The selected file is saved, output this to user in logs	
The delete button is pressed, file is deleted, then the delete button is pressed again	The selected file is deleted, output to logs. Next time files are shown, the deleted file no longer appears.	

I have attempted to fix this one failed test about it crashing by inserting a check if the list box doesn't have a selected item when the submit button is pressed.

Input	Expected Output	Actual Output
A file is not selected, submit button is pressed	Output in box that a file must be selected to delete	<pre>private void btnDeleteYes_Click(object sender, EventArgs e) { if (lbSelectFile.SelectedItem.ToString() == null) { rtxtSavingOutput.Text = ("Please select a file."); } else { deleteSave(lbSelectFile.SelectedItem.ToString()); rtxtSavingOutput.Text = ("File deleted."); } }</pre> <p>Exception Unhandled</p> <p>System.NullReferenceException: 'Object reference not set to an instance of an object.'</p> <p>System.Windows.Forms.ListBox.SelectedItem.get returned null.</p>

This however has not worked, I think because of the .ToString() method not being needed.

Input	Expected Output	Actual Output
A file is not selected, submit button is pressed	Output in box that a file must be selected to delete	

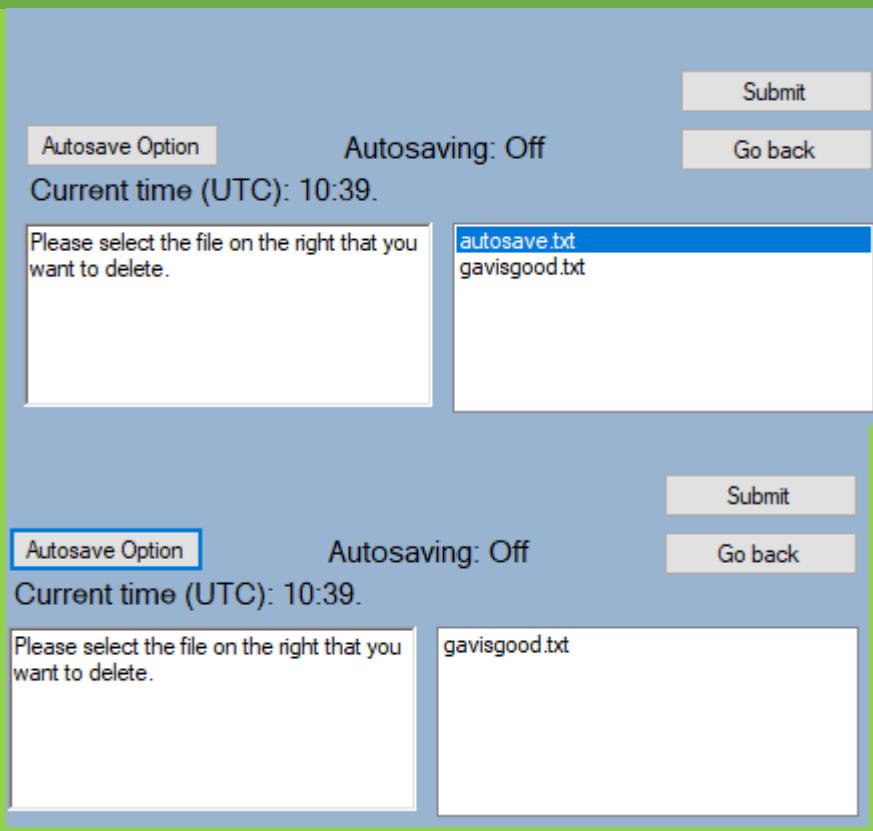
Now the next test to fix is deleting a file and reshowing the file directory again. I have worked out that there are 2 issues:

1. Looking in the file directory, the files are not actually being deleted, so there is something wrong with the deleteFiles() function
2. In displayFiles(), the list box is just added to repeatedly, and never cleared, thus explaining the duplicated items

I have fixed this second point by adding the line `lbSelectFile.Items.Clear();` before the foreach loop in displayFiles.

The second part has been solved by re-writing the deleteSave() function:

```
void deleteSave(string fileName) {
    string selectedFile = ("./" + fileName);
    File.Delete(selectedFile);
}
```

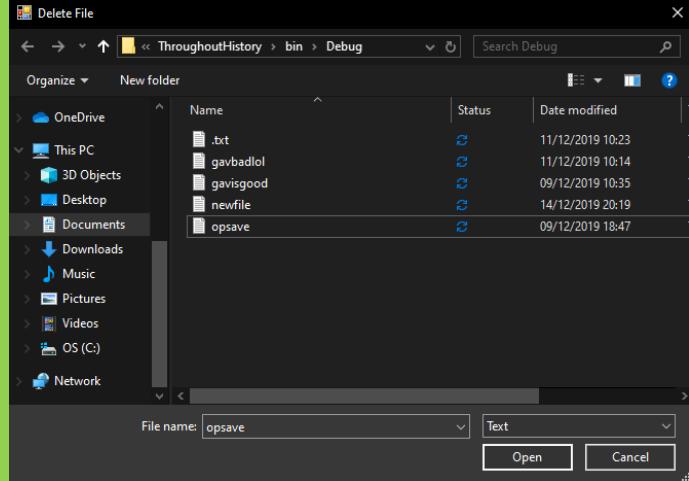
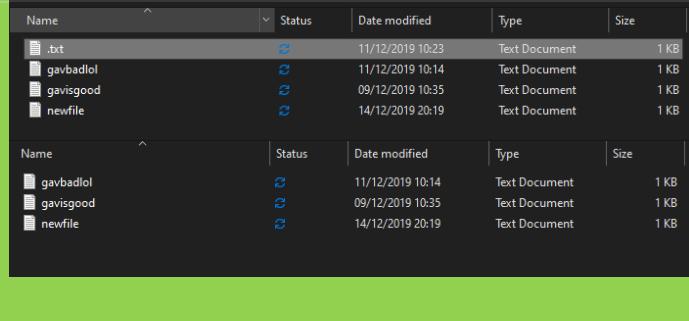
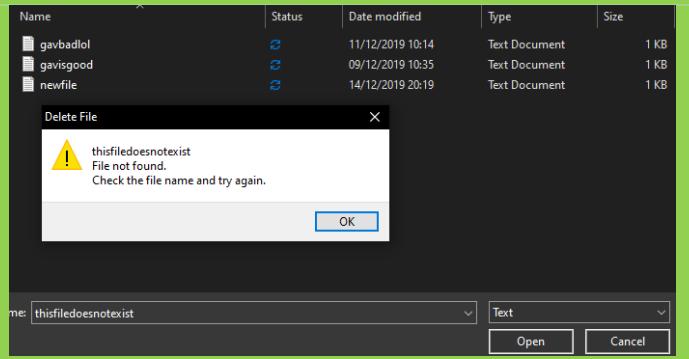
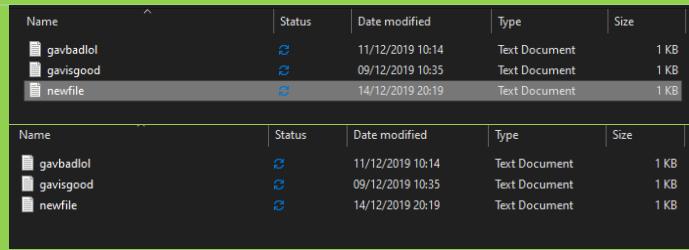
Input	Expected Output	Actual Output
The delete button is pressed, file is deleted, then the delete button is pressed again	The selected file is deleted, output to logs. Next time files are shown, the deleted file no longer appears.	 <p>The screenshot shows a Windows-style file dialog. At the top, there are buttons for 'Autosave Option' (highlighted in blue), 'Submit', and 'Go back'. Below that, it says 'Autosaving: Off' and 'Current time (UTC): 10:39'. A message box says 'Please select the file on the right that you want to delete.' On the right side, there is a list of files: 'autosave.txt' and 'gavisgood.txt', with 'autosave.txt' having a blue selection bar around it. At the bottom, there are 'Submit' and 'Go back' buttons.</p>  <p>The screenshot shows a Windows-style file dialog. At the top, there are buttons for 'Autosave Option' (highlighted in blue), 'Submit', and 'Go back'. Below that, it says 'Autosaving: Off' and 'Current time (UTC): 10:39'. A message box says 'Please select the file on the right that you want to delete.' On the right side, there is a list of files: 'gavisgood.txt'. At the bottom, there are 'Submit' and 'Go back' buttons.</p>

V2 15/12/19

The deleting saves code is still very simple:

```
void deleteSave() {
    // Set dialog filter to text files only
    openDialogue.Filter = "Text|*.txt";
    // Show dialogue
    DialogResult result = openDialogue.ShowDialog();
    // If user pressed OK button
    if (result == DialogResult.OK) {
        string selectedFile = openDialogue.FileName;
        // Delete file
        File.Delete(selectedFile);
        txtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ":" +
selectedFile + " has been deleted permanently.");
    }
}
```

I would test this using my test table, however the table assumes I have all the previous windows controls like the yes and no buttons etc, which is no longer what is required. Therefore I will just use improvised tests for each section.

Input	Expected Output	Actual Output
Delete save button is pressed	Open file dialog appears with text file filter on	
Delete save button is pressed, file in the selected directory is entered or selected, OK button pressed	Open file dialog appears with text file filter on, when OK button pressed, dialog closes and save deleted	
Delete save button is pressed, a file name is entered, even though that file does not exist in the current directory	Open file dialog appears with text file filter on, when non-existent file name entered, message box appears saying “[filename] File not found. Check the file name and try again”	
Delete save button is pressed, cancel button pressed	Open file dialog appears with text file filter on, when cancel button pressed the dialog closes and nothing happens	

Autosaving

V1 29/11/19

```
private void autosaveTimer_Tick(object sender, EventArgs e) {
    autosaveTick++;
    rtxtSaves.Text += ("\n" + autosaveTick);
    if ((autosaveTick % 300) == 0) {
        saveToFile("autosave");
        rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": The
game has been autosaved!");
    }
}
```

Input	Expected Output	Actual Output
Game is first loaded up, 5 minutes (300 ticks) pass	First autosave happens, to file "autosave.txt". If it already exists, overwrite file. Show autosave has been made in logs	300 12:26: File has been saved to ./autosave.txt 12:26: The game has been autosaved! 301 600 12:31: File has been saved to ./autosave.txt 12:31: The game has been autosaved! 601
The autosave button is pressed, autosave timer currently on	The autosave timer is toggled off, output that to logs and label	Autosave Option Autosave interval: 5 minutes Autosave Option Autosaving off.
The autosave button is pressed, autosave timer currently off	The autosave timer is toggled on, output that to logs and label	Autosave Option Autosaving: Off Autosave Option Autosave interval: 5 minutes

V2 15/12/19

Only one line changes here:

```
saveToFile(true);
```

Input	Expected Output	Actual Output
Game is first loaded up, 5 minutes (300 ticks) pass	First autosave happens, to file "autosave.txt". If it already exists, overwrite file. Show autosave has been made in logs	300 12:26: File has been saved to ./autosave.txt 12:26: The game has been autosaved! 301

		<p>600 12:31: File has been saved to ./autosave.txt 12:31: The game has been autosaved!</p> <p>601</p>
The autosave button is pressed, autosave timer currently on	The autosave timer is toggled off, output that to logs and label	
The autosave button is pressed, autosave timer currently off	The autosave timer is toggled on, output that to logs and label	

V3 20/12/19

I have looked over my success criteria again and have found that I am missing criteria B3, where on game close the game is autosaved. This is very easy to complete – just call the saveToFile(true) function when the exit button on the form is pressed. The only drawback of this solution is that when the program is closed through any method other than pressing the red close button, the game will not be autosaved, but that is not really something I can get around. I've made a small test for this:

Input	Expected Output	Actual Output
Exit button clicked, autosave happens just before exit	Autosave file updated	<p>First few values of the autosave file before game closed :</p> <p>First few values of the autosave file after game closed:</p>

[name]

Candidate No: xxxx

Centre No:xxxxx

New File structure

[V1 02/12/19](#)

Below is an image of the file structures part I had in my design.

File Structures

Each line in the below table represents a line in the save file. Each item will be separated by a # because then the program can go through the file and check for each #, and thus pull out/put in each relevant part of the file.

Wood amount	Stone amount	Food amount	Metal amount
Wood rate	Stone rate	Food rate	Metal rate
Wood capacity	Stone capacity	Food capacity	Metal capacity
Wood gather multiplier	Stone gather multiplier	Food gather multiplier	Metal gather multiplier
Housing amount	Housing remaining		
Troop block	Troop attack	Troop health	Troop amount
Number of tiles held by player	Number of tiles held by enemy		
Wood storage upgrade cost	Stone storage upgrade cost	Food storage upgrade cost	Metal storage upgrade cost
Wood workers upgrade cost	Stone workers upgrade cost	Food workers upgrade cost	Metal workers upgrade cost
Science upgrade cost			
Shack housing cost	Boot camp housing cost	Barracks housing cost	
Shack housing increase	Boot camp housing increase	Barracks housing increase	
System time			

However, the number of global variables that I have got now has increased during development and I need to rethink how I need to layout my files.

Based off all of my current globalData variables, I have built a new file structure:

Wood amount	Stone amount	Food amount	Metal amount	
Wood rate	Stone rate	Food rate	Metal rate	
Wood capacity	Stone capacity	Food capacity	Metal capacity	
Wood gather multiplier	Stone gather multiplier	Food gather multiplier	Metal gather multiplier	
Science data				
Total housing				
Housing remaining				
Wood storage cost	Stone storage cost	Food storage cost	Metal storage cost	
Wood workers cost	Stone workers cost	Food workers cost	Metal workers cost	
Aqueducts cost	Stamp-mill cost	Trip-hammer cost	Hushing cost	Villa cost
Shack cost	Boot camp cost	Barracks cost		
Health cost	Block cost	Attack cost	Troop cost	

Storage multiplier	Workers multiplier	Research multiplier	Housing multiplier	Combat multiplier
Player health	Enemy health			
Player block	Enemy block			
Player damage	Enemy damage			
Player troop count	Enemy troop count			
Tick count				
Current war				
War number				
Year				
Actual year				
Era				

Where each row represents a different variable, and multiple items in one row means the variable is an array. So here, for example, the file looks like this, when a fresh new game is started:

```
1#1#1#1
1#1#1#1
100#100#100#100
1#1#1#1
0
5
5
50#50#50#50
100#100#100#100
100#500#1000#3000#5000
50#100#500
200#500#300#200
100#100
20#20
50#20
1#1
0
0
0
-500
500
BC
```

Saving to file

V1 05/12/19

Now that I have this new file format, I can write the saving to file code. First, I need to write the check statement if the filename is null, and the part that checks if a file with that name already exists in the directory.

```
void saveToFile(string fileName) {
    if (fileName == null) {
        rtxtSavingOutput.Text = ("Please input a valid save name. It cannot
        contain any of the following characters: \\ / : * ? " < > |");
    } else {
        // Get files from local directory
        DirectoryInfo localFolder = new
        DirectoryInfo(System.IO.Path.GetDirectoryName(System.Windows.Forms.Application.Exec
utablePath));
        // Finds all text files in the directory
        FileInfo[] files = localFolder.GetFiles("*.txt");
        foreach (FileInfo f in files) {
            // If selected file = current file
            if (fileName + ".txt" == f.Name) {
                rtxtSavingOutput.Text = ("This file already exists.");
                break;
            }
        }
    }
}
```

Next is the code that actually writes the data to the file in the correct way.

```
// Stop global timer so that all values are paused at the correct
place
globalTimer.Stop();
// Get total filename path
string selectedFile = ("./" + fileName + ".txt");
// Now that validation is done, make new file in directory, open it
and write to it
Stream cs = File.Create(selectedFile);
using (StreamWriter sw = new StreamWriter(cs)) {
    // Write resourcesData to the file
    for (int i = 0; i < 4; i++) {
        sw.WriteLine(GlobalData.resourcesData[i][0] + "#" +
GlobalData.resourcesData[i][1] + "#" + GlobalData.resourcesData[i][2] + "#" +
GlobalData.resourcesData[i][3]);
    }

    sw.WriteLine(GlobalData.scienceData);
    sw.WriteLine(GlobalData.totalHousing);
    sw.WriteLine(GlobalData.housingRemaining);

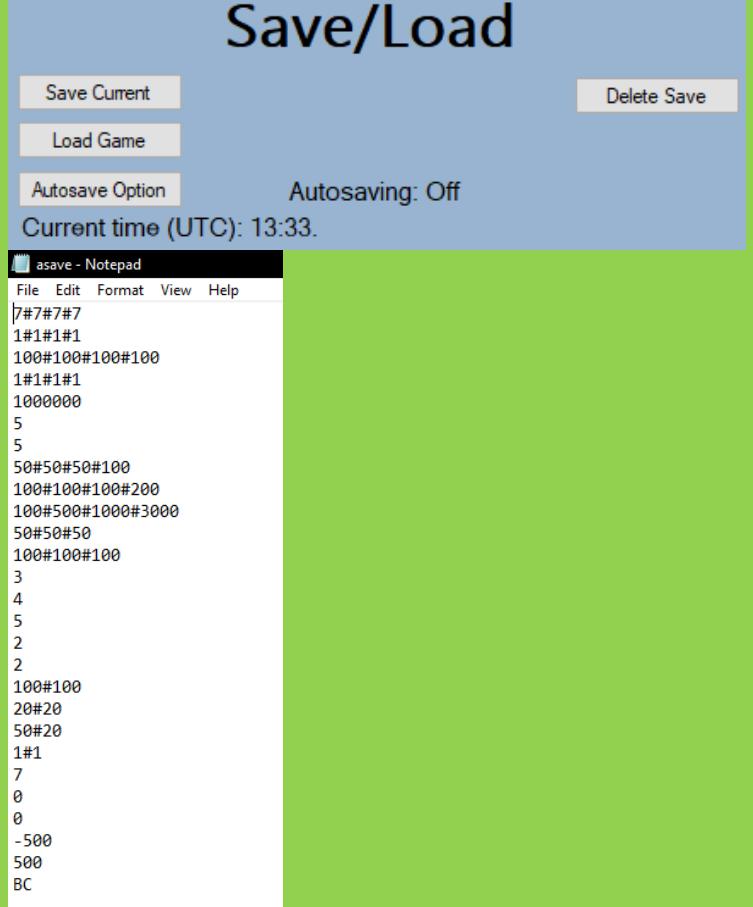
    // Write upgradesData to file
    for (int i = 0; i < 3; i++) {
        sw.WriteLine(GlobalData.upgradesCosts[i][0] + "#" +
GlobalData.upgradesCosts[i][1] + "#" + GlobalData.upgradesCosts[i][2] + "#" +
GlobalData.upgradesCosts[i][3]);
    }
    for (int i = 0; i < 2; i++) {
        sw.WriteLine(GlobalData.upgradesCosts[i][0] + "#" +
GlobalData.upgradesCosts[i][1] + GlobalData.upgradesCosts[i][2]);
    }
}
```

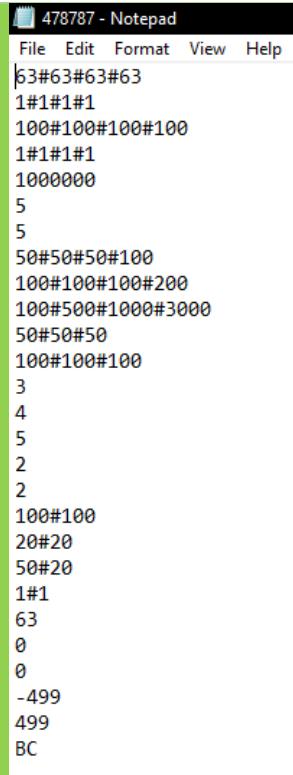
```
// Write costMultipliers to file
for (int i = 0; i < 5; i++) {
    sw.WriteLine(GlobalData.costMultipliers[i]);
}

// Write combatData to file
for (int i = 0; i < 4; i++) {
    sw.WriteLine(GlobalData.combatData[i][0] + "#" +
GlobalData.combatData[i][1]);
}

sw.WriteLine(GlobalData.tickCounter);
sw.WriteLine(GlobalData.currenWar);
sw.WriteLine(GlobalData.warNumber);
sw.WriteLine(GlobalData.year);
sw.WriteLine(GlobalData.actualYear);
sw.WriteLine(GlobalData.era);
}
// Restart the global timer now that the saving is finished
globalTimer.Start();
```

Input	Expected Output	Actual Output
Save game button is pressed	Text box and submit button to enter filename to save to appears, load game and delete game buttons hide	

<p>Filename “asave” is entered and submit button pressed after save game button pressed</p>	<p>A new file is created in the executable file path called “asave.txt” with all the appropriate values inside. Output file saved successfully into logs and output box</p>	 <p>The screenshot shows a "Save/Load" interface with buttons for "Save Current", "Load Game", "Autosave Option", and "Delete Save". Below these is a status bar indicating "Autosaving: Off" and "Current time (UTC): 13:35". A Notepad window titled "asave.txt - Notepad" is open, displaying the following text:</p> <pre> File Edit Format View Help 100#100#100#100 1#1#1#1 100#100#100#100 1#1#1#1 1000000 5 5 50#50#50#100 100#100#100#200 100#500#1000#3000 50#50#50 100#100#100 3 4 5 2 2 100#100 20#20 50#20 1#1 114 0 0 -499 499 BC </pre>

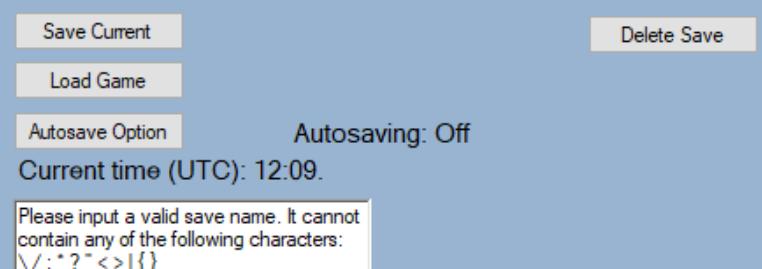
Filename "478787" is entered	A new file called "478787.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	 <p>The screenshot shows a Notepad window titled "478787 - Notepad". The content of the file is as follows:</p> <pre>File Edit Format View Help 63#63#63#63 1#1#1#1 100#100#100#100 1#1#1#1 1000000 5 5 50#50#50#100 100#100#100#200 100#500#1000#3000 50#50#50 100#100#100 3 4 5 2 2 100#100 20#20 50#20 1#1 63 0 0 -499 499 BC</pre> <p>Below the Notepad window is a "Save/Load" dialog box:</p> <p>Save/Load</p> <p>Buttons: Save Current, Load Game, Delete Save.</p> <p>Options: Autosave Option, Autosaving: Off, Current time (UTC): 10:41.</p>
------------------------------	---	---

<p>Filename “bobby45800.txt” is entered</p>	<p>A new file called “bobby45800.txt” is created with all the appropriate values inside. Output file saved successfully into logs and output box</p>		
<p>A file called “gavgood” is saved, and then the same name is entered for the next one</p>	<p>The file “gavgood” is saved, then the next time it is overwritten into the same file</p>		

Filename containing any of the following characters in its name is entered: “\/*?”<> []@#”	The output box says that the file cannot be created because one of the invalid filename characters has been entered. Output file saved successfully into logs and output box	<pre>// Now that validation is done, make new file in directory, open it and write to it Stream cs = File.Create(selectedFile); using (StreamWriter sw = new StreamWriter(cs)) { // Write resourcesData to the file for (int i = 0; i < 4; i++) { sw.WriteLine(GlobalData.resources[i]); } sw.WriteLine(GlobalData.scienceData); sw.WriteLine(GlobalData.totalHousing); }</pre>
---	---	---

To fix this final test, all I have to do is add more conditions for each letter type at the beginning if statement:

```
if (fileName == null || fileName.Contains("\\\\") || fileName.Contains("/") ||
fileName.Contains(":") || fileName.Contains("*") || fileName.Contains("?") ||
fileName.Contains(" ") || fileName.Contains("<") || fileName.Contains(">") ||
fileName.Contains("|") || fileName.Contains("[") || fileName.Contains("]")) {
    rtxtSavingOutput.Text = ("Please input a valid save name. It
cannot contain any of the following characters: \\\ / : * ? < > | { }")
```

Input	Expected Output	Actual Output
Filename containing any of the following characters in its name is entered: “\/*?”<> []@#”	The output box says that the file cannot be created because one of the invalid filename characters has been entered. Output file saved successfully into logs and output box	

If I find more characters to break, I could always use a list of characters later.

V2 15/12/19

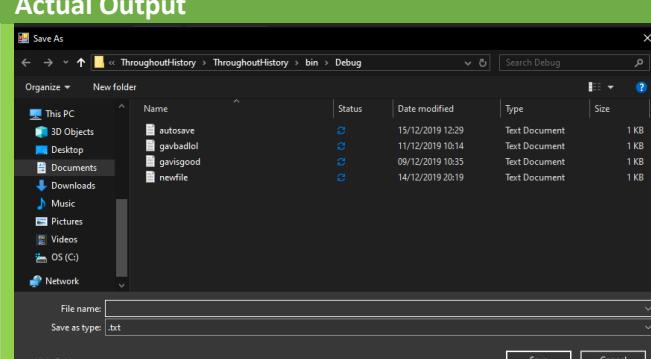
The code is far simpler now that most of the validation checking is done automatically by the save dialog box. The only complication was to have the dialog only open up when the user presses save button, and not when an autosave is made, because that would confuse the user and not make any sense. The solution I came up with means that the same function won't have to be written twice for button press and autosave events.

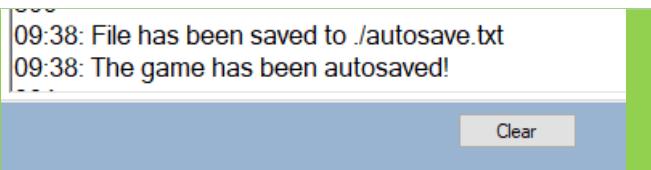
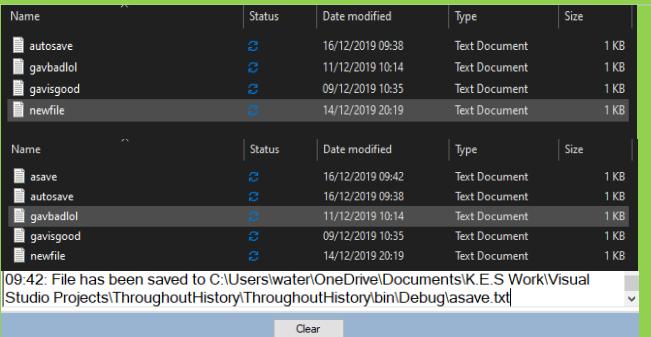
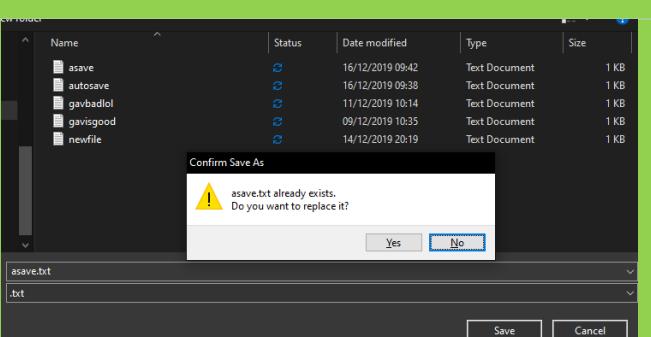
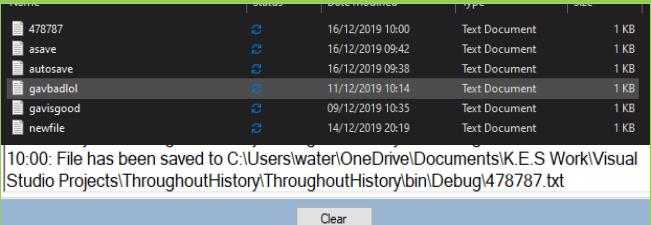
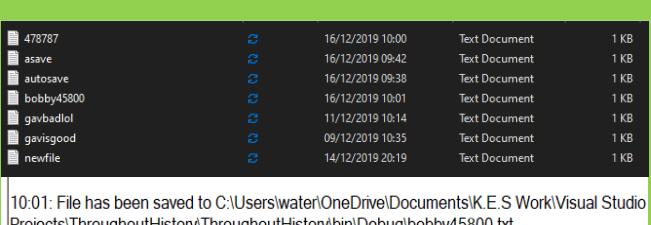
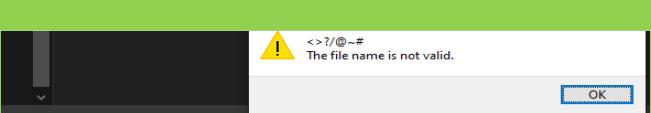
```

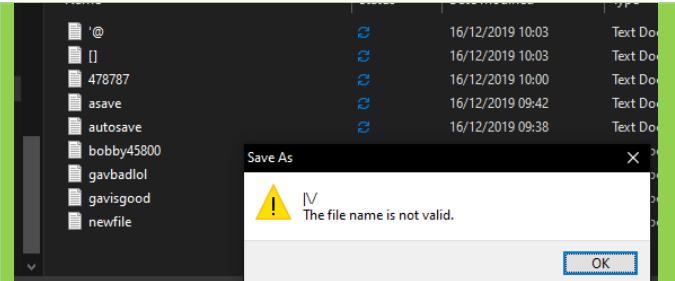
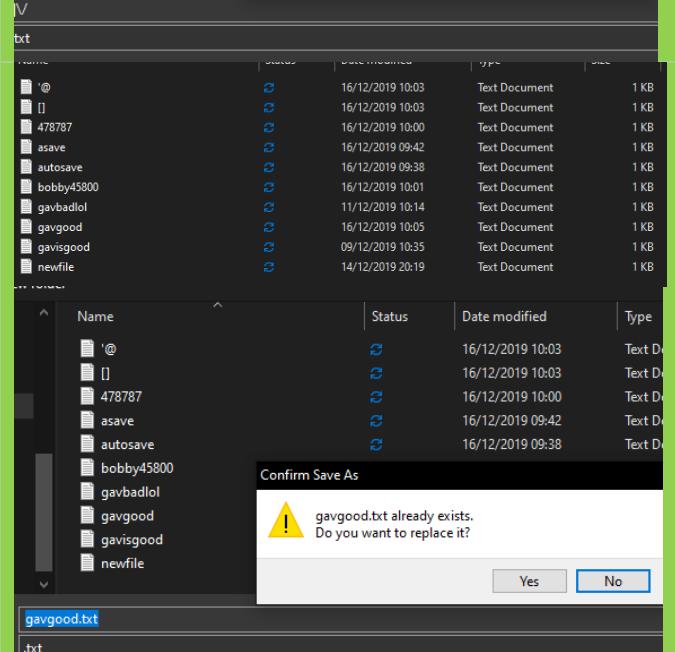
void saveToFile(bool isAutosave) {
    string selectedFile = "";
    bool goAhead = false;
    // If being called by autosave timer
    if (isAutosave) {
        selectedFile = "./autosave.txt";
        goAhead = true;
    } // Being called by button press
    else {
        // Set dialog filter to text files only
        saveDialog.Filter = ".txt|*.txt";
        // Open saving dialog
        DialogResult result = saveDialog.ShowDialog();
        // If user clicks OK button
        if (result == DialogResult.OK) {
            selectedFile = saveDialog.FileName;
            goAhead = true;
        }
    }
    if (goAhead) {
        // Stop global timer so that all values are paused at the correct
place
        globalTimer.Stop();
        // Now that validation is done, make new file in directory, open it
and write to it
        Stream cs = File.Create(selectedFile);
        using (StreamWriter sw = new StreamWriter(cs)) {
            // Writing data to file as seen previously
        }
        // Restart the global timer now that the saving is finished
        globalTimer.Start();
        rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": File
has been saved to " + selectedFile);
    }
}

```

Again, some new tests will have to be written to compensate for the new save system.

Input	Expected Output	Actual Output
Save button pressed	Save dialog box appears, only allows .txt files	

Autosave happens	Save dialog box does not appear, autosave.txt saved or overwritten to directory of executable	
Filename "asave" is entered and submit button pressed after save game button pressed	A new file is created in the executable file path called "asave.txt" with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename "asave.txt" is entered	A new file called "asave.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename "478787" is entered	A new file called "478787.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename "bobby45800.txt" is entered	A new file called "bobby45800.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename containing any of the following characters in its name is entered: "\/*?<> #{}"	The output box says that the file cannot be created because one of the invalid filename characters has been entered.	

	<p>Output file saved successfully into logs and output box</p>	
A file called "gavgood" is saved, and then the same name is entered for the next one	The file "gavgood" is saved, then the next time it is overwritten into the same file	

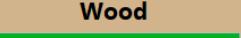
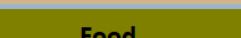
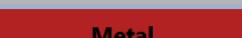
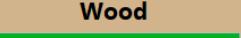
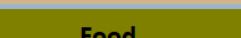
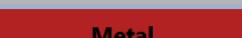
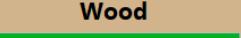
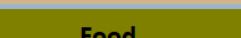
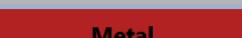
Loading from file

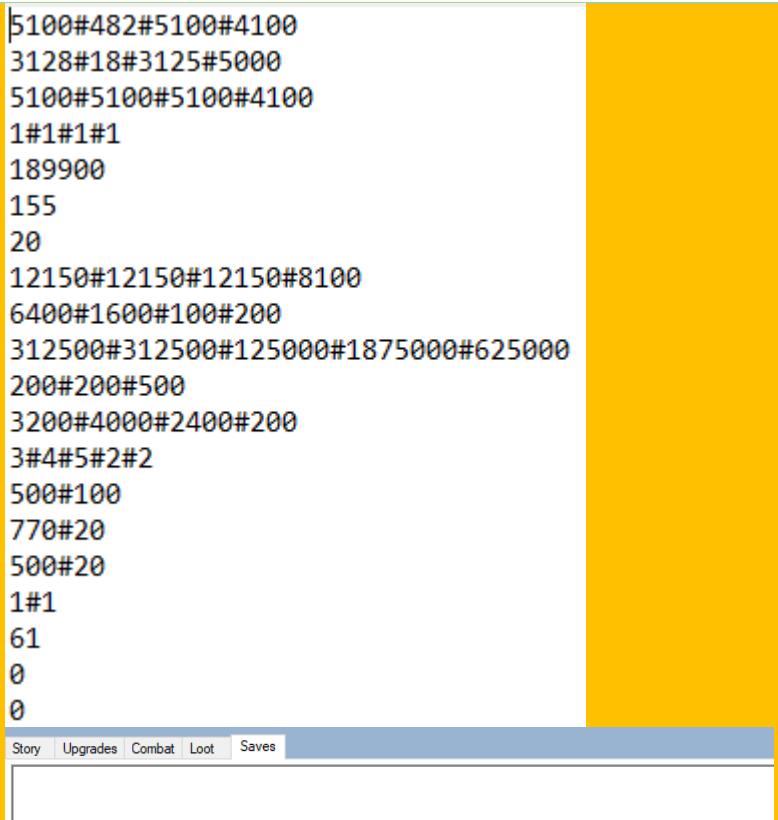
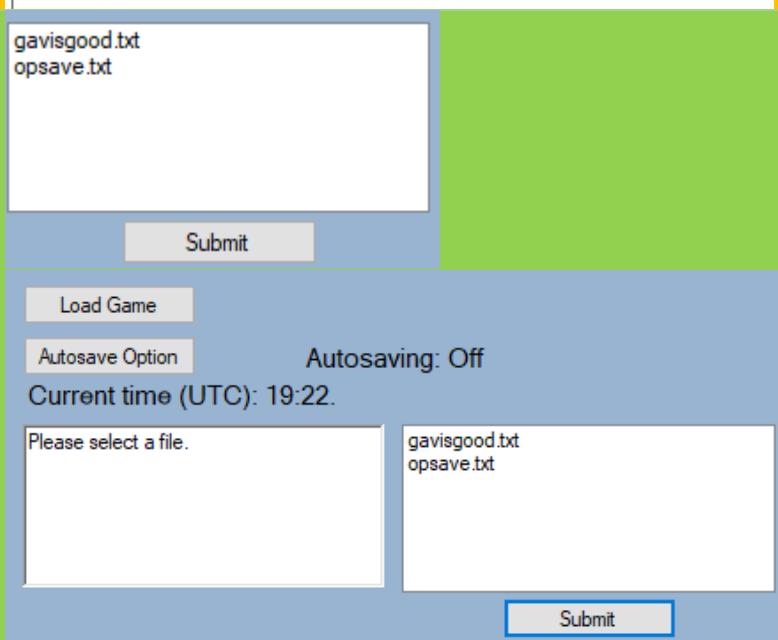
V1 09/12/19

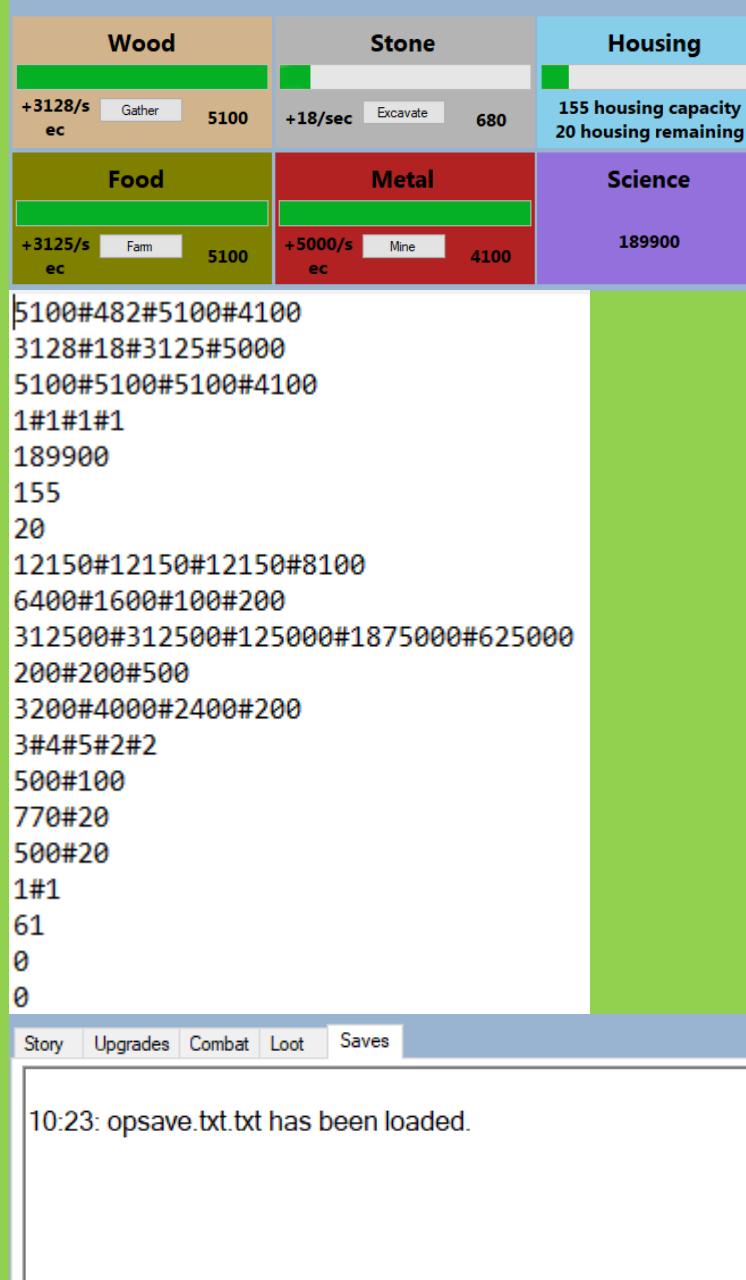
```
void loadFromFile(string fileName) {
    string selectedFile = ("./" + fileName);
    using (StreamReader sr = File.OpenText(selectedFile)) {
        string s = "";
        int lineNum = 1;
        // While not end of file
        while ((s = sr.ReadLine()) != null) {
            // Split each part into this currentSplits array
            string[] currentSplits = s.Split('#');
            // Write to global data variables based on which line is on
            switch (lineNum) {
                // Amount
                case 1:
                    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.resourcesData[0][i] = Convert.ToInt32(currentSplits[i]);}
                    break;
                // Rate
                case 2:
                    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.resourcesData[1][i] = Convert.ToInt32(currentSplits[i]);}
                    break;
                // Capacity
                case 3:
                    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.resourcesData[2][i] = Convert.ToInt32(currentSplits[i]);}
                    break;
                // Gather multiplier
                case 4:
                    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.resourcesData[3][i] = Convert.ToInt32(currentSplits[i]);}
                    break;
                // Science data
                case 5:
                    GlobalData.scienceData =
Convert.ToInt32(currentSplits[0]);
                    break;
                // Total housing
                case 6:
                    GlobalData.totalHousing =
Convert.ToInt32(currentSplits[0]);
                    break;
                // Housing remaining
                case 7:
                    GlobalData.housingRemaining =
Convert.ToInt32(currentSplits[0]);
                    break;
                // Storage costs
                case 8:
                    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.upgradesCosts[0][i] = Convert.ToInt32(currentSplits[i]);}
                    break;
                // Workers costs
                case 9:
                    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.upgradesCosts[1][i] = Convert.ToInt32(currentSplits[i]);}
                    break;
            }
        }
    }
}
```

```
// Science costs
case 10:
    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.upgradesCosts[2][i] = Convert.ToInt32(currentSplits[i]);}
    break;
// Housing costs
case 11:
    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.upgradesCosts[3][i] = Convert.ToInt32(currentSplits[i]);}
    break;
// Combat costs
case 12:
    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.upgradesCosts[4][i] = Convert.ToInt32(currentSplits[i]);}
    break;
// Cost multipliers
case 13:
    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.costMultipliers[i] = Convert.ToInt32(currentSplits[i]);}
    break;
// Health
case 14:
    for (int i = 0; i < currentSplits.Length; i++)
{GlobalData.combatData[0][i] = Convert.ToInt32(currentSplits[i]);}
    break;
// Block
case 15:
    for (int i = 0; i < currentSplits.Length; i++) {
GlobalData.combatData[1][i] = Convert.ToInt32(currentSplits[i]); }
    break;
// Damage
case 16:
    for (int i = 0; i < currentSplits.Length; i++) {
GlobalData.combatData[2][i] = Convert.ToInt32(currentSplits[i]); }
    break;
// No of troops
case 17:
    for (int i = 0; i < currentSplits.Length; i++) {
GlobalData.combatData[3][i] = Convert.ToInt32(currentSplits[i]); }
    break;
// Tick counter
case 18:
    GlobalData.tickCounter =
Convert.ToInt32(currentSplits[0]);
    break;
// Current war
case 19:
    GlobalData.curretWar =
Convert.ToInt32(currentSplits[0]);
    break;
// War number
case 20:
    GlobalData.warNumber =
Convert.ToInt32(currentSplits[0]);
    break;
// Year
case 21:
    GlobalData.year = Convert.ToInt32(currentSplits[0]);
    break;
```

```
// Actual year
        case 22:
            GlobalData.actualYear =
Convert.ToInt32(currentSplits[0]);
                    break;
// Era
        case 23:
            GlobalData.era = currentSplits[0];
                    break;
    }
    lineNum++;
}
}
```

Input	Expected Output	Actual Output						
Load game button is pressed	List box showing all possible files to load from appears with submit button for it. Save and delete games buttons hide	<p>Load Game</p> <p>Autosave Option</p> <p>Autosaving: Off</p> <p>Current time (UTC): 19:20.</p> <p>Please select one of the files to the right to load from.</p> <p>gavisgood.txt opsave.txt</p> <p>Submit</p>						
A file in the list box is selected and submit button is pressed	The game pauses whilst the information in the file is read from and put into the global data variables. Logs output that a file has been loaded into the game	<table border="1"> <tbody> <tr> <td style="background-color: #e0c0a0; padding: 5px;"> Wood  +3128/s ec Gather 5100 </td> <td style="background-color: #d0c0d0; padding: 5px;"> Stone  +18/sec Excavate 680 </td> <td style="background-color: #b0e0ff; padding: 5px;"> Housing  155 housing capacity 20 housing remaining </td> </tr> <tr> <td style="background-color: #a0c0e0; padding: 5px;"> Food  +3125/s ec Farm 5100 </td> <td style="background-color: #e0c0c0; padding: 5px;"> Metal  +5000/s ec Mine 4100 </td> <td style="background-color: #d0b0ff; padding: 5px;"> Science 189900 </td> </tr> </tbody> </table>	Wood  +3128/s ec Gather 5100	Stone  +18/sec Excavate 680	Housing  155 housing capacity 20 housing remaining	Food  +3125/s ec Farm 5100	Metal  +5000/s ec Mine 4100	Science 189900
Wood  +3128/s ec Gather 5100	Stone  +18/sec Excavate 680	Housing  155 housing capacity 20 housing remaining						
Food  +3125/s ec Farm 5100	Metal  +5000/s ec Mine 4100	Science 189900						

		<pre>5100#482#5100#4100 3128#18#3125#5000 5100#5100#5100#4100 1#1#1#1 189900 155 20 12150#12150#12150#8100 6400#1600#100#200 312500#312500#125000#1875000#625000 200#200#500 3200#4000#2400#200 3#4#5#2#2 500#100 770#20 500#20 1#1 61 0 0</pre>  <p>The screenshot shows a game interface with a character's stats listed on the left. Below the stats is a navigation bar with tabs: Story, Upgrades, Combat, Loot, and Saves. The 'Saves' tab is selected. A large empty area below the stats is likely a placeholder for a map or scene. At the bottom of the screen, there is a blue bar containing several controls: a list box with two items ('gavisgood.txt' and 'opsave.txt'), a 'Submit' button, a 'Load Game' button, an 'Autosave Option' section with a radio button set to 'Off', a status message 'Autosaving: Off', a timestamp 'Current time (UTC): 19:22.', a text input field with the placeholder 'Please select a file.', and another list box with the same two items ('gavisgood.txt' and 'opsave.txt'). A 'Submit' button is also located at the bottom right of this bar.</p>
Nothing is selected in the list box when the submit button is pressed	The output box tells the user to select a file to load from	 <p>The screenshot shows a game interface with a character's stats listed on the left. Below the stats is a navigation bar with tabs: Story, Upgrades, Combat, Loot, and Saves. The 'Saves' tab is selected. A large empty area below the stats is likely a placeholder for a map or scene. At the bottom of the screen, there is a blue bar containing several controls: a list box with two items ('gavisgood.txt' and 'opsave.txt'), a 'Submit' button, a 'Load Game' button, an 'Autosave Option' section with a radio button set to 'Off', a status message 'Autosaving: Off', a timestamp 'Current time (UTC): 19:22.', a text input field with the placeholder 'Please select a file.', and another list box with the same two items ('gavisgood.txt' and 'opsave.txt'). A 'Submit' button is also located at the bottom right of this bar.</p>

Input	Expected Output	Actual Output
A file in the list box is selected and submit button is pressed	The game pauses whilst the information in the file is read from and put into the global data variables. Logs output that a file has been loaded into the game	 <p>The screenshot shows a game interface with resource management panels and a log window.</p> <p>Resource Panels:</p> <ul style="list-style-type: none"> Wood: +3128/sec Gather 5100 Stone: +18/sec Excavate 680 Housing: 155 housing capacity 20 housing remaining Food: +3125/sec Farm 5100 Metal: +5000/sec Mine 4100 Science: 189900 <p>Logs:</p> <pre> 5100#482#5100#4100 3128#18#3125#5000 5100#5100#5100#4100 1#1#1#1 189900 155 20 12150#12150#12150#8100 6400#1600#100#200 312500#312500#125000#1875000#625000 200#200#500 3200#4000#2400#200 3#4#5#2#2 500#100 770#20 500#20 1#1 61 0 0 </pre> <p>Saves Tab:</p> <p>Story Upgrades Combat Loot Saves</p> <p>10:23: opsave.txt.txt has been loaded.</p>

V2 14/12/19

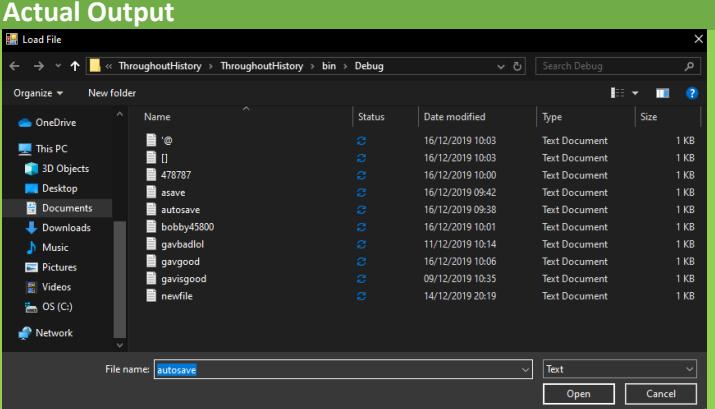
My stakeholders have said that they want a file dialog to open so that they can locate a save file from wherever on their computer system they have stored it. New code:

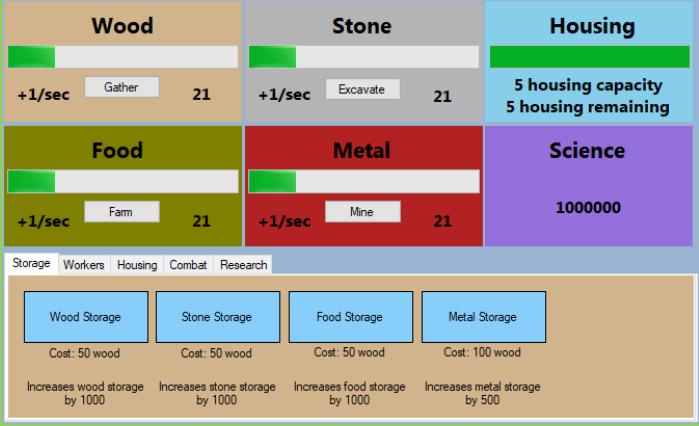
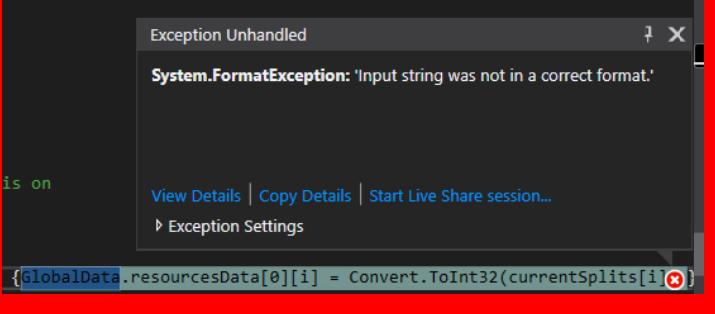
```

void loadFromFile() {
    // Set filter to only text files
    openDialogue.Filter = "Text|*.txt";
    // Show dialog
    DialogResult result = openDialogue.ShowDialog();
    // If user presses OK button
    if (result == DialogResult.OK) {
        // Get file to load
        string selectedFile = openDialogue.FileName;
        // Load data from file
        using (StreamReader sr = File.OpenText(selectedFile)) {
            string s = "";
            int lineNumber = 1;
            // While not end of file
            while ((s = sr.ReadLine()) != null) {
                // Split each part into this currentSplits array
                string[] currentSplits = s.Split('#');
                // Write to global data variables based on which line
                is on
                switch (lineNumber) {
                    // Loading from file code
                }
                lineNumber++;
            }
        }
        // Output to logs
        rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ":" + selectedFile + ".txt has been loaded.");
    }
}

```

Again, some new tests have to be written:

Input	Expected Output	Actual Output
Load game button pressed	Open save dialog appears	

<p>Load game button pressed, file “autosave.txt” is selected and submit button pressed</p>	<p>Open save dialog appears, on submit button press, load all data from “autosave.txt” and output to logs</p>	<pre>20#20#20#20 1#1#1#1 100#100#100#100 1#1#1#1 1000000 5 5 50#50#50#100 100#100#100#200 100#500#1000#3000#5000 50#100#500 200#500#300#200 3#4#5#2# 100#100 20#20 50#20 1#1 20 0 0 -500 500 8C</pre>  <p>Note: Value was 20 each resource, but in time it took for me to pause game for screenshot, resource count had incremented by 1.</p>
<p>Load game button pressed, file “autosave.txt” is selected and cancel button pressed</p>	<p>Open save dialog appears, on cancel button press, dialog closes and nothing happens</p>	
<p>Load game button pressed, file “shoppinglist” is selected and submit button pressed</p>	<p>Open save dialog appears, on submit button press, dialog closes and nothing happens (because file is invalid to load from)</p>	

For context, “shoppinglist.txt” looks like:

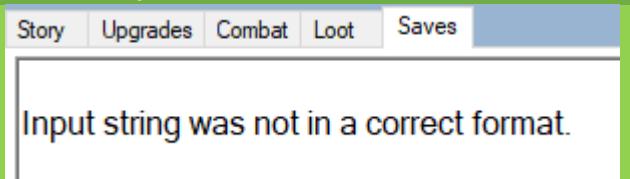
```
Apples
Bacon
Chocolate Bar
24 Diamonds
```

Obviously, the program is unable to handle this because it is not in the correct format of a save file. I have 2 solutions for this:

1. Put a little “code” at the top of a save file to mark it as the game’s “save” file. Then when the file is loaded, the program will check if the “code” is there. If it is not, stop trying to load the file. The issue with this is that a random text file could be created with this “code” at the top and random stuff underneath in order to fix it (or somehow by chance).
2. Put the loading code into a try catch box, so that when the invalid format error occurs, this will catch that, close the dialog box and the program will not crash.

The latter solution is the best here.

```
try {
    // Loading code here
    // Output to logs
    rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ":" + 
selectedFile + ".txt has been loaded.");
} catch (Exception ex) {
    rtxtSaves.Text += ("\n" + ex.Message);
}
```

Input	Expected Output	Actual Output
Load game button pressed, file “shoppinglist” is selected and submit button pressed	Open save dialog appears, on submit button press, dialog closes and nothing happens (because file is invalid to load from)	 Story Upgrades Combat Loot Saves Input string was not in a correct format. Dialog box closed after file selected, nothing else happened.

Stakeholder Feedback

Now that I have written all of this code and fixed the bugs, I have given the program to each of my stakeholders to test each of the parts – delete files, autosaving, saving files, loading files. There is one query all of my stakeholders had: “Why do we have to select a file to load/save to/delete from a list box? Why can’t it be from a file dialog like most other applications?”. So therefore, I have decided to change some of the parts in order to use a file dialog to fit my stakeholders’ needs. The advantages of this over using a list box is:

1. Automatic built in features from the .NET windows forms tools e.g. search, filter etc
2. The user can search their entire file system to load from any location they have stored it (the previous system only displays files from the same directory as the application executable)
3. Better user experience in general as it looks far more professional (as it is a prebuilt tool) than just a list box and a button

I will need to redesign the GUI a little bit as well as the code to make way for this new and improved system.

These changes are shown in the V2 parts of each of the previous listed sections.

Balancing [E]

Now that the game mechanics and various systems are finished, I need to balance the game not only so that it actually allows the user to progress, but also have fun and feel somewhat rewarded of their game decisions.

The first part that I have noticed is definitely an issue, is that by the time the wood storage upgrade costs 12800 wood, the maximum amount of wood that the player can have is only 8500. Here, I have opted to decrease the cost multipliers for the storage upgrades, from 3 to 2. Additionally, I have noticed similar issues with the other parts of the upgrades, thus I have decreased the housing and combat multiplier costs from 3 to 2x.

I think that early game, progress is very slow and quite boring. It takes a long time just to get enough wood for a wood storage upgrade and food for more wood (to upgrade the food storage so more wood/food workers can be bought). Therefore, I will try increasing the added rate from toggling the button to +3/sec, and I also want to increase the worker rate from +1/sec to +5/sec for food and wood workers.

After playing for 10 minutes, I think that the first war should happen much sooner than the first 5 minutes in. This is because perhaps the user should be introduced to the war system quickly enough to grab their attention before they assume that they might have to unlock a certain upgrade or get to a resource milestone to start a war. Therefore, I have decided to get the first war started after only 60 ticks or 1 minute.

Review 4

- Green – Feature finished (does not matter if there is a minor error or change)
- Orange – Some of feature is there, some is not (**note that this is a different key to the previous review stages**)
- Red – Feature skipped/not going to be worked on

Criteria	How to evidence criteria being met	Section Code
Section A: Resources/upgrades system		
Resources go up over time	2 screenshots – first one taken before, second one taken after to show progress with resources	A1
Resource rate increased through upgrade	Screenshot of increased rate	A2
Resources reach “milestone” where the number shortens	Screenshot of 4,000 resource shown has 4k resource	A3 <i>Optional</i>
Button to manually increase rate of resource collection increased, button turns brown, all other buttons turn grey and toggle off, that resource rate increases by 10%	2 screenshots – first one taken before with one button pressed, second one taken after with another button pressed to show increased rate and only one rate bonus allowed at a time	A4 <i>Optional</i>
[Resource] storage upgrade is bought	Screenshots to show max [resource] capacity increased	A5
Worker for [resource] upgrade is bought, not enough housing space	Screenshot to show “Not enough housing space” in tutorial logs	A6
Worker for [resource] upgrade is bought, enough housing space	Screenshot to show increased [resource] rate	A7
Housing upgrade is bought	Screenshot to show housing number increased	A8
Science upgrade is bought, but not enough science points available	Screenshot to show “Not enough science to buy this” in tutorial logs	A9
Science upgrade for combat is bought, new combat upgrade button appears in combat tab	Screenshot to show new upgrade button	A10
Different upgrade types tabs selected	Screenshot to show only worker upgrades shown in workers tab	A11 <i>Optional</i>
Combat upgrade to increase block bought	Screenshot to show troops’ increased block	A12
Enough resources are available to buy upgrade	Screenshot to show upgrade button turning to “clickable” state	A13 <i>Optional</i>
Section B: Save/load system		
Game is auto-saved every 2 minutes	3 screenshots – first one taken when auto-save happens, second one taken 2 minutes later when second auto-save happens, third taken of auto-	B1

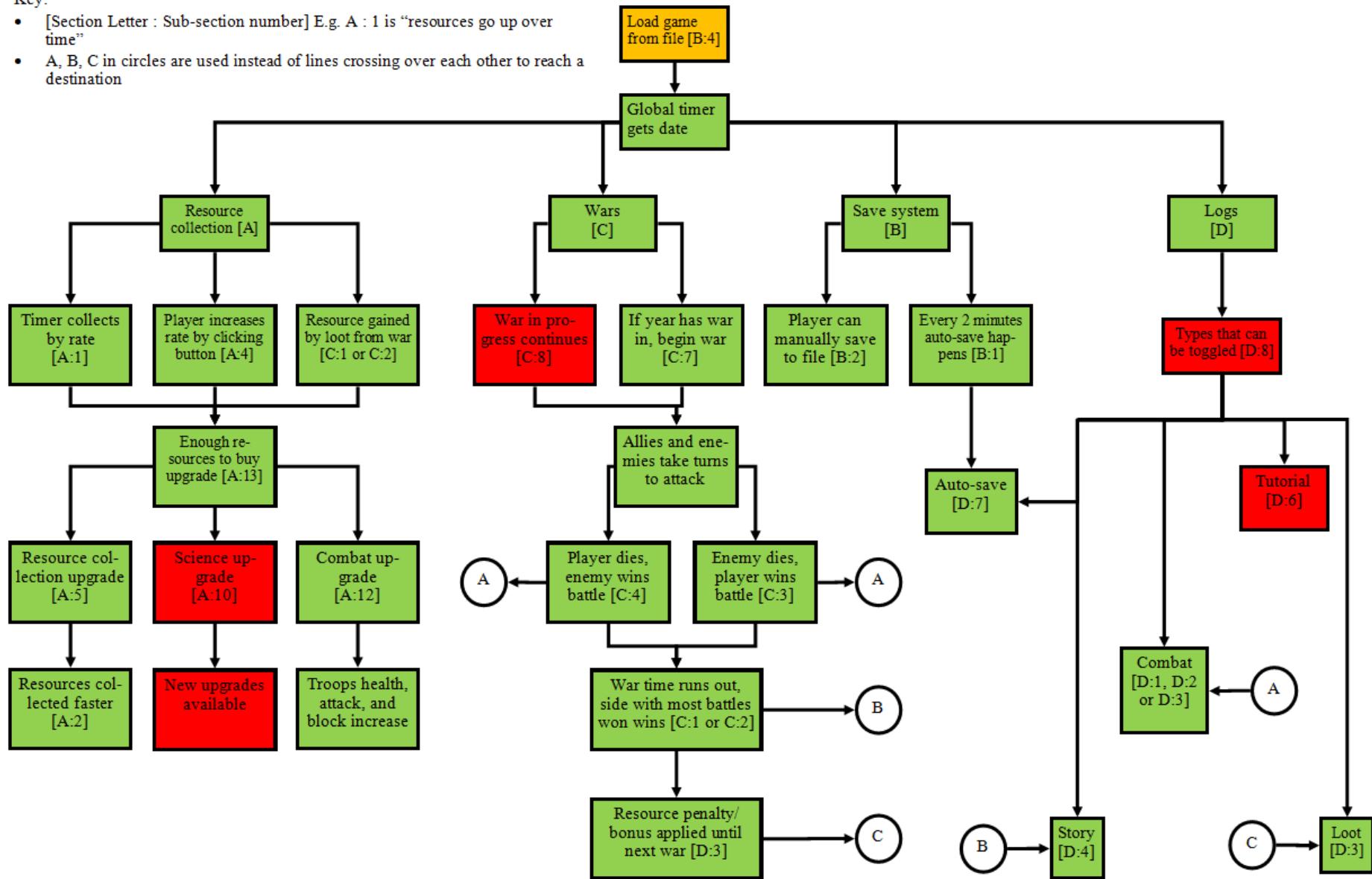
	save file changed (see time stamp in file)	
Player manually presses the save game button	Screenshot of new save file created	B2
Game closed	Screenshot of auto-save file changed	B3
Game loaded up, player selects save file to load, message box tells player how many resources were made when they were offline	Screenshot of before game closed, screenshot of after game reopened and save file loaded, screenshot of message box	B4
Section C: Combat/war system		
War is won by player	Screenshot of resources rate gain, and screenshot of grid reset	C1
War is lost by player	Screenshot of resources rate loss, and screenshot of grid reset	C2
Battle is won by player	Screenshot of player gaining green tile from enemy's red tile	C3
Battle is lost by player	Screenshot of enemy gaining red tile from player's green tile	C4
Player's army with 170 health and 50 block gets hit by 100 attack, so health goes down to 120	Screenshot before and after turn to show health/block/attack works	C5
Number of troops increased, total army health increases by (current upgrade of health) * number of new troops	Screenshot to show the stats of army increased	C6
New war started when right year reached, enemies have higher health/attack/block than in previous war	Screenshot to show stats in old war, and then stats in new	C7
Game loaded up, war in progress continues as before	Screenshot showing war before game close, after game close	C8
Section D: Logs system		
A battle is lost or won, show the message in the combat logs	Screenshot of combat logs	D1
A war is lost or won, show the statistics in the combat logs	Screenshot of combat logs	D2 <i>Optional</i>
A war is lost or won, logs show resource penalty/bonus until next war	Screenshot of loot logs	D3
A tech upgrade is made, and story progresses, show message in logs	Screenshot of story logs	D4
A new year begins, print new year in logs	Screenshot of story logs	D5
Player battles for first time, buys first upgrade or reaches first	Screenshot of tutorial logs	D6

resource milestone, print various tips or explanations in tutorial logs		
Game is auto-saved or manually saved by user, show message in auto-saves logs	Screenshot of auto-save logs	D7
A log type button is clicked to toggle it off, button changes from green to red and those logs stop showing	Screenshot of log toggle buttons and logs to show right logs toggled off	D8
The clear logs button is clicked, all logs toggled on cleared	Screenshot of nothing in logs	D9
Save logs to file button is clicked, all logs saved to a text file	Screenshot of file it saved to	D10 <i>Optional</i>
Section E: Usability		
The game balanced so that it can't be progressed really quickly (amount upgrades cost, how much the affect various parts of the game etc.)	Screenshots of player being able to keep up with enemies' combat stats, to show balanced progression through time	E1 <i>Optional</i>
Intuitive and easy to use menu	All of the game menus can be traversed within 2 clicks of the main game screen. Screenshots showing each menu	E2
Game does not have low framerate	Screenshot of CPU load reduction when game closed	E3
Big buttons, some colour coding to show toggles and being able to be clicked	Screenshots of buttons changing colour/position when various buttons pressed	E4 <i>Optional</i>

I have managed to do everything but three criteria. A3 was optional, but B4 and A10 not. Talking about the offline progress system in B4, I think having it as not optional criteria is silly because it would have been a very difficult task and make my program far more complex than it needs to be. However, **note that the first part (loading files) is still a successful part of the criteria, and should have been two different criteria in itself.** A10 could have been met but would have added a fair bit of program complexity.

Key:

- [Section Letter : Sub-section number] E.g. A : 1 is “resources go up over time”
- A, B, C in circles are used instead of lines crossing over each other to reach a destination



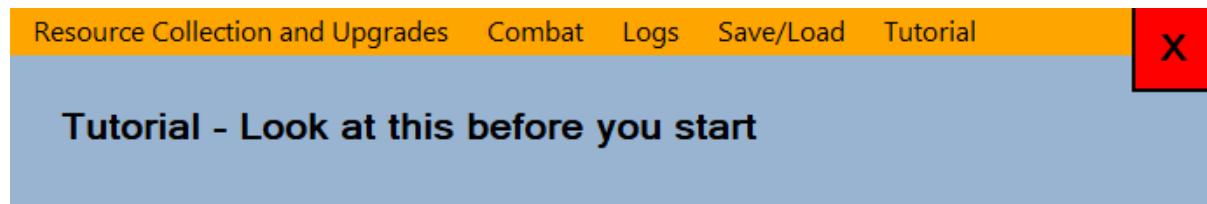
4. Evaluation

Walkthrough of Solution

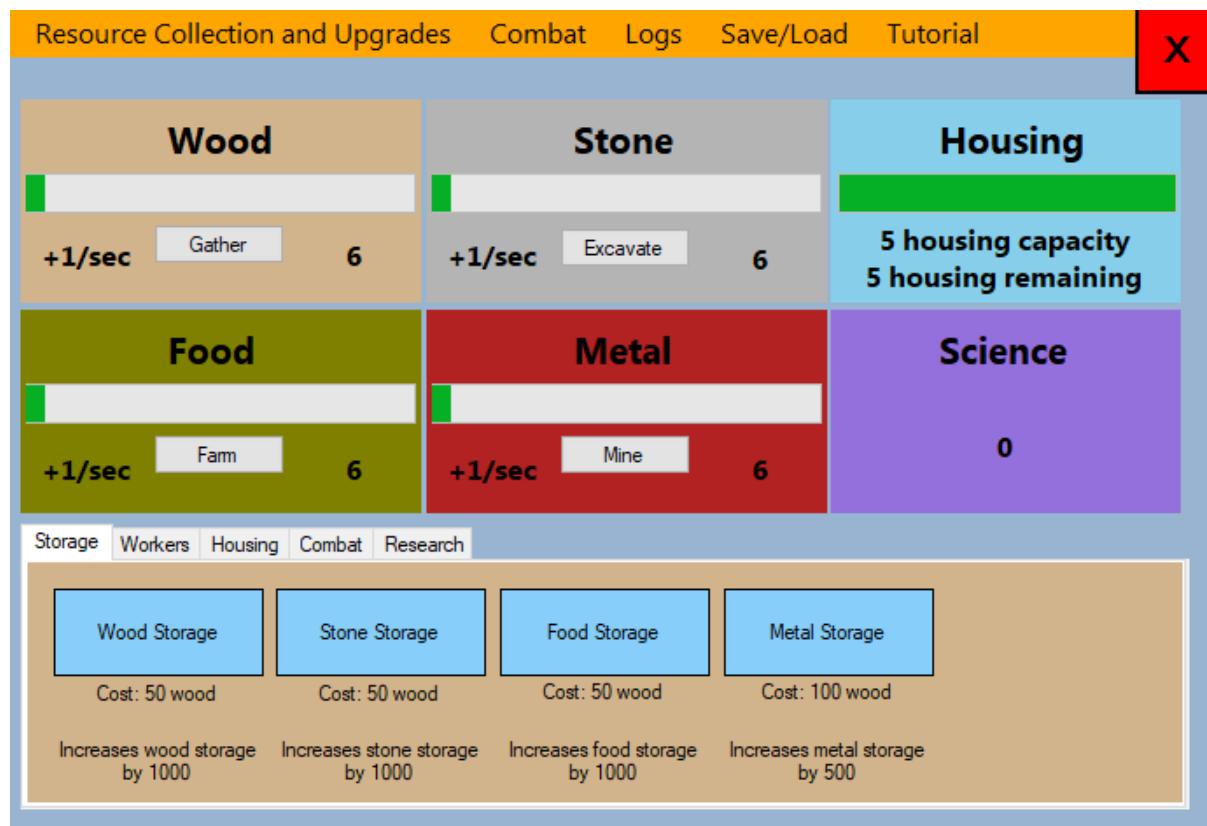
Before I get into showing the evidence of the success criteria, I will do a little “walkthrough” of the game.

When you first load up the game, you are first shown the tutorials menu (which is currently empty because it could be done in future development as talked in usability testing). The user can also move the form around the screen by just clicking and holding on one of the blue parts and dragging the form around.

(Side note: the game form size never actually changes – I have cropped some of the empty parts out to make the screenshots smaller).

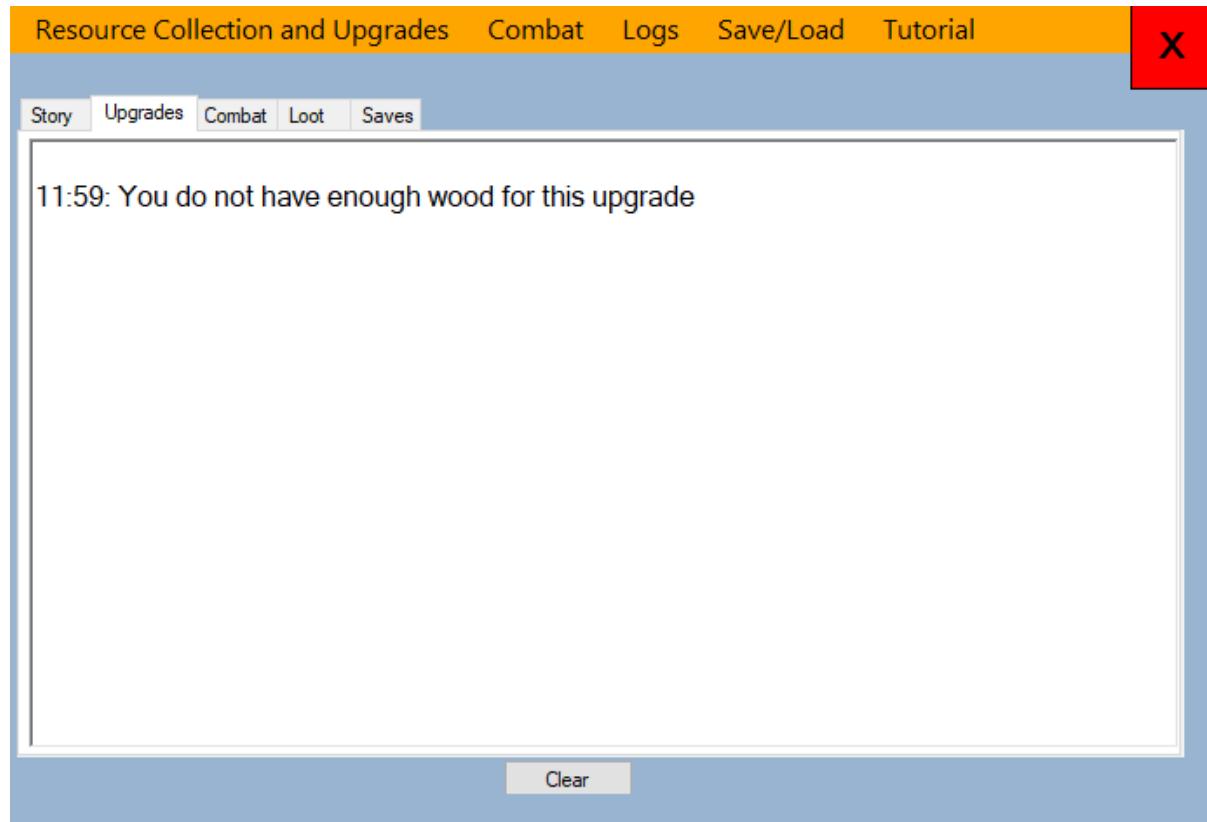


The player then has 2 options – click on one of the menu strip tabs along the orange bar, or click the big red “X” button to close the program (this will also autosave the game). The first thing they will eventually end up on is resource collection and upgrades, as that menu has the only stuff they can do at the moment.

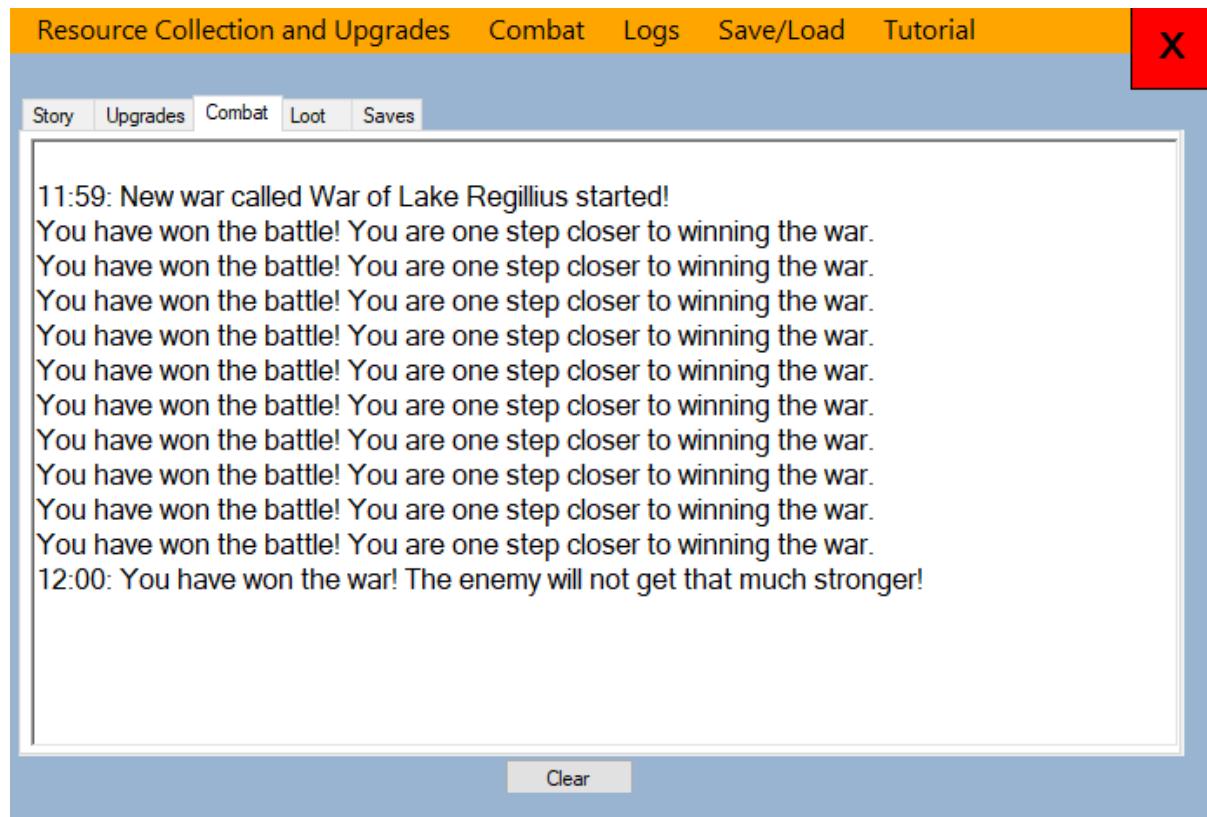


So they click buttons to see what happens. The “gather” button allows them to collect wood faster. The indication that a button is pressed is shown by the button turning chocolate colour.

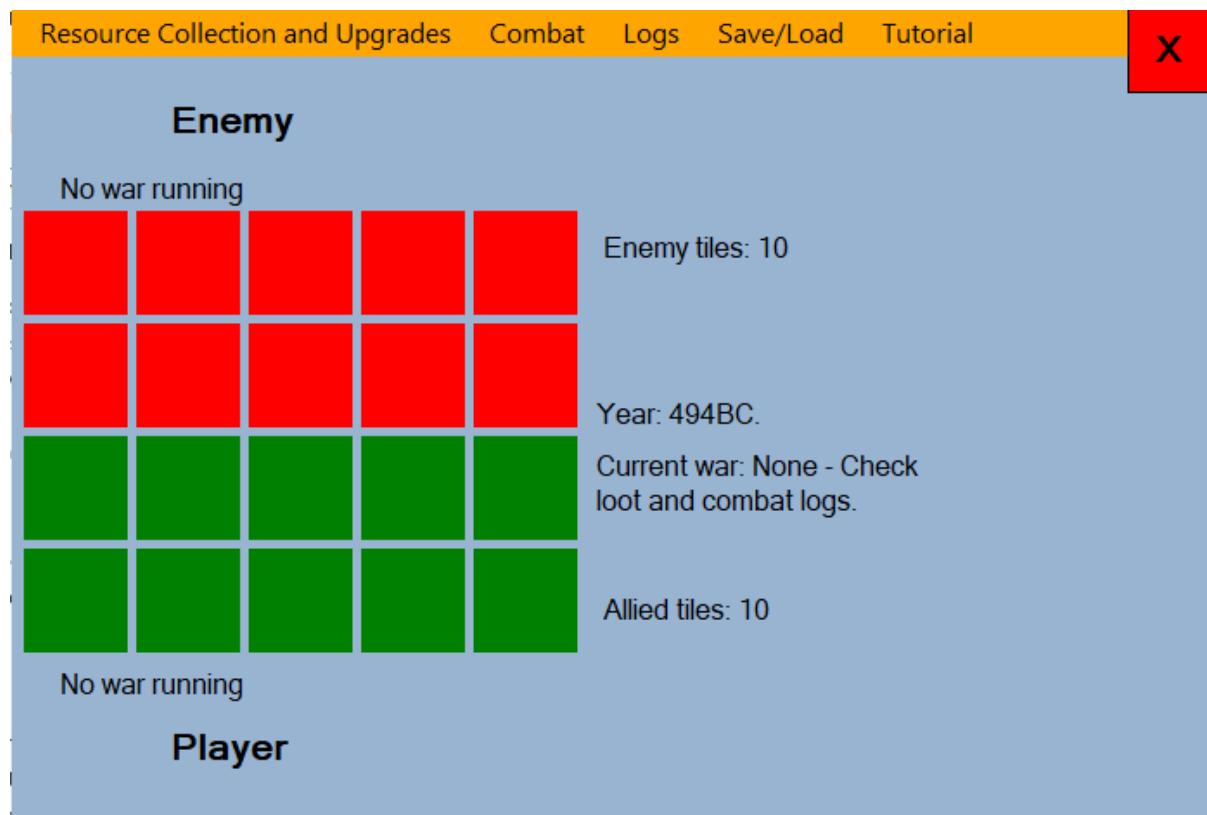
The player will mess with stuff to see what they can do, and perhaps build up loads of resources over time. At some point, they might press an upgrade button without having the resources and wonder why nothing is happening. They might see the logs tab and see if anything is sent there.



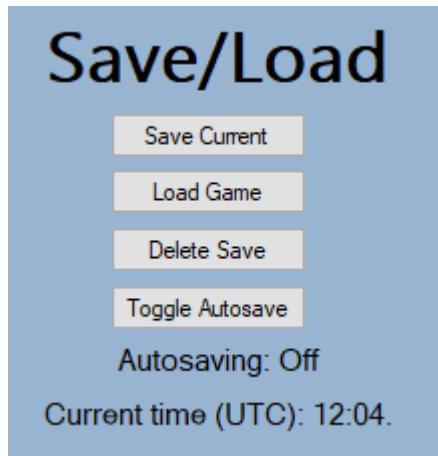
They may look around the logs tabs and see the combat tab, and perhaps anything inside it if a war has already started.



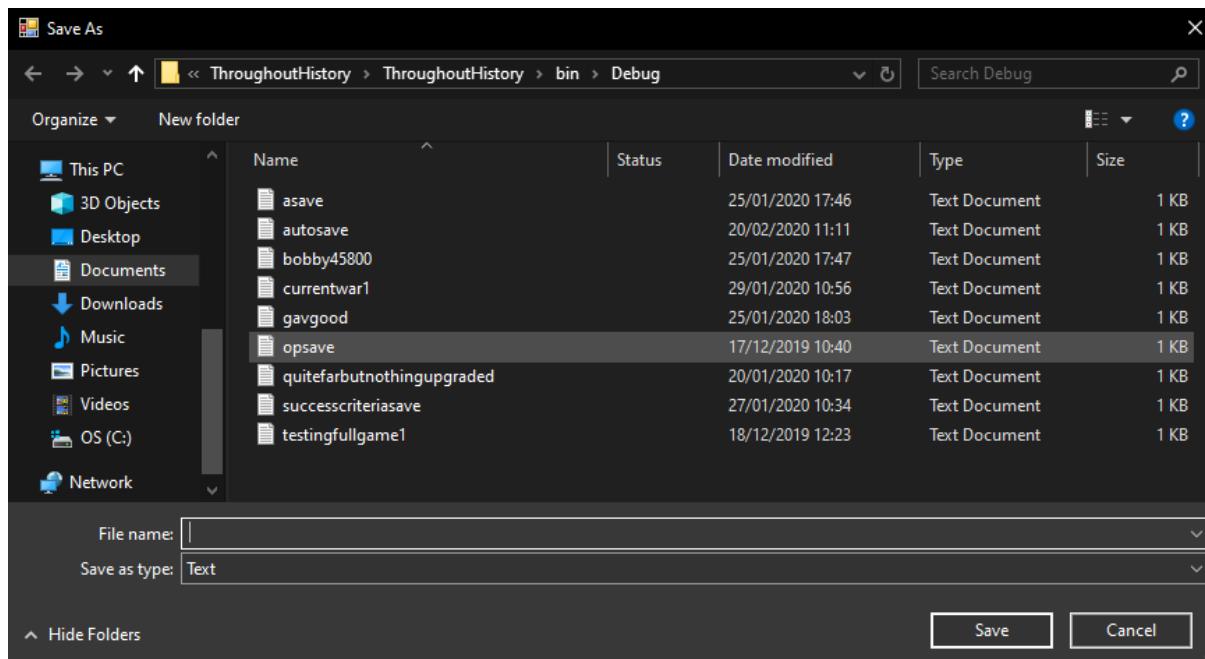
And then consequently look at the combat tab.



If the player then wants to leave the game and not lose their progress, they can either just exit the game and load up the autosave on next load, or save a game manually, through the saves menu.

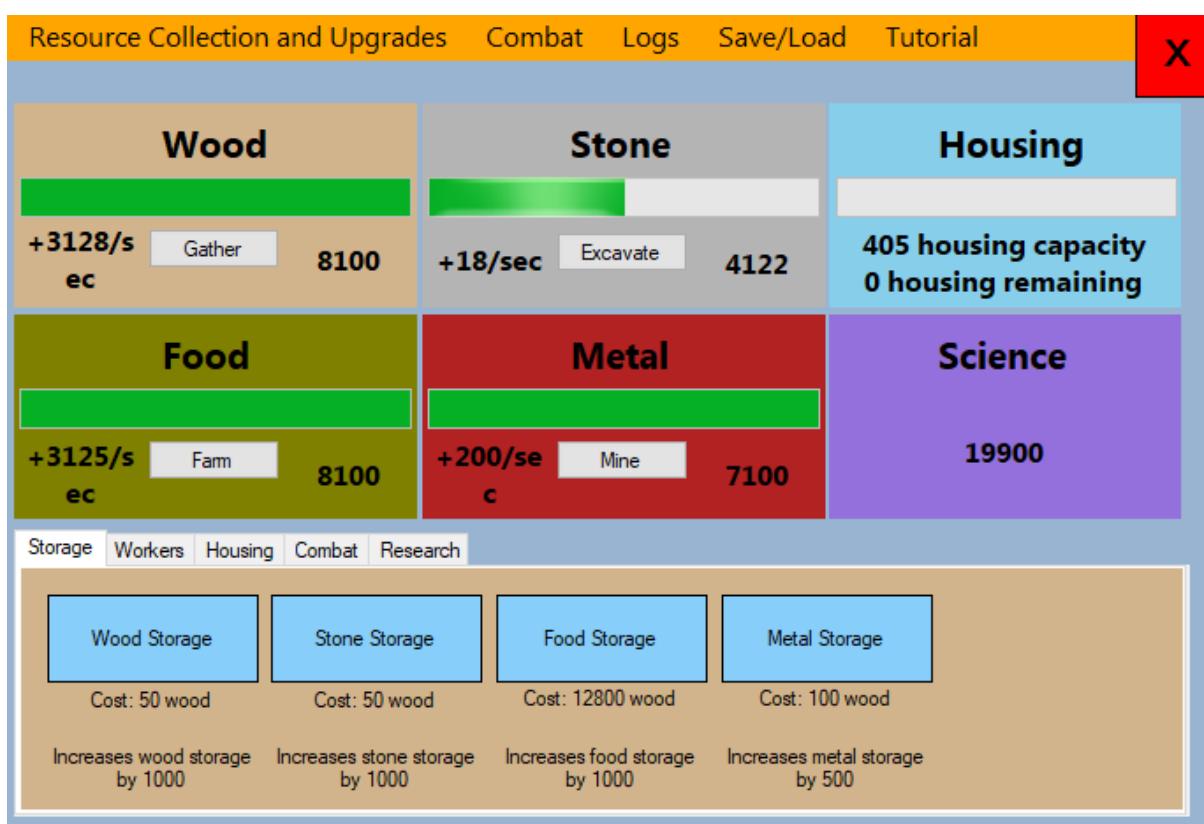
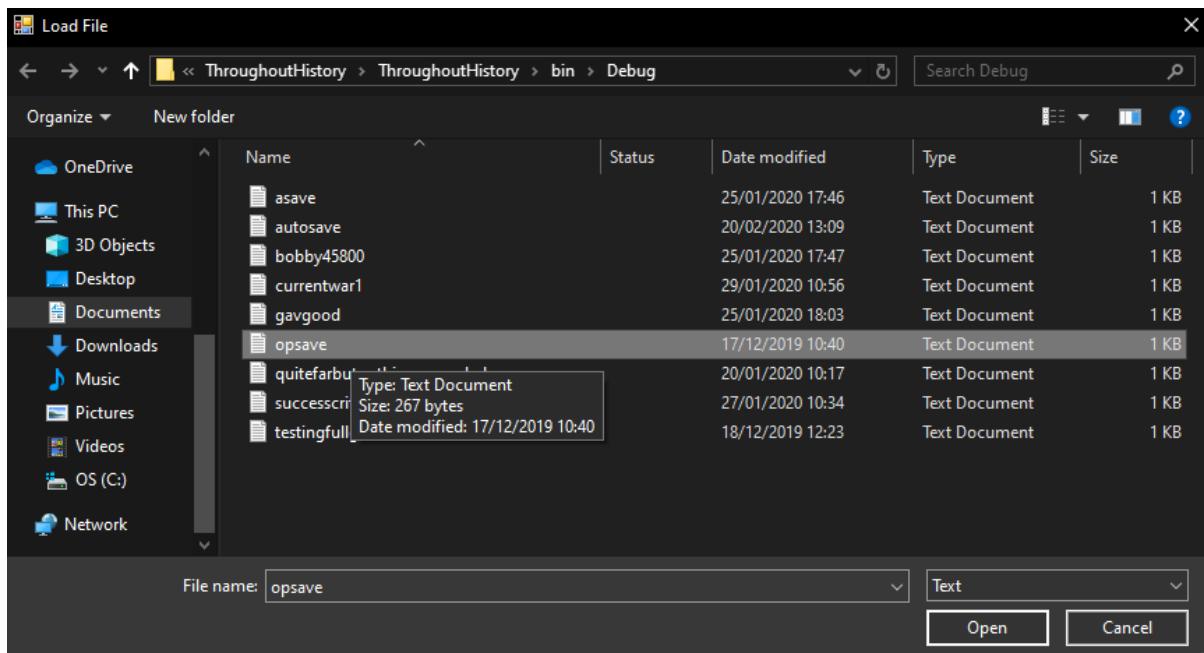


Let's say they save their current game, this save dialog appears:



They can then save their game, in any location they want, with any normally supported file name, or overwrite an already existing file.

Now they want to load, say, "opsave" which is where every upgrade is bought 5 times each and the current year is 496BC.



Beta Testing

Usability Tests

Each stakeholder will test this, then I will take the feedback and do any last edits to the prototype game to fit their needs.

Question
Can they switch between tabs easily?
Can they press a button to toggle faster rate for that resource?
Can they press it again to toggle it off?
Can they buy a storage upgrade if they have enough wood?
Can they buy a worker upgrade if they have enough food?
Can they buy a housing upgrade if they have enough stone?
Can they buy a combat upgrade if they have enough metal?
Can they buy a science upgrade if they have enough science points?
First war starts, can they work out what is going on in the combat menu?
If they do not have enough of a resource for an upgrade, can they easily find out why they cannot buy the upgrade from logs?
If they win/lose/draw a war, can they find the loot gained/outcome in logs?
If they save, load or the game autosaves, can they find the messages in logs?
Can they save the game to any file location?
Can they load the game from any file location?
Can they delete a save?
Can they toggle autosaving on or off?

Stakeholder Feedback

Stakeholder 1:

Question	Can they do it?
Can they switch between tabs easily?	Yes
Can they press a button to toggle faster rate for that resource?	Yes, but I didn't understand what it does easily
Can they press it again to toggle it off?	Yes
Can they buy a storage upgrade if they have enough wood?	Yes
Can they buy a worker upgrade if they have enough food?	Yes
Can they buy a housing upgrade if they have enough stone?	Yes
Can they buy a combat upgrade if they have enough metal?	Yes
Can they buy a science upgrade if they have enough science points?	Yes
First war starts, can they work out what is going on in the combat menu?	Took a little explaining, but was understandable after 30-60 secs
If they do not have enough of a resource for an upgrade, can they easily find out why they cannot buy the upgrade from logs?	Yes, although nothing told me to check logs
If they win/lose/draw a war, can they find the loot gained/outcome in logs?	Yes

If they save, load or the game autosaves, can they find the messages in logs?	Yes
Can they save the game to any file location?	Yes
Can they load the game from any file location?	Yes
Can they delete a save?	Yes
Can they toggle autosaving on or off?	Yes

"The game is good once I understood what was going on but it took a bit of explaining so I think there should be some sort of tutorial or tooltip system when hovering over things. Logs are also a bit hidden away. Combat is good."

Stakeholder 2:

Question	Can they do it?
Can they switch between tabs easily?	Yes, though not always obvious which tab I'm in – an indication would be good
Can they press a button to toggle faster rate for that resource?	Yes, though a little disconcerting that the button takes a second to become highlighted after clicking it, as it isn't instant
Can they press it again to toggle it off?	Same as above
Can they buy a storage upgrade if they have enough wood?	Yes. An indication as to whether buying the upgrade was a success might be useful, as well as some visual display on the button as to whether I currently have enough before clicking
Can they buy a worker upgrade if they have enough food?	
Can they buy a housing upgrade if they have enough stone?	
Can they buy a combat upgrade if they have enough metal?	
Can they buy a science upgrade if they have enough science points?	My science doesn't appear to be increasing at all. Edit: I tabbed back in and the science had gone up to 5000, perhaps an explanation of where science comes from might be good? Edit2: It says it in the war menu to check the logs but that kind of important info feels way too far away.
First war starts, can they work out what is going on in the combat menu?	I didn't realise a war had started – I was in the resource collection menu. After happening to click that tab I saw the war going on but it seemed a bit disconnected
If they do not have enough of a resource for an upgrade, can they easily find out why they cannot buy the upgrade from logs?	I never thought to check the logs for info like that
If they win/lose/draw a war, can they find the loot gained/outcome in logs?	Eventually. The message to check logs doesn't really stand out amongst all the other info so I missed it originally
If they save, load or the game autosaves, can they find the messages in logs?	Yes
Can they save the game to any file location?	Yes

Can they load the game from any file location?	Yes
Can they delete a save?	Yes
Can they toggle autosaving on or off?	Seems so, though 'Autosave Option' is a weird name for a button, since it isn't really an action. Maybe 'toggle autosave'?

"The game is fun but there is not much content yet and there is no information on any aspect of the game that tells you what different parts do, and by the time I worked out how some parts like war worked, the first war had already begun. I think that logs should be on every screen and all the different types appear on the same area, because currently they are disconnected from the rest of the game."

Stakeholder 3:

Question	Can they do it?
Can they switch between tabs easily?	Yes
Can they press a button to toggle faster rate for that resource?	Yes
Can they press it again to toggle it off?	Yes
Can they buy a storage upgrade if they have enough wood?	Yes
Can they buy a worker upgrade if they have enough food?	Yes
Can they buy a housing upgrade if they have enough stone?	Yes
Can they buy a combat upgrade if they have enough metal?	Yes
Can they buy a science upgrade if they have enough science points?	Yes
First war starts, can they work out what is going on in the combat menu?	Missed the first war
If they do not have enough of a resource for an upgrade, can they easily find out why they cannot buy the upgrade from logs?	Many steps to go to logs and select correct tabs, not immediately obvious
If they win/lose/draw a war, can they find the loot gained/outcome in logs?	Yes
If they save, load or the game autosaves, can they find the messages in logs?	Yes
Can they save the game to any file location?	Yes
Can they load the game from any file location?	Yes
Can they delete a save?	Yes
Can they toggle autosaving on or off?	Yes but the button name isn't very clear of what it does

"The war was confusing because of a lack of explanations of alerts but is an overall interesting game. The rest of the game was fairly intuitive, the GUI is quite good being colour coded and segmented into different windows".

Overall Feedback

The main feedback I've got is that getting going with the game is difficult, and it takes a while to work out how different parts work. Most have said they want some sort of tab with a tutorial section, maybe with some text and a couple of screenshots to help explain it.

Another part that needs working on a bit is logs – when the player cannot buy an upgrade because they do not have enough resources, they are sometimes confused as to why and the logs are a tucked away area to show this. Another point when this is an issue is when a war ends, the player is usually confused where the information about the war outcome is. On the first war, 1 minute into the fresh game, the player does not yet understand much about the game and how any of it works. Here, I think the underlying issue is still the lack of any tutorial area.

When a war is won, and the player is rewarded troops as part of the loot, every single stakeholder asked where they can see the number of troops that they have. This is a clear oversight because they should be easy to find so that the player stats can be easily worked out.

When a resource collection button is clicked, there is “lag” which is actually just waiting for the tick to pass for it to update. This is because the button colours are changed within the resource collection menu which is called every tick.

It is also unclear when the first war starts – every stakeholder was confused about how they obtained science and how they saw the wars (they all missed the first war 1 minute in).

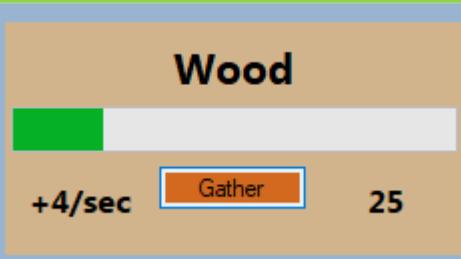
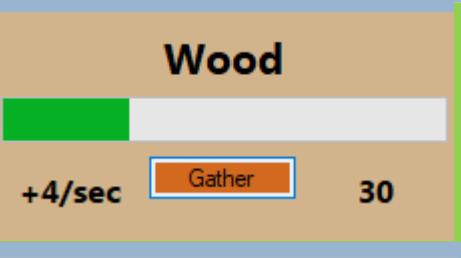
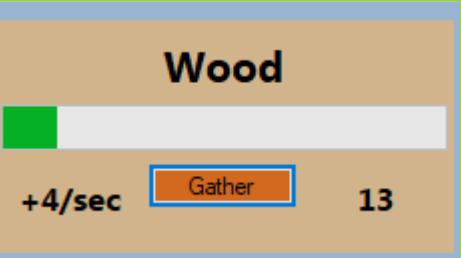
Finally, the autosave button has an odd name so should be changed to something nicer.

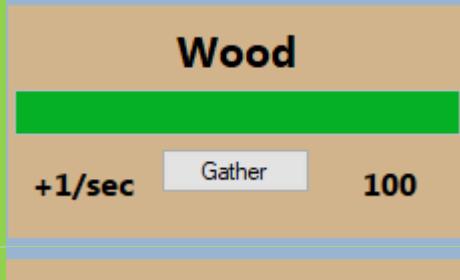
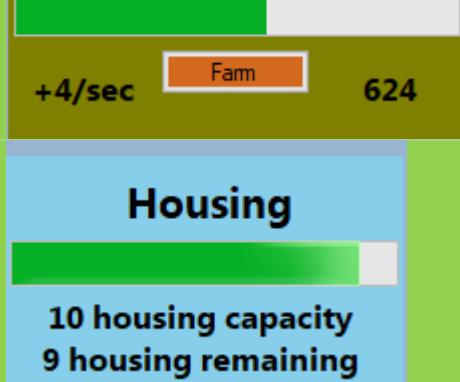
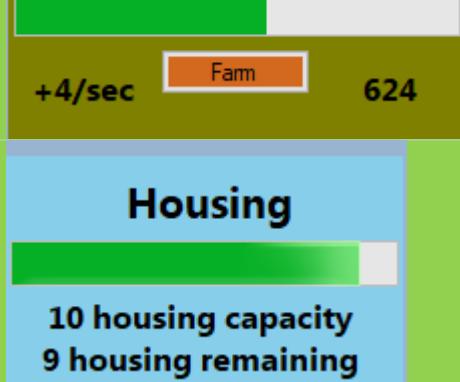
This feedback could be implemented through future development on the prototype game.

Destructive tests

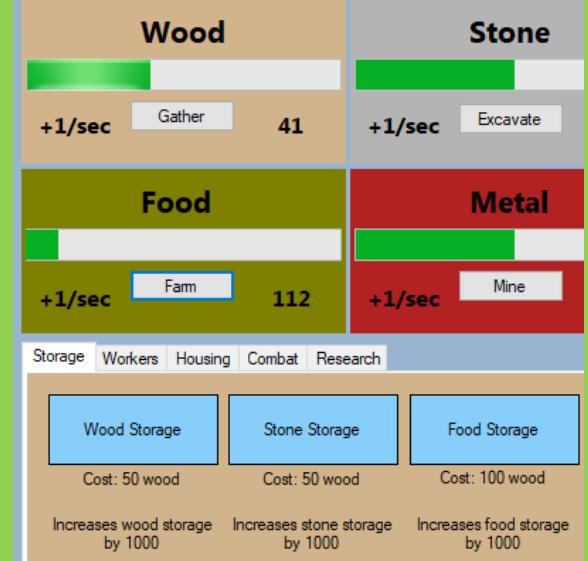
Note: Since making this table at the end of design, some of the rate numbers and other constants have changed e.g. in this first test, the rate is said to increase by 1/sec but now because of my balancing changes, the rate increases by 3/sec on button press. Therefore, I will mark any tests with only incorrect constants (that I have since changed during development) as a pass.

Some of the tests I made before development are not valid anymore due to the development process, e.g. saving/loading, so I have replaced those tests with the ones I made in the saving/loading section of development.

Input	Expected Output	Actual Output
Resource Collection Tests [Section A]		
The wood button is clicked	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on)	
The wood button is clicked, then clicked again	The rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When it is clicked again, the rate decreases by 1/sec, and the colour goes back to grey (button is toggled off)	
The wood button is clicked, then the food button is clicked	The wood rate increases by 1/sec, and the colour of the button turns brown (button is toggled on). When the food button is clicked, the wood rate decreased by 1/sec and the colour goes back to grey (button is toggled off). The food button then turns brown and food rate is increased by 1/sec (button is toggled on)	

		 <p>Wood</p> <p>+1/sec Gather 73</p> <p>Food</p> <p>+4/sec Farm 91</p>
The wood progress bar reaches maximum	The wood value no longer increases and the progress bar stays filled.	 <p>Wood</p> <p>+1/sec Gather 100</p> <p>Food</p> <p>+4/sec Farm 91</p>
A worker upgrade is bought	The rate of increase increments	 <p>Wood</p> <p>+6/sec Gather 68</p> <p>Food</p> <p>+4/sec Farm 624</p>
A storage upgrade is bought	Maximum amount of the resource is increased, causing the amount in the progress bar to be compressed into a small space	 <p>Housing</p> <p>10 housing capacity 9 housing remaining</p>
A housing type or upgrade is bought	The amount of free housing space is shown by the progress bar, and the amount left is increased in the amount label	 <p>Housing</p> <p>10 housing capacity 9 housing remaining</p>

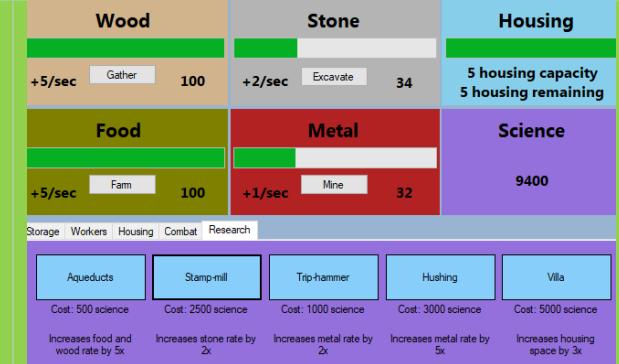
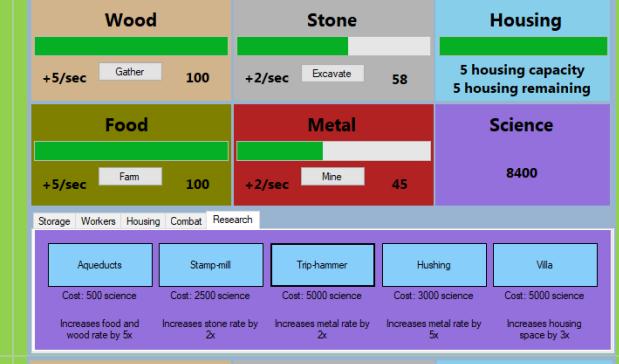
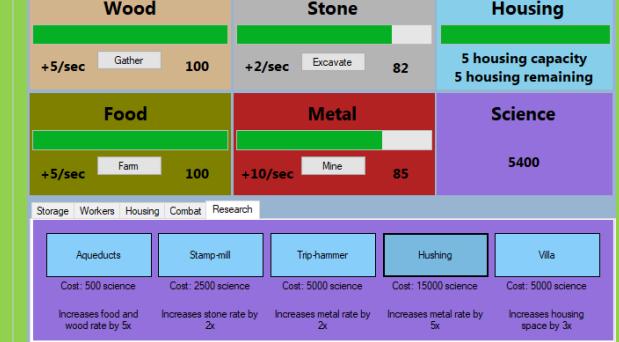
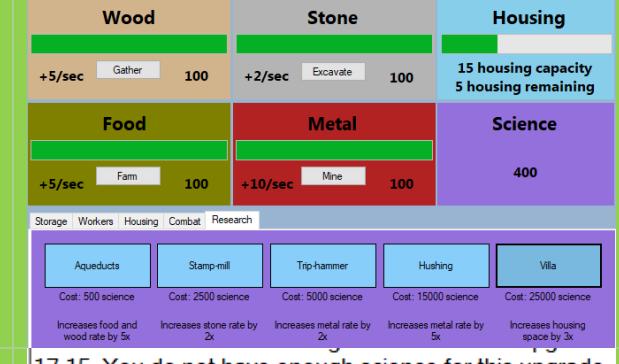
A battle is won/lost/tied	Nothing happens	<p>Enemy</p> <p>100/100, 25 attack, 20 block</p> <p>Enemy tiles: 7</p> <p>Year: 499BC. Current war: War of Lake Regillus.</p> <p>Allied tiles: 13</p> <p>100/100, 50 attack, 20 block, +25 bonus!</p> <p>Player</p>
A war is won/lost/tied	Rate increases, science increases, other possible parts increase or decrease depending on the outcome and loot gained	War has been won in this case: Science 5000 10:59. You have gained half of the enemy's troops, and gained 500 of each resource + 5000 science!
Upgrades Tests [Section A]		
There are not enough resources to buy wood storage	Print "You do not have enough resources for this" in upgrades logs, upgrade is not bought	15:20: You do not have enough wood for this upgrade
It costs 50 wood to buy wood storage, user has reached full capacity of the wood storage, buys storage	Wood amount goes down by 50, the upgrade goes through, the storage is increased by 1000, so progress bar is pushed further back, storage cost increased by 3x	<p>Wood</p> <p>+1/sec Gather 52</p> <p>Food</p> <p>+1/sec Farm 35</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Stone Storage</p> <p>Cost: 100 wood Cost: 50 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000</p>

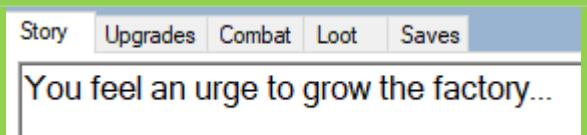
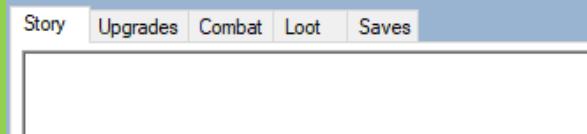
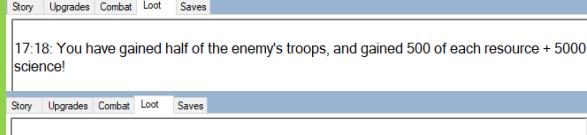
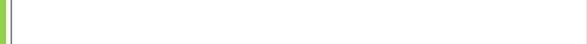
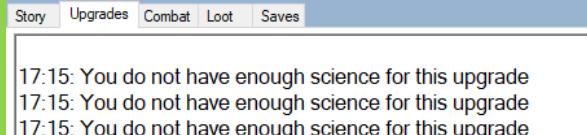
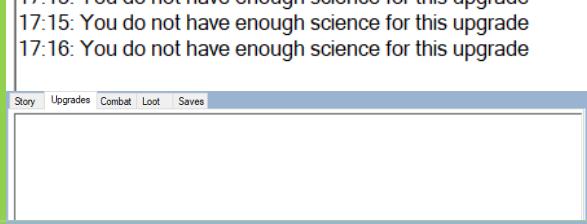
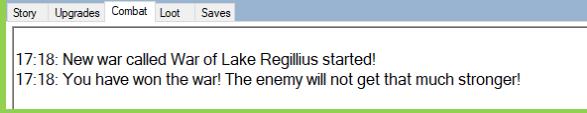
Food storage upgrade is bought	The maximum of the wood stays the same, maximum of the food increases by 1000	 <p>Wood +1/sec Gather 41 Stone +1/sec Excavate</p> <p>Food +1/sec Farm 112 Metal +1/sec Mine</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Stone Storage Food Storage Cost: 50 wood Cost: 50 wood Cost: 100 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000 Increases food storage by 1000</p>
The troop upgrade is bought (enough housing and metal)	The amount of metal goes down by cost, the housing amount goes down by 1, number of troops increased by 1, cost increased by 4 times	<p>Housing 5 housing capacity 4 housing remaining</p> <p>Science 32400</p> <p>Troop Cost: 800 metal, 1 housing Increases troop amount by 1</p> <p>Note: I edited science to have lots of it so testing would not take ages waiting to get enough resources – there are separate systems so will not affect the results.</p>
There is not enough food (but there is enough housing space) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	15:53: You do not have enough food for this upgrade

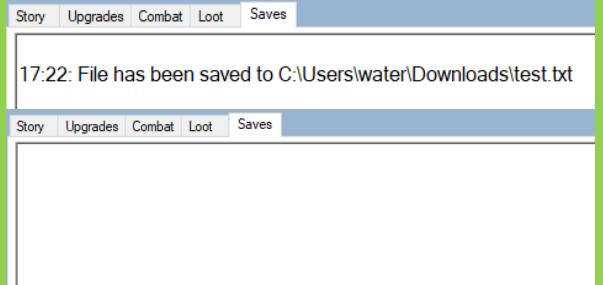
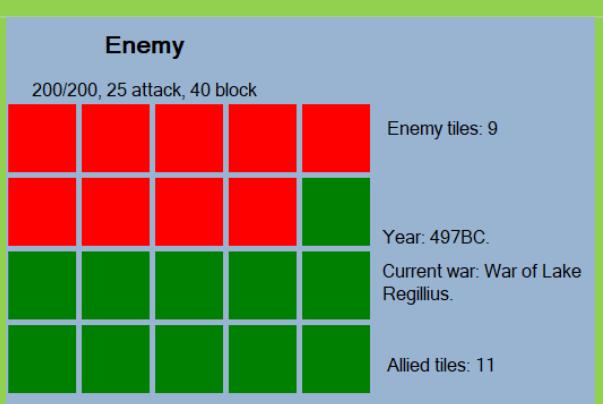
There is not enough housing space (but there is enough food) to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	<h2>Housing</h2> <p>5 housing capacity 0 housing remaining</p> <p>15:55: You do not have enough food for this upgrade</p>												
There is not enough food or housing space to buy the butcher upgrade	Print "You do not have enough resources or housing space for this" in upgrades logs, upgrade is not bought	15:55: You do not have enough food for this upgrade												
Food has reached full capacity, quarrier upgrade is bought	Food amount goes down by cost of upgrade, housing space goes down by 1, stone rate increased by 1/sec	<p>Wood: +1/sec Gather 90</p> <p>Stone: +3/sec Excavate 98</p> <p>Food: +1/sec Farm 4</p> <p>Metal: +1/sec Mine 90</p> <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr> <td>Gatherer</td> <td>Quarier</td> <td>Butcher</td> <td>Mine</td> </tr> <tr> <td>Cost: 100 food, 1 housing</td> <td>Cost: 400 food, 1 housing</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 200 food, 1 housing</td> </tr> <tr> <td>Increases wood rate by 1</td> <td>Increases stone rate by 1</td> <td>Increases food rate by 1</td> <td>Increases metal rate by 1</td> </tr> </table>	Gatherer	Quarier	Butcher	Mine	Cost: 100 food, 1 housing	Cost: 400 food, 1 housing	Cost: 100 food, 1 housing	Cost: 200 food, 1 housing	Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1
Gatherer	Quarier	Butcher	Mine											
Cost: 100 food, 1 housing	Cost: 400 food, 1 housing	Cost: 100 food, 1 housing	Cost: 200 food, 1 housing											
Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1	Increases metal rate by 1											
Metal has reached full capacity, troop upgrade is bought	Metal amount goes down by cost of upgrade, housing space goes down by 1, troop amount increased by 1	<p>Troop</p> <p>Cost: 800 metal, 1 housing</p> <p>Increases troop amount by 1</p>												
Food storage upgrade is bought, then butcher upgrade	Wood amount goes down by cost of food storage, food capacity goes up by 1000, butcher upgrade bought so food amount goes down by cost, housing space reduced by 1 and food rate increased by 1/sec	<p>Food: +6/sec Farm 21</p> <p>Metal: +2/sec Mine</p> <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr> <td>Gatherer</td> <td>Quarier</td> <td>Butcher</td> </tr> <tr> <td>Cost: 100 food, 1 housing</td> <td>Cost: 400 food, 1 housing</td> <td>Cost: 400 food, 1 housing</td> </tr> <tr> <td>Increases wood rate by 1</td> <td>Increases stone rate by 1</td> <td>Increases food rate by 1</td> </tr> </table>	Gatherer	Quarier	Butcher	Cost: 100 food, 1 housing	Cost: 400 food, 1 housing	Cost: 400 food, 1 housing	Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1			
Gatherer	Quarier	Butcher												
Cost: 100 food, 1 housing	Cost: 400 food, 1 housing	Cost: 400 food, 1 housing												
Increases wood rate by 1	Increases stone rate by 1	Increases food rate by 1												

Food has reached just about enough to buy gatherer upgrade, housing space left is 1	Food and housing goes down to 0, food rate increased by 1/sec	<p>Note: I just realised that the descriptions of each upgrade is wrong – e.g. gatherer upgrade increases wood rate by +5/sec. This does not invalidate the test, as this is what is meant to happen.</p>
The shack is bought for 50 stone	50 stone deducted, housing space increased by 5, cost increased by 2x	
The bootcamp is bought for 100 stone	100 stone deducted, housing space increased by 10, cost increased by 2x	
The barracks is bought for 500 stone	500 stone deducted, housing space increased by 20, cost increased by 2x	
The user tries to buy shack upgrade, but there is not enough stone	Upgrade does not go through, print not enough resources in upgrades logs	17:10: You do not have enough stone for this upgrade

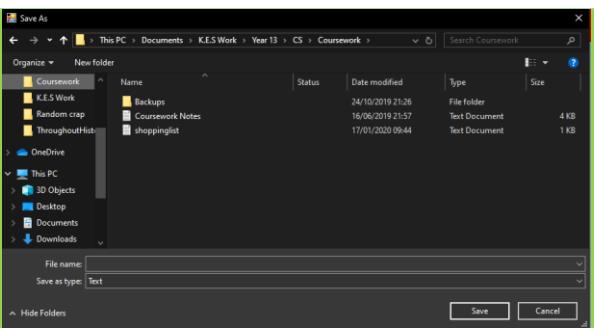
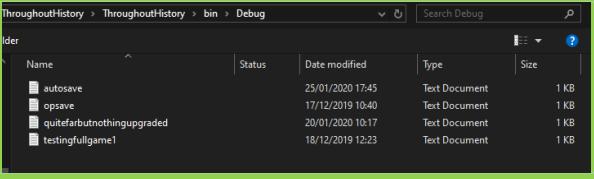
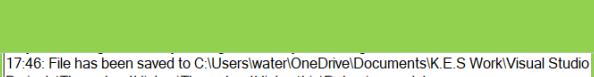
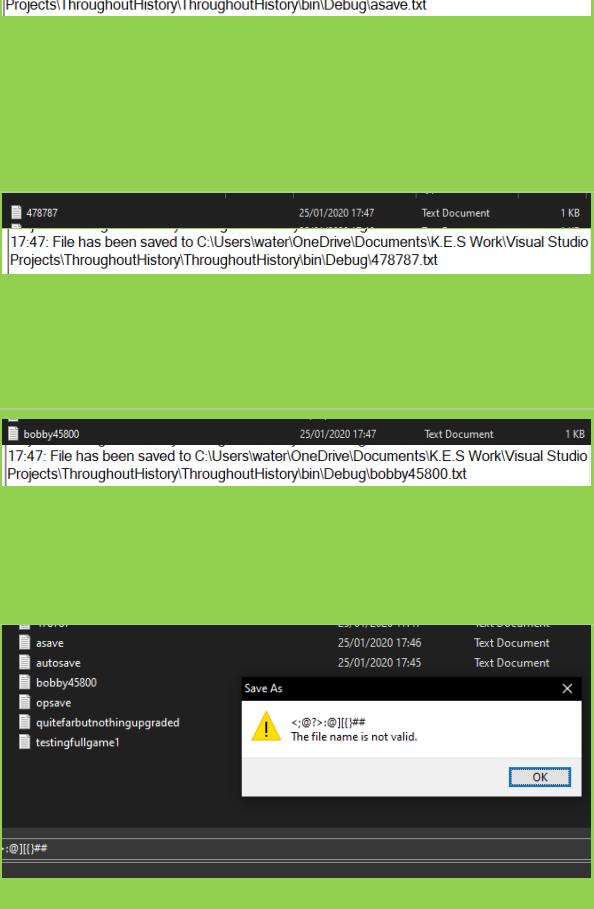
Troop health upgrade bought	Metal reduced by cost, health goes up by 100																												
Troop block upgrade bought	Metal reduced by cost, block goes up by 250																												
Troop attack upgrade bought	Metal reduced by cost, attack goes up by 300																												
Troop health upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console	17:12: You do not have enough metal for this upgrade																											
Troop block upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console	17:12: You do not have enough metal for this upgrade																											
Troop attack upgrade bought, not enough metal	Print "You do not have enough metal for this" in upgrades console	17:12: You do not have enough metal for this upgrade																											
Aqueducts upgrade bought	Science reduced by cost, food and wood rate go up by 5x, cost multiplied by 5x	<p>Note: Science started at 10,000</p> <table border="1"> <thead> <tr> <th>Wood</th> <th>Stone</th> <th>Housing</th> </tr> </thead> <tbody> <tr> <td>+5/sec Gather 15</td> <td>+1/sec Excavate 8</td> <td>5 housing capacity 5 housing remaining</td> </tr> <tr> <th>Food</th> <th>Metal</th> <th>Science</th> </tr> <tr> <td>+5/sec Farm 20</td> <td>+1/sec Mine 8</td> <td>9900</td> </tr> </tbody> </table> <p>Storage Workers Housing Combat Research</p> <table border="1"> <thead> <tr> <th>Aqueducts</th> <th>Stamp-mill</th> <th>Trip-hammer</th> <th>Hushing</th> <th>Villa</th> </tr> </thead> <tbody> <tr> <td>Cost: 500 science</td> <td>Cost: 500 science</td> <td>Cost: 1000 science</td> <td>Cost: 3000 science</td> <td>Cost: 5000 science</td> </tr> <tr> <td>Increases food and wood rate by 5x</td> <td>Increases stone rate by 2x</td> <td>Increases metal rate by 2x</td> <td>Increases metal rate by 5x</td> <td>Increases housing space by 3x</td> </tr> </tbody> </table>	Wood	Stone	Housing	+5/sec Gather 15	+1/sec Excavate 8	5 housing capacity 5 housing remaining	Food	Metal	Science	+5/sec Farm 20	+1/sec Mine 8	9900	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 500 science	Cost: 500 science	Cost: 1000 science	Cost: 3000 science	Cost: 5000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x
Wood	Stone	Housing																											
+5/sec Gather 15	+1/sec Excavate 8	5 housing capacity 5 housing remaining																											
Food	Metal	Science																											
+5/sec Farm 20	+1/sec Mine 8	9900																											
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa																									
Cost: 500 science	Cost: 500 science	Cost: 1000 science	Cost: 3000 science	Cost: 5000 science																									
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x																									

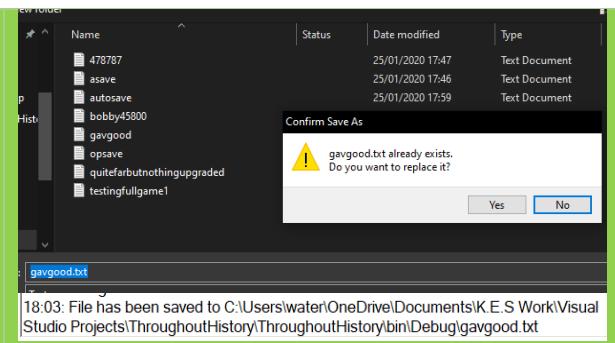
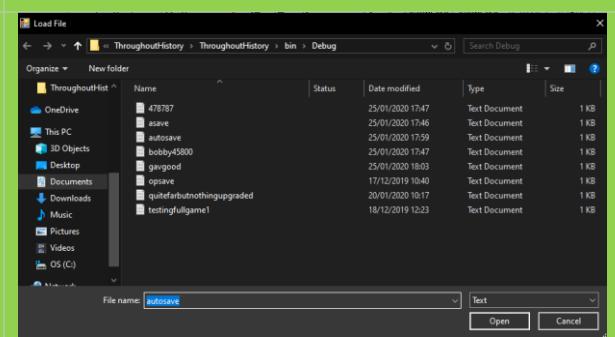
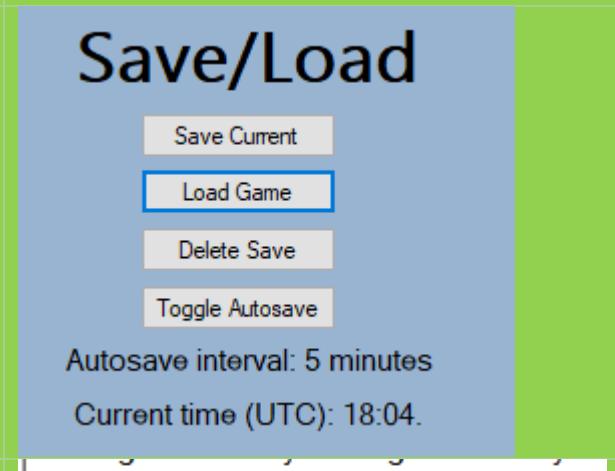
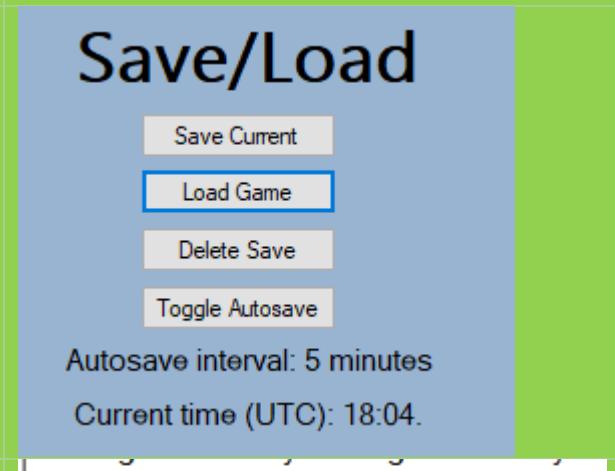
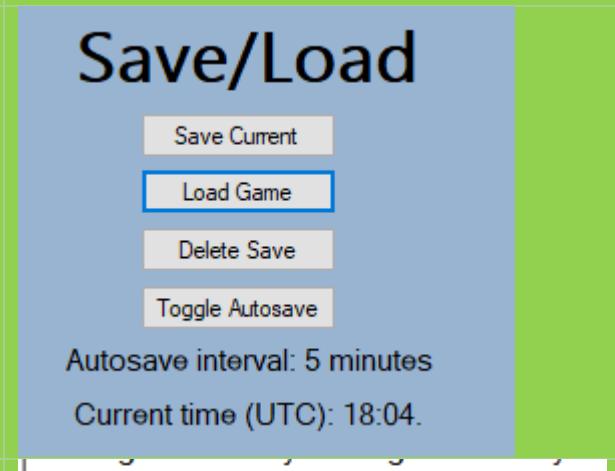
Stamp-mill upgrade bought	Science reduced by cost, stone rate goes up by 2x, cost multiplied by 5x	 <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr><td>Aqueducts</td><td>Stamp-mill</td><td>Trip-hammer</td><td>Hushing</td><td>Villa</td></tr> <tr><td>Cost: 500 science</td><td>Cost: 2500 science</td><td>Cost: 1000 science</td><td>Cost: 3000 science</td><td>Cost: 5000 science</td></tr> <tr><td>Increases food and wood rate by 5x</td><td>Increases stone rate by 2x</td><td>Increases metal rate by 2x</td><td>Increases metal rate by 5x</td><td>Increases housing space by 3x</td></tr> </table>	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 500 science	Cost: 2500 science	Cost: 1000 science	Cost: 3000 science	Cost: 5000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa													
Cost: 500 science	Cost: 2500 science	Cost: 1000 science	Cost: 3000 science	Cost: 5000 science													
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x													
Trip-hammer upgrade bought	Science reduced by cost, metal rate goes up by 2x, cost multiplied by 5x	 <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr><td>Aqueducts</td><td>Stamp-mill</td><td>Trip-hammer</td><td>Hushing</td><td>Villa</td></tr> <tr><td>Cost: 500 science</td><td>Cost: 2500 science</td><td>Cost: 5000 science</td><td>Cost: 3000 science</td><td>Cost: 5000 science</td></tr> <tr><td>Increases food and wood rate by 5x</td><td>Increases stone rate by 2x</td><td>Increases metal rate by 2x</td><td>Increases metal rate by 5x</td><td>Increases housing space by 3x</td></tr> </table>	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 500 science	Cost: 2500 science	Cost: 5000 science	Cost: 3000 science	Cost: 5000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa													
Cost: 500 science	Cost: 2500 science	Cost: 5000 science	Cost: 3000 science	Cost: 5000 science													
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x													
Hushing upgrade bought	Science reduced by cost, metal rate goes up by 5x, cost multiplied by 5x	 <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr><td>Aqueducts</td><td>Stamp-mill</td><td>Trip-hammer</td><td>Hushing</td><td>Villa</td></tr> <tr><td>Cost: 500 science</td><td>Cost: 2500 science</td><td>Cost: 5000 science</td><td>Cost: 15000 science</td><td>Cost: 5000 science</td></tr> <tr><td>Increases food and wood rate by 5x</td><td>Increases stone rate by 2x</td><td>Increases metal rate by 2x</td><td>Increases metal rate by 5x</td><td>Increases housing space by 3x</td></tr> </table>	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 500 science	Cost: 2500 science	Cost: 5000 science	Cost: 15000 science	Cost: 5000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa													
Cost: 500 science	Cost: 2500 science	Cost: 5000 science	Cost: 15000 science	Cost: 5000 science													
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x													
Villa upgrade bought	Science reduced by cost, housing space goes up by 3x, cost multiplied by 5x	 <p>Storage Workers Housing Combat Research</p> <table border="1"> <tr><td>Aqueducts</td><td>Stamp-mill</td><td>Trip-hammer</td><td>Hushing</td><td>Villa</td></tr> <tr><td>Cost: 500 science</td><td>Cost: 2500 science</td><td>Cost: 5000 science</td><td>Cost: 15000 science</td><td>Cost: 25000 science</td></tr> <tr><td>Increases food and wood rate by 5x</td><td>Increases stone rate by 2x</td><td>Increases metal rate by 2x</td><td>Increases metal rate by 5x</td><td>Increases housing space by 3x</td></tr> </table>	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 500 science	Cost: 2500 science	Cost: 5000 science	Cost: 15000 science	Cost: 25000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa													
Cost: 500 science	Cost: 2500 science	Cost: 5000 science	Cost: 15000 science	Cost: 25000 science													
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x													
Aqueducts upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	17:15: You do not have enough science for this upgrade															
Stamp-mill upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	17:15: You do not have enough science for this upgrade															

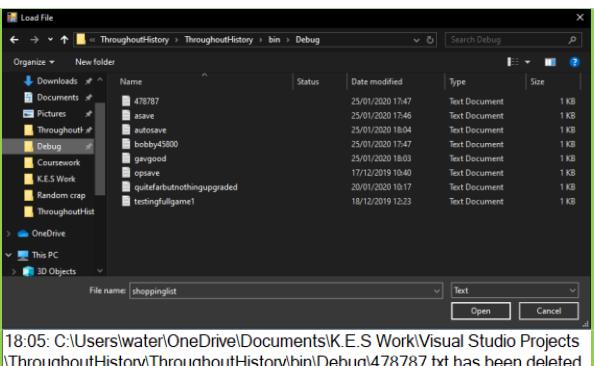
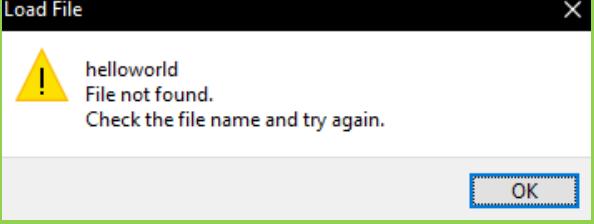
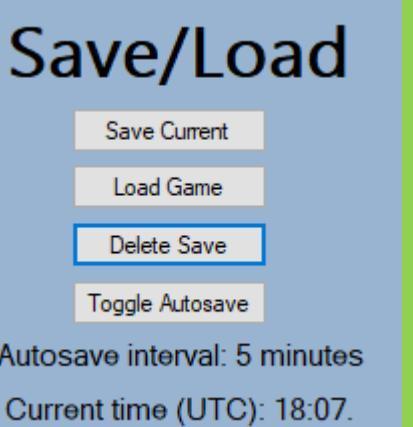
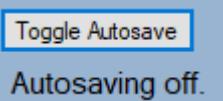
Trip-hammer upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	17:15: You do not have enough science for this upgrade
Hushing upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	17:15: You do not have enough science for this upgrade
Villa upgrade bought, not enough science	Print "You do not have enough science for this" in upgrades logs	17:16: You do not have enough science for this upgrade
Logs Tests [Section D]		
Some text is in the story logs, clear button pressed	Only text in the story logs is cleared	 
Some text is in the loot logs, clear button pressed	Only text in the loot logs is cleared	 
Some text is in the upgrades collection logs, clear button pressed	Only text in the upgrades collection logs is cleared	 
Some text is in the combat logs, clear button pressed	Only text in the combat logs is cleared	 

		
Some text is in the saves logs, clear button pressed	Only text in the saves logs is cleared	
Combat Tests [Section C]		
300 seconds (ticks) have passed since the game first loaded up (not from save file)	A new war starts, the health, attack and block labels are set to the correct values, the year and war name is shown, the grid is in the default state, new war message printed into combat logs	 <p>Enemy</p> <p>200/200, 25 attack, 40 block</p> <p>Enemy tiles: 9</p> <p>Year: 497BC.</p> <p>Current war: War of Lake Regillius.</p> <p>Allied tiles: 11</p> <p>1000/1000, 500 attack, 200 block, +10 bonus!</p> <p>Player</p> <p>Story Upgrades Combat Loot Saves</p>
New war starts. Player has the upper hand so beats the enemy.	The player receives winning loot. Board reset, labels reset, winning message and loot gained printed into loot logs.	 <p>17:24: New war called War of Lake Regillius started!</p> <p>17:25: You have won the war! The enemy will not get that much stronger!</p> <p>17:25: You have gained half of the enemy's troops, and gained 500 of each resource + 5000 science!</p> <p>Enemy</p> <p>No war running</p> <p>Enemy tiles: 10</p> <p>Year: 496BC.</p> <p>Current war: None - Check loot and combat logs.</p> <p>Allied tiles: 10</p> <p>No war running</p> <p>Player</p> <p>Story Upgrades Combat Loot Saves</p>

A war has ended, another few ticks pass until a new war begins again.	Everything resets properly, new war message printed into combat logs.	<p>Enemy</p> <p>205/700, 350 attack, 280 block</p> <p>1440/1500, 750 attack, 300 block, +25 bonus!</p> <p>17:27: New war called War of Corbio started!</p>
Enemy wins tiles	Grid updated to show one more red tile	<p>Enemy</p> <p>200/200, 50 attack, 40 block, +5 bonus!</p> <p>Enemy tiles: 11</p> <p>Year: 499BC.</p> <p>Current war: War of Lake Regillius.</p> <p>Allied tiles: 9</p> <p>100/100, 50 attack, 20 block</p> <p>Player</p>
Player wins tiles	Grid updated to show one more green tile	<p>Enemy</p> <p>100/100, 25 attack, 20 block</p> <p>Enemy tiles: 9</p> <p>Year: 499BC.</p> <p>Current war: War of Lake Regillius.</p> <p>Allied tiles: 11</p> <p>1000/1000, 500 attack, 200 block, +10 bonus!</p> <p>Player</p>
Saving/Loading Tests [Section B]		

Save button pressed	Save dialog box appears, only allows .txt files	
Autosave happens	Save dialog box does not appear, autosave.txt saved or overwritten to directory of executable	
Filename "asave" is entered and submit button pressed after save game button pressed	A new file is created in the executable file path called "asave.txt" with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename "asave.txt" is entered	A new file called "asave.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename "478787" is entered	A new file called "478787.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename "bobby45800.txt" is entered	A new file called "bobby45800.txt" is created with all the appropriate values inside. Output file saved successfully into logs and output box	
Filename containing any of the following characters in its name is entered: "\/*?<> #{}"	The output box says that the file cannot be created because one of the invalid filename characters has been entered. Output file saved successfully into logs and output box	

A file called "gavgood" is saved, and then the same name is entered for the next one	The file "gavgood" is saved, then the next time it is overwritten into the same file	
Load game button pressed	Open save dialog appears	
Load game button pressed, file "autosave.txt" is selected and submit button pressed	Open save dialog appears, on submit button press, load all data from "autosave.txt" and output to logs	
Load game button pressed, file "autosave.txt" is selected and cancel button pressed	Open save dialog appears, on cancel button press, dialog closes and nothing happens	
Load game button pressed, file "shoppinglist" is selected and submit button pressed	Open save dialog appears, on submit button press, dialog closes and nothing happens (because file is invalid to load from)	

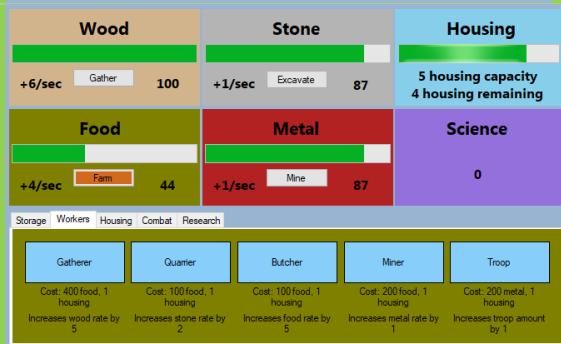
Delete save button is pressed	Open file dialog appears with text file filter on	
Delete save button is pressed, file in the selected directory is entered or selected, OK button pressed	Open file dialog appears with text file filter on, when OK button pressed, dialog closes and save deleted	18:05: C:\Users\water\OneDrive\Documents\K.E.S Work\Visual Studio Projects\ThroughoutHistory\ThroughoutHistory\bin\Debug\478787.txt has been deleted permanently.
Delete save button is pressed, a file name is entered, even though that file does not exist in the current directory	Open file dialog appears with text file filter on, when non-existent file name entered, message box appears saying “[filename] File not found. Check the file name and try again”	
Delete save button is pressed, cancel button pressed	Open file dialog appears with text file filter on, when cancel button pressed the dialog closes and nothing happens	
Game is first loaded up, 5 minutes (300 ticks) pass	First autosave happens, to file “autosave.txt”. If it already exists, overwrite file. Show autosave has been made in logs	300 12:26: File has been saved to ./autosave.txt 12:26: The game has been autosaved! 301 600 12:31 File has been saved to ./autosave.txt 12:31 The game has been autosaved! 601
The autosave button is pressed,	The autosave timer is toggled off, output that to logs and label	

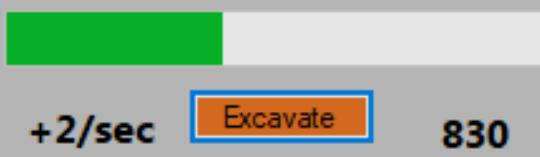
autosave timer currently on		
The autosave button is pressed, autosave timer currently off	The autosave timer is toggled on, output that to logs and label	Toggle Autosave Autosave interval: 5 minutes

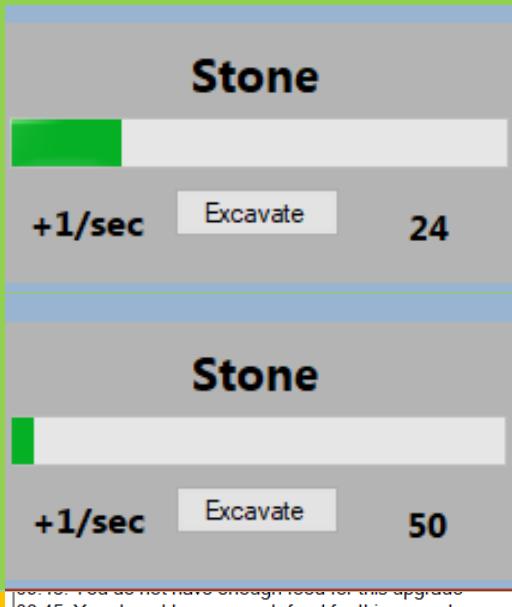
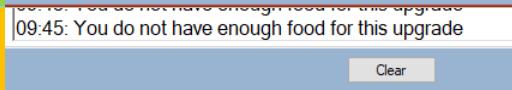
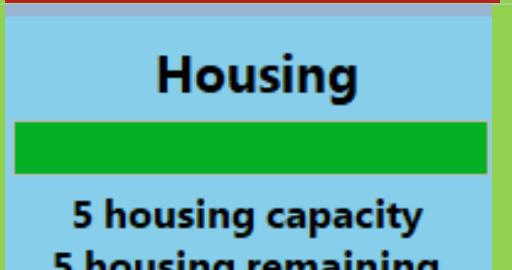
Solution Success

Success Criteria

- Green – Feature finished
- Orange – Feature works but has a minor error, or a constant has changed during development (e.g. on button press rate is not +1/sec for every resource anymore), or has a part that works and a part that doesn't/hasn't been implemented
- Red – Feature failed/not in game

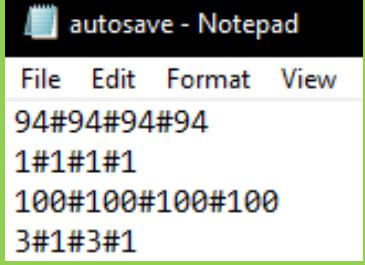
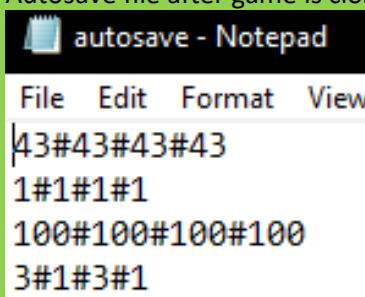
Criteria	Evidence of the criteria being met	How well criteria was met/comments of development	Section Code
Section A: Resources/upgrades system			
Resources go up over time	 	Criteria was 100% met. There were no design changes during the development (as this is a foundation of the game)	A1
Resource rate increased through upgrade		Criteria was 100% met. There were no design changes during the development (as this is a foundation of the game)	A2

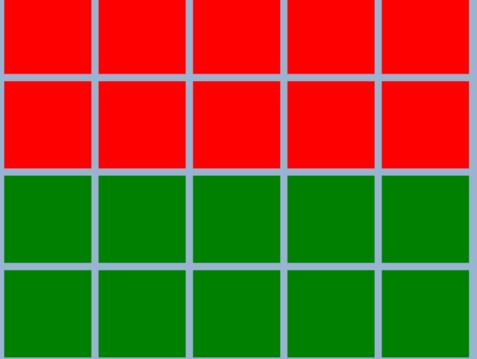
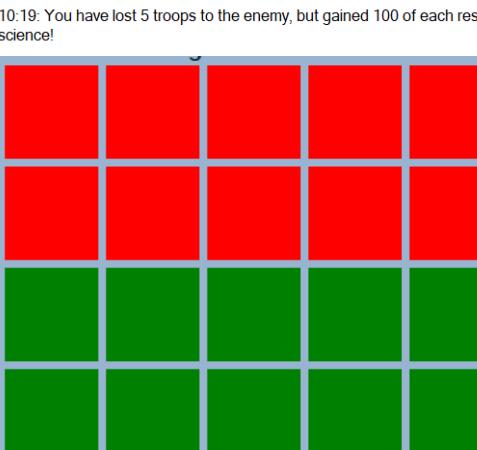
<p>Resources reach “milestone” where the number shortens</p>	<p>Wood</p>  <p>+750/se c</p> <p>Food</p>  <p>+125/se c</p>	<p>Criteria not met. Criteria not once considered throughout development, although a way to show larger numbers was considered without the use of this system.</p>	<p>A3 <i>Opti onal</i></p>
<p>Button to manually increase rate of resource collection increased, button turns brown, all other buttons turn grey and toggle off, that resource rate increases by 10%</p>	<p>Stone</p>  <p>+2/sec</p> <p>Metal</p>  <p>+1/sec</p> <p>Stone</p>  <p>+1/sec</p> <p>Metal</p>  <p>+2/sec</p>	<p>Criteria met, although the increase of resource rate by 10% not. The rate increases by 3 for wood and food, 1 for stone and metal. This changed during the balancing changes at the end of development.</p>	<p>A4 <i>Opti onal</i></p>

[Resource] storage upgrade is bought		Criteria 100% met.	A5
Worker for [resource] upgrade is bought, not enough housing space		Criteria met as housing can be bought, but the output says there is not enough food for the upgrade, when in fact there is no enough housing space (i.e. wrong logs message).	A6
Worker for [resource] upgrade is bought, enough housing space		Criteria 100% met, works as intended from the start.	A7
Housing upgrade is bought		Criteria 100% met.	A8

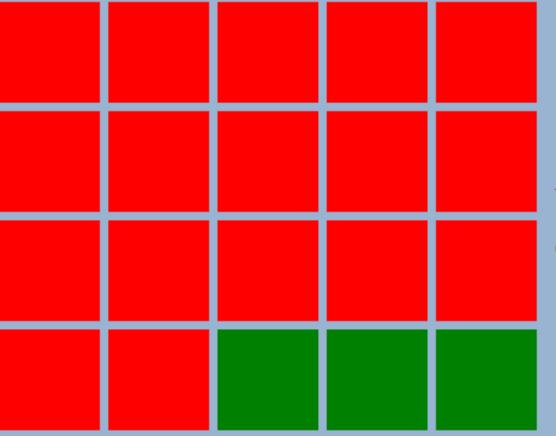
		Housing																						
		10 housing capacity 10 housing remaining																						
Science upgrade is bought, but not enough science points available		09:57: You do not have enough science for this upgrade	Criteria 100% met.	A9																				
Science upgrade for combat is bought, new combat upgrade button appears in combat tab		<table border="1"> <thead> <tr> <th>Storage</th> <th>Workers</th> <th>Housing</th> <th>Combat</th> <th>Research</th> </tr> </thead> <tbody> <tr> <td>Aqueducts</td> <td>Stamp-mill</td> <td>Trip-hammer</td> <td>Hushing</td> <td>Villa</td> </tr> <tr> <td>Cost: 12500 science</td> <td>Cost: 500 science</td> <td>Cost: 1000 science</td> <td>Cost: 3000 science</td> <td>Cost: 5000 science</td> </tr> <tr> <td>Increases food and wood rate by 5x</td> <td>Increases stone rate by 2x</td> <td>Increases metal rate by 2x</td> <td>Increases metal rate by 5x</td> <td>Increases housing space by 3x</td> </tr> </tbody> </table>	Storage	Workers	Housing	Combat	Research	Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa	Cost: 12500 science	Cost: 500 science	Cost: 1000 science	Cost: 3000 science	Cost: 5000 science	Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x	Criteria not met. This criteria was not required and could fairly easily be added through future developments of the prototype. The game functions without requiring it (basically just extra content).	A10
Storage	Workers	Housing	Combat	Research																				
Aqueducts	Stamp-mill	Trip-hammer	Hushing	Villa																				
Cost: 12500 science	Cost: 500 science	Cost: 1000 science	Cost: 3000 science	Cost: 5000 science																				
Increases food and wood rate by 5x	Increases stone rate by 2x	Increases metal rate by 2x	Increases metal rate by 5x	Increases housing space by 3x																				
Different upgrade types tabs selected		<table border="1"> <thead> <tr> <th>Storage</th> <th>Workers</th> <th>Housing</th> <th>Combat</th> <th>Research</th> </tr> </thead> <tbody> <tr> <td>Gatherer</td> <td>Quarier</td> <td>Butcher</td> <td>Miner</td> <td>Troop</td> </tr> <tr> <td>Cost: 100 food, 1 housing</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 100 food, 1 housing</td> <td>Cost: 200 food, 1 housing</td> <td>Cost: 200 metal, 1 housing</td> </tr> <tr> <td>Increases wood rate by 5</td> <td>Increases stone rate by 2</td> <td>Increases food rate by 5</td> <td>Increases metal rate by 1</td> <td>Increases troop amount by 1</td> </tr> </tbody> </table>	Storage	Workers	Housing	Combat	Research	Gatherer	Quarier	Butcher	Miner	Troop	Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 200 food, 1 housing	Cost: 200 metal, 1 housing	Increases wood rate by 5	Increases stone rate by 2	Increases food rate by 5	Increases metal rate by 1	Increases troop amount by 1	Criteria 100% met – different upgrades types are in different tabs.	A11 <i>Optional</i>
Storage	Workers	Housing	Combat	Research																				
Gatherer	Quarier	Butcher	Miner	Troop																				
Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 100 food, 1 housing	Cost: 200 food, 1 housing	Cost: 200 metal, 1 housing																				
Increases wood rate by 5	Increases stone rate by 2	Increases food rate by 5	Increases metal rate by 1	Increases troop amount by 1																				
Combat upgrade to increase block bought		Where starting block is 20: 270 block after upgrade	Criteria 100% met. No changes through design or development.	A12																				

Enough resources are available to buy upgrade	<p>Wood</p> <p>+25/sec Gather 1100</p> <p>Food</p> <p>+25/sec Farm 2100</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Stone Storage</p> <p>Cost: 100 wood Cost: 50 wood</p> <p>Increases wood storage by 1000 Increases stone storage by 1000</p>	Criteria met, but there isn't a "clickable" or "unclickable" state, but you still can and can't buy an upgrade based off how much of a resource player has/upgrade costs.	A13 <i>Opti onal</i>
Section B: Save/load system			
Game is auto-saved every 2 minutes	<p>Story Upgrades Combat Loot Saves</p> <p>10:30: File has been saved to ./autosave.txt 10:30: The game has been autosaved! 10:35: File has been saved to ./autosave.txt 10:35: The game has been autosaved! 10:40: File has been saved to ./autosave.txt 10:40: The game has been autosaved!</p>	Criteria met, but autosaves changed to every 5 minutes during design based off stakeholder feedback.	B1
Player manually presses the save game button	<p>NewSaveGame.txt</p>	Criteria 100% met.	B2

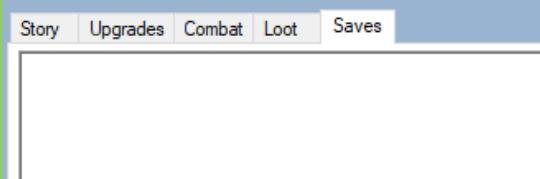
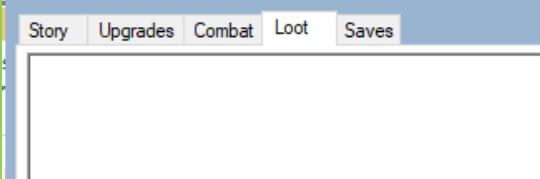
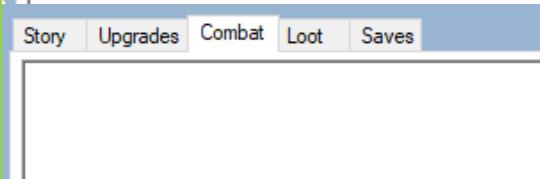
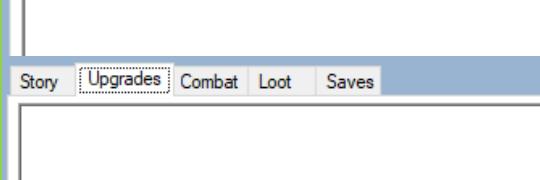
	<p>15#15#15#15 1#1#1#1 100#100#100#100 3#1#3#1 5000 5 5 50#50#50#100 100#100#100#200 100#500#1000#3000#5000 50#100#500 200#500#300#200 2#4#5#2#2 100#100 20#20 50#20 1#1 15 0 0 -500 500 BC</p>		
Game closed	<p>Autosave file before game is closed:</p>  <pre>94#94#94#94 1#1#1#1 100#100#100#100 3#1#3#1</pre> <p>Autosave file after game is closed:</p>  <pre>43#43#43#43 1#1#1#1 100#100#100#100 3#1#3#1</pre>	<p>Criteria 100% met – game is autosaved on close.</p>	B3

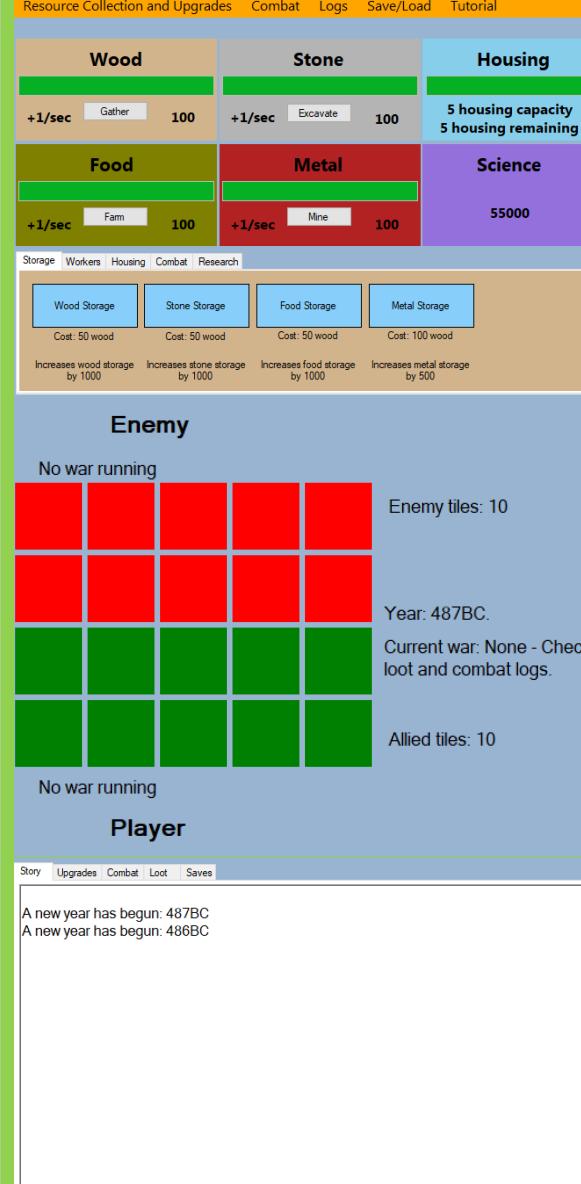
Game loaded up, player selects save file to load, message box tells player how many resources were made when they were offline	<h1>Save/Load</h1> <p>Save Current</p> <p>Load Game</p> <p>Delete Save</p> <p>Toggle Autosave</p> <p>Autosaving: Off</p> <p>Current time (UTC): 14:57.</p> <p>Can't evidence offline progress system as it does not exist.</p>	Criteria partially met – game can be loaded from a file, however there is no offline progress system.	B4
Section C: Combat/war system			
War is won by player	 <p>10.38: You have gained half of the enemy's troops, and gained 500 of each resource + 500 science!</p>	Criteria 100% met – grid resets when a war is complete.	C1
War is lost by player	 <p>10.19: You have lost 5 troops to the enemy, but gained 100 of each resource + 1000 science!</p>	Grid correctly resets on war loss	C2

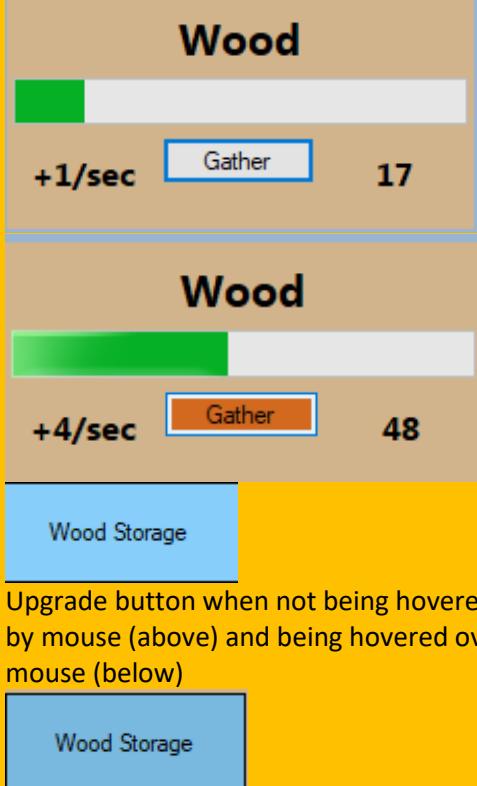
Battle is won by player	<p>100/100, 25 attack, 20 block, +10 bonus!</p> <p>85/100, 50 attack, 20 block</p> <p>Enemy</p> <p>100/100, 25 attack, 20 block, +10 bonus!</p> <p>85/100, 50 attack, 20 block</p> <p>Player</p>	Tiles can be won by the player.	C3
Battle is lost by player	<p>100/100, 250 attack, 200 block</p> <p>100/100, 50 attack, 20 block, +10 bonus!</p>	Tiles can be lost by the player/won by the enemy.	C4

	<p>100/100, 0, 250 attack, 200 block, +5 bonus!</p> 		
Player's army with 170 health and 50 block gets hit by 100 attack, so health goes down to 120	Too difficult to evidence, as I would either have to manually edit the game and remove RNG temporarily to allow these exact stats, or forge the results.	Criteria 100% met in any value of the player/enemy stats.	C5
Number of troops increased, total army health increases by (current upgrade of health) * number of new troops	<p>100/100, 50 attack, 20 block</p> <p>Player</p> <p>1 troop (above), 2 troops (below)</p> <p>200/200, 100 attack, 40 block,</p> <p>Player</p>	Criteria 100% met.	C6
New war started when right year reached, enemies have higher health/attack/block than in previous war	<p>Enemy</p> <p>100/100, 25 attack, 20 block</p> <p>First war (above), second war (below)</p> <p>Enemy</p> <p>200/200, 100 attack, 40 block</p>		C7
Game loaded up, war in progress continues as before	Can't screenshot non-existent feature	When a game is saved, the current war stats are not saved in the file. Because the current tick is, and a war starts based on the value of the tick counter, the war would essentially	C8

		“end” when a game is loaded, without any outcomes, thus not meeting the criteria.	
Section D: Logs system			
A battle is lost or won, show the message in the combat logs	12:15: New war called War of Lake Regillius started! You have won the battle! You are one step closer to winning the war. You have lost the battle! You are one step closer to losing the war. You have won the battle! You are one step closer to winning the war.	Criteria met.	D1
A war is lost or won, show the statistics in the combat logs	12:17: You have won the war! The enemy will not get that much stronger!	Criteria 95% met. Other “5%” is the fact that there technically aren’t any statistics shows, but that doesn’t matter.	D2 <i>Optimal</i>
A war is lost or won, logs show resource penalty/bonus until next war		Criteria met.	D3
A tech upgrade is made, and story progresses, show message in logs		Criteria met.	D4
A new year begins, print new year in logs	A new year has begun: 497BC A new year has begun: 496BC	Criteria met.	D5
Player battles for first time, buys first upgrade or reaches first resource milestone, print various tips or explanations in tutorial logs		Tutorial logs removed during development. Instead a new tutorial tab discussed to be added during future development.	D6
Game is auto-saved or manually saved by user, show message in auto-saves logs	11:19: File has been saved to C:\Users\water1\OneDrive\Documents\K.E.S Work\Visual Studio Projects\ThroughoutHistory\ThroughoutHistory\bin\Debug\autosave.txt	Criteria met. Shows entire file directory has well.	D7
A log type button is	Cannot provide screenshot evidence of non-existent feature	This log system was scrapped during design	D8

clicked to toggle it off, button changes from green to red and those logs stop showing		so this criteria cannot be met any longer.	
The clear logs button is clicked, all logs toggled on cleared	    	Criteria met. “Toggled on” part not, because that was part of a system I had envisioned before design, which was then scrapped. The clear button now just clears that one selected section.	D9
Save logs to file button is clicked, all logs saved to a text file	Screenshot of file it saved to	Logs are not saved, as stakeholders thought it not necessary during design.	D10 <i>Optional</i>
Section E: Usability			
The game balanced so that it can't be progressed really quickly (amount upgrades cost, how much the	Too difficult to provide screenshot evidence here.	Criteria met, but evidence can't be shown, other than going to the balancing section in development.	E1 <i>Optional</i>

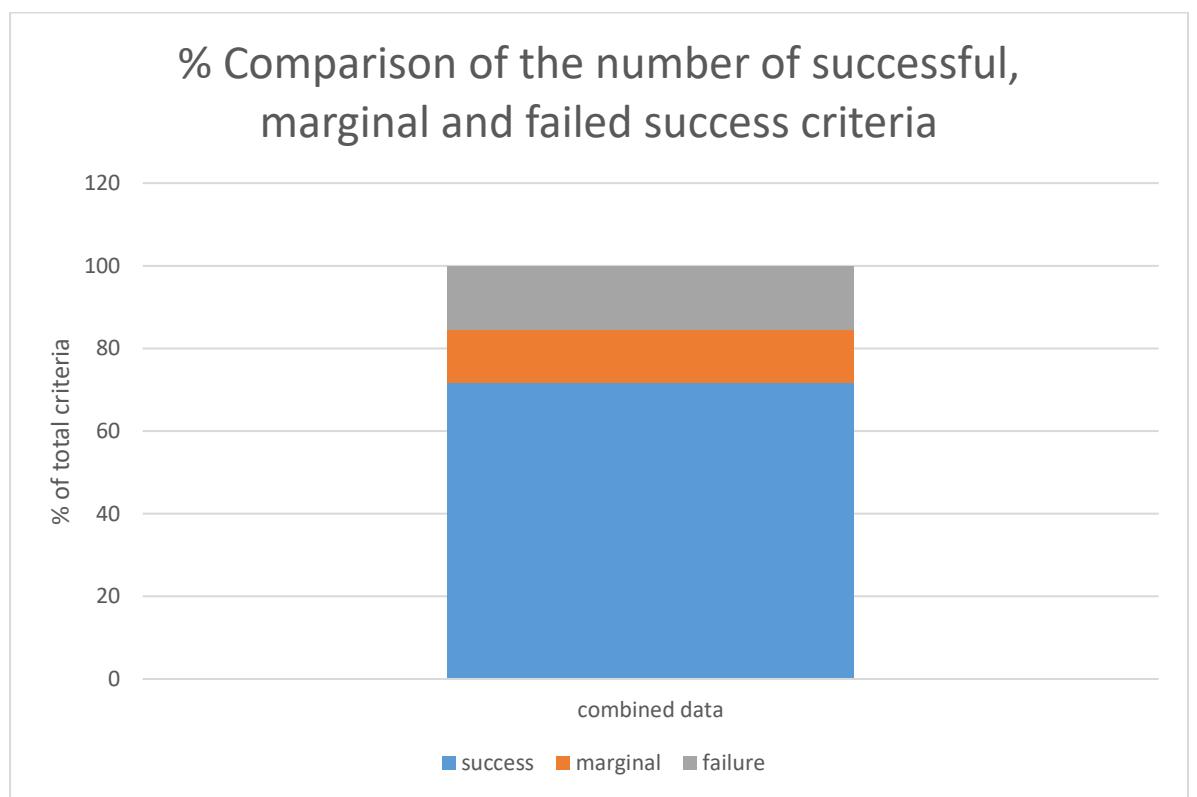
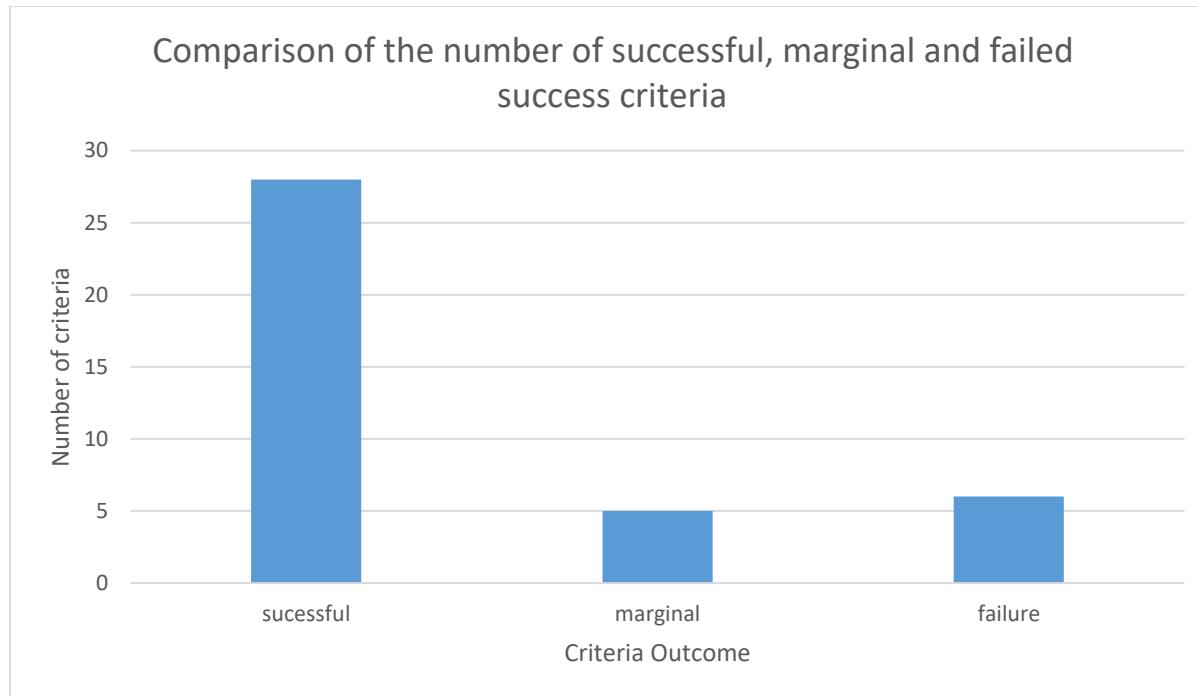
affect various parts of the game etc.)		
Intuitive and easy to use menu	 <p>Resource Collection and Upgrades</p> <p>Combat</p> <p>Logs</p> <p>Save/Load</p> <p>Tutorial</p> <p>Wood</p> <p>Stone</p> <p>Housing</p> <p>Food</p> <p>Metal</p> <p>Science</p> <p>Storage Workers Housing Combat Research</p> <p>Wood Storage Cost: 50 wood Increases wood storage by 1000</p> <p>Stone Storage Cost: 50 wood Increases stone storage by 1000</p> <p>Food Storage Cost: 50 wood Increases food storage by 1000</p> <p>Metal Storage Cost: 100 wood Increases metal storage by 500</p> <p>Enemy</p> <p>No war running</p> <p>Enemy tiles: 10</p> <p>Year: 487BC.</p> <p>Current war: None - Check loot and combat logs.</p> <p>No war running</p> <p>Player</p> <p>Story Upgrades Combat Loot Saves</p> <p>A new year has begun: 487BC A new year has begun: 486BC</p> <p>Save/Load</p> <p>Clear</p> <p>Save Current</p> <p>Load Game</p> <p>Delete Save</p> <p>Toggle Autosave</p> <p>Autosaving: Off</p> <p>Current time (UTC): 11:28.</p>	<p>Tutorial logs menu can be easily added to the top bar during future development.</p> <p>E2</p>

Game does not have low framerate	 CPU load on left, RAM load on right	Because the game is all static menus, and the only “visible” updates happen every 1 or more second, the framerate doesn't actually need to appear more than 1fps. Game also has minimal system impact.	E3
Big buttons, some colour coding to show toggles and being able to be clicked	 +1/sec Gather 17	None of the buttons disappear or reappear on button press anymore – this used to be in the saving/loading menu but was removed during development after a much simpler system was created. Buttons do however change colour on press or hover.	E4 <i>Optional</i>

6 of my 39 success criteria have not been met. Of these, 2 were optional, and 4 were not. 5 criteria have been partially met or have had minor changes, and of these, 2 were optional. This comes out to be:

No. of criteria	Successful criteria	% of total	Marginal criteria	% of total	Failure criteria	% of total	Optional marginal criteria	% of total	Failure optional criteria	%
39	28	~72	5	~13	6	~15	2	~5	2	~5

From looking at this table, I would say that the solution is about 85% successful.



Failed Criteria

- A3 was a failed optional criteria. This was failed because it was outside of the scope of the program, thus I did not feel it was required in the success of the solution.
- A10 was a failed compulsory criteria. It could fairly easily be added through future developments of the prototype. The game functions without requiring it (basically just extra content). I would not consider this a major failure.

- I failed C8 because when a game is saved, the current war stats are not saved in the file. Because the current tick is, and a war starts based on the value of the tick counter, the war would essentially “end” when a game is loaded, without any outcomes, thus not meeting the criteria. I call this a failure of a major success criteria because when a player saves their game mid-war, they would understandably expect the war to continue on exactly where it was left off. This is not the case currently, although the file handling system is a very solid foundation for feature development so with some work could be added later on.
- Again, D6 is a fairly sizable criteria that I did not meet. The idea was to have tooltip that could appear when hovering over one part of the game, or give game tips whilst the game is being played in logs, given a certain toggleable tutorial option. However, the tutorial logs were removed during development because it was deemed an unnecessary feature and would cause more time constraints. Instead a new tutorial tab was discussed to be added during future development in the usability tests, as it was the most highly requested feature. My stakeholders could not easily understand what was going on until either they played for a few minutes (missing the first war) or I explained it to them. I would therefore call this a major failure.
- The D8 log system was scrapped during design. It is not optional, however because of the nature of the criteria, I do not think that this is a major failure.
- I had asked the stakeholders about D10 during the design process and they thought it not necessary, so logs are not saved/loaded. This criteria is optional, so this is a minor failure.

Overall, there are two major failures in this solution. Definitely the biggest failure of meeting a success criteria is C8, where a war should carry on as it was when loaded. The player is right to expect this feature as it shouldn't be too difficult to implement through adding some code onto the end of an already existing file handling system, yet I have failed to meet it due to time constraints. As I have talked about in the beta testing section earlier, the D6 criteria – essentially tutorial/tooltip system – could fairly easily be added into the game in future developments through either adding a new tab dedicated to explaining the game on the menu strip bar (which is the open tab when the user first loads the game up), or adding a little tooltip box that appears next to the cursor when hovering over certain features (given an option to turn it on/off perhaps).

Marginal Criteria

- A4 is an optional criteria which I have met, although with a very slight change during development which is that the rate of manual buttons is not +10% but actually a certain amount given a certain resource
- A6 works fine but there is a very minor issue which is that it shows the wrong log message when you don't have enough housing (it says not enough food). This shows a slight lack of success in realising all the different edge cases the program has.
- A13 works but because there are no “clickable” or “unclickable” states (i.e. locking the buttons if user cannot afford them rather than using logs) this has been sort of failed. I think if this criteria were fully met, the solution would actually be a lot better than it is now, because rather than having logs telling the user they can't afford something in another menu (which is confusing at first), having a visual of the buttons being “locked” is a much better way of telling the user why they can't afford an upgrade. This could still be used together with logs to tell them any extra info e.g. they are out of housing space to buy a worker rather than food itself.

- B1 works but the only change is that autosaves are every 5 minutes rather than 2 which is what was changed during design.
- B4 works in that you can save/load the game, however the issue with this is that when writing the success criteria, I had assumed that I or the stakeholders wanted an offline progress system. Because of time constraints of the project, I chose during design (it is obvious it takes a long time to add a large feature like that) to not implement such a system. If you look back in analysis, the stakeholder requirements asked for there to be an offline progress system only for the resource collection until storages are full.
- E4 is so minor and the only “failed” aspect of it is just that nothing shows or hides which means that generally the program is less complex than it could have been, making this part more successful that if I had completely followed it.

Overall, the only marginal criteria that is maybe a little issue is A13 because it would make the user experience a little better, but other than that there is not any problems here.

Stakeholder Requirements

- Green – Everything mentioned is in the game
- Orange – Some stuff mentioned is in the game, some not
- Red – Everything mentioned is not in the game

Feature	Proposed Solution	Comments on outcome
Resource collection	<p>There will be 5 resources:</p> <ul style="list-style-type: none"> • Population • Food • Wood • Stone • Metal <p>These will be the 5 resources throughout the whole game, but will be able to be upgraded as the game progresses. Research will be a separate part which requires resources to make for “prototyping” which thus allows the player to buy upgrades, using science plus varying resources depending on the upgrade type.</p> <p>The population will also use food, so the rate of food increase will also be decreased by this (forcing the player to buy more food workers) The player can also manually increase resource rate (but only one resource at a time) by toggling a button for a resource to increase the rate by 10%.</p> <p>The resource visualisation will be using progress bars, with the rate of increase and the number of resources underneath it. It will use a resource collection system like that of Trunks' and Kittens Game's – capped and resources fill up over time. There will also be a milestone system that changes each main unit from a large string of zeros to a “K” or “M” (for thousand or million). E.g. 4,850 will become 4.85K.</p>	This is quite similar to what the game currently has. The only things that are not are 1. Food decreasing based on how much population you have – this would be an interesting mechanic if the stakeholders had chosen not to stick with it 2. Milestone system which was an optional criteria but not met because of time constraints (could have perhaps been added in future development to appease the stakeholders).
Upgrades	<p>There will be 5 types of upgrades:</p> <ul style="list-style-type: none"> • Storage – this will cost wood, and increases the amount of resource space • Workers – this will cost food, and increases the rate of resource gain • Housing – this will cost stone, and increases the number of workers and troops the player can have • Combat equipment – this will cost metal, which will be used to increase the health, attack and block of the player's troops 	

	<ul style="list-style-type: none"> Science/tech – this will cost science, and unlocks various new housing types, combat equipment and upgrades worker speeds and maximum storage capacity 	
Starting era	Roman empire, because there was already a decent amount of tech, and a lot of progress of tech. There were also quite a number of major wars which my game can follow	The game only runs from 500 BC to 500 AD in order to have enough content to show off how the game works without bogging me down with just more content that has no extra “foundational” systems.
Logs	<p>The game will have logs which track and help the player progress through the game. The logs will have 5 different sections:</p> <ul style="list-style-type: none"> Combat logs – records which wars have been won or lost, and other war-based statistics Story line – shows the current year, and the other story based lines Tutorial logs – If the player wishes to, they can show the tutorial logs which explain the game step by step, when various progression milestones are made Auto-save logs – Shows a little message when the game is auto-saved Loot gained – When wars are won or lost, this log will show what loot is gained, and what bonuses and penalties winning/losing wars will have on the player’s resources <p>All of the logs can be toggled on/off, so the player can only show the tutorial logs, or story line and combat logs etc. There will also be an option to clear each section of logs separately. Logs can also be saved to a text file</p>	Tutorial logs doesn't exist is the only different thing.
Offline progress system	The offline progress system will only progress resource collection, until the storage is full	Not added due to time constraints of the project.
Saving	The auto-saves will happen every 2 minutes, and a message in the auto-saves section of the logs will show. The game can also be saved manually by the player, and they will have to load the games manually from file when they load the game back up. All auto-saves will save data into a text file	The same however in the game now autosaves are every 5 minutes rather than 2.

Combat	<p>There will be a grid, and when a war is started, the allies will start at the bottom on green tiles (owned land) and the enemies will start at the top on red tiles. The side that wins the most battle after a set amount of time wins the war. The set amount of time is the length of time the war lasted for in real world history.</p> <p>Combat will consist of 3 main mechanics:</p> <ul style="list-style-type: none">• Attack – The amount of damage the unit does to the enemy, regardless of how much it blocks• Health – The health of the unit, separate to block• Block – How much attack the enemy can “absorb” before taking damage. E.g. enemy does 30 attack, but player has 25 block, so only takes 5 damage <p>The combat is turn based. So the player attacks first, and then the enemy attacks. There will be an upgrade to automatically attack (when player is online) so that they don't have to press a button to send more troops every time they lose a battle</p>	<p>Every system works the same as I had thought in analysis.</p>
---------------	---	--

Description of Final Product

Limitations

I think the largest limitation of this game is how complex the game is to learn without any tutorial or tooltip system. That was something that all of my stakeholders had asked for but I did not successfully implement. This can be implemented through future developments.

Another part of the game that has been an issue throughout is the logs system. I think the idea of having separate tabs for each log type was good theoretically, but in practice it is just annoying to switch to different tabs to see the logs you want. On top of this, I think that the logs should be viewed whilst on any of the other menus so that the player can see exactly why they, say, can't buy an upgrade in real-time without having to exit the menu then and click on the upgrade tab.

Additionally, many stakeholders have said that there is not enough information provided by many of the menus on parts like combat information, years until next war, upgrades, housing etc. I wanted to avoid having too much information on the screen because it would be difficult to fit and not look good. Therefore, in future development I could add a tooltip system that activates a little box under the cursor when you hover over certain parts, to provide more information. This could block the view of some items and some players may be experienced enough to know what they do however, so there could be an option to turn this on or off.

There is also no online system, where on load the game could be automatically loaded from browser cache. This would not change the game, but perhaps make the user experience overall better, as they will not have to load their save game every single time they load the game up.

Lastly, although perhaps a small thing, is that because of the way the game has been built, there is no way any sort of multi-player system could be built in to introduce some sort of competition between players.

Maintenance and Development of Solution

The program has been coded using the functional programming method. This is usually nice for smaller programs, however the game ended up being over 1000 lines long so by the end it got quite messy and a bit of a pain to follow whilst debugging. Therefore, later on I could refactor the code so that it utilises the Object Orientated Programming method. This would make adding new features/content to the game far easier than currently, as I would not have to write similar code over and over again. The current code is “foundational” – it provides the base line for new content that could be added afterwards – however it is not ideal to continue in this manner without switching to using objects for most features because the code would end up becoming unnecessarily large.

Also mentioned in limitations is the tooltips/tutorial/logs systems that were not great. These could be improved upon in future developments.

Final Code

GlobalData.cs

```
using System;
using System.Collections.Generic;

namespace ThroughoutHistory {
    /// <summary>
    /// GlobalData stores ALL global scoped variables
    /// </summary>
    public static class GlobalData {
        // Resource data
        // [][0]Wood, [][1]Stone, [][2]Food, [][3]Metal
        public static List<int[]> resourcesData = new List<int[]> {
            new int[4] {0, 0, 0, 0}, // Amount
            new int[4] {1, 1, 1, 1}, // Rate, in milliseconds, by the timer (which is multiplied by 1000 later)
            new int[4] {100, 100, 100, 100}, // Capacity
            new int[4] {3, 1, 3, 1} // Gather multiplier (e.g. 2 would be +2/sec)
        };
        public static int scienceData = 0;
        public static int[] housingData = new int[3] {5, 10, 20}; // How much space each housing type gives
        public static int totalHousing = 5; // Total housing space
        public static int housingRemaining = 5;

        // Upgrades data
        public static List<int[]> upgradesCosts = new List<int[]> {
            new int[4] {50, 50, 50, 100}, // Storage Costs
            new int[4] {100, 100, 100, 200}, // Workers Costs
            new int[5] {100, 500, 1000, 3000, 5000}, // Science Costs
            new int[3] {50, 100, 500}, // Housing Costs
            new int[4] {200, 500, 300, 200} // Combat Costs ([][0]Health, [][1]Block, [][2]Damage, [][3]NoTroops)
        };
        public static int[] costMultipliers = new int[] {2, 4, 5, 2, 2}; // Storage, workers, science, housing, combat

        // Combat data
        // [][0]Player combat data, [][1]Enemy combat data
        public static List<int[]> combatData = new List<int[]> {
            new int[2] {100, 100}, // Health
            new int[2] {20, 20}, // Block (e.g. 2 is *2 block)
        }
    }
}
```

```
    new int[2] {50, 20}, // Damage
    new int[2] {1, 1} // No of troops
};

// War data
public static int tickCounter = 0;
public static int[] warTimes = new int[]
{180, 3240, 6000, 6800, 9600, 12300, 14160, 16920, 21000, 22200, 27540, 28200, 28920,
 32580, 33660, 39660, 47760, 50640}; // Measured in tick numbers (see tickCounter), each war lasts 10 minutes (600)
public static int currentWar = 0;
public static int warNumber = 0;
public static int year = -500;
public static int actualYear = 500;
public static string era = "BC";
public static string[] warNames = new string[18] {
    "War of Lake Regillius", "War of Corbio", "War of Veii", "War of Allia River", "Latin War", "War of Aquilonia",
    "First Punic War", "Second Punic War", "Third Punic War", "Achaean War", "War of Alesia", "War of Zela",
    "War of Actium", "War of the Medway", "The Battle of Watling Street", "First Persian Wars", "Second Persian Wars", "Third
Persian Wars"
};
};

// Current time variable
public static string currentTime;
}
```

MainForm.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace ThroughoutHistory {
    /// <summary>
    /// Responsible for showing the viewable parts of the program
    /// </summary>
    public partial class MainForm : Form {
        public MainForm() {
            InitializeComponent();
            // Set the spawn location to be at the top left corner
            Point spawnLocation = new Point(0, 0);
            Location = spawnLocation;
            // Set each section panel to an element of the panel array
            panelSections[0] = pnlResourceCollection;
            panelSections[1] = pnlCombat;
            panelSections[2] = pnlLogs;
            panelSections[3] = pnlFileHandling;
            panelSections[4] = pnlDebugging;

            // Hide every single panel except the resource collection panel (on program start up)
            for (int i = 0; i < panelSections.Length; i++) {
                // If the current panel is the tutorial one, show it
                if (panelSections[i] == pnlResourceCollection) {
                    panelSections[4].Show();
                // If not, hide it
                } else {
                    panelSections[i].Hide();
                }
            }

            // Output a first story log to explain what is going on
            rtxtStoryLogs.Text = "You feel an urge to grow the factory...";

            // Reset the combat grid to normal
            resetBoard();
        }
    }
}
```

```
#region customFormSetup
//Set up some consts and dlls to allow form dragging
private const int HT_CAPTION = 0x2;
private const int WM_NCLBUTTONDOWN = 0x00A1;
[DllImport("user32", CharSet = CharSet.Auto)]
private static extern bool ReleaseCapture();
[DllImport("user32", CharSet = CharSet.Auto)]
private static extern int SendMessage(IntPtr hWnd, int wMsg, int wParam, int lParam);

protected override void OnMouseDown(MouseEventArgs e) {
    //When a mouse button is pressed
    if (e.Button == MouseButtons.Left) {
        Rectangle rct = DisplayRectangle;
        //If the button pressed is the left button
        if (rct.Contains(e.Location)) {
            //Move the form
            ReleaseCapture();
            SendMessage(Handle, WM_NCLBUTTONDOWN, HT_CAPTION, 0);
        }
    }
}

void BtnCloseFormClick(object sender, EventArgs e) {
    Close();
}
#endregion

Panel[] panelSections = new Panel[5];

void MsMainItemClicked(object sender, ToolStripItemClickedEventArgs e) {
    // When a button on the menu strip is clicked, hide all panels
    for (int i = 0; i < panelSections.Length; i++) {
        panelSections[i].Hide();
    }
}

private void resourceCollectionToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the resource collection panel is clicked, show it
    pnlResourceCollection.Show();
```

```
}

private void combatToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the combat panel is clicked, show it
    pnlCombat.Show();
}

private void logsToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the logs panel is clicked, show it
    pnlLogs.Show();
}

private void saveLoadToolStripMenuItem_Click(object sender, EventArgs e) {
    // When the file handling panel is clicked, show it
    pnlFileHandling.Show();
}

private void debuggingToolStripMenuItem_Click(object sender, EventArgs e) {
    pnlDebugging.Show();
}

void currentTime() {
    // Get current time in UTC from system clock
    GlobalData.currentTime = DateTime.UtcNow.ToString("T");
    // Print into save/load label
    lblTime.Text = ("Current time (UTC): " + GlobalData.currentTime.ToString() + ".");
}

private void btnChangeTick_Click(object sender, EventArgs e) {
    GlobalData.tickCounter = Convert.ToInt32(nudChangeTick.Value);
}
}
```

ResourceCollection.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Timers;

namespace ThroughoutHistory {
    public partial class MainForm {
        bool[] btnToggled = new bool[4] {false, false, false, false};
        bool isPauseToggled = false;

        #region Buttons & Timers
        void BtnGatherClick(object sender, EventArgs e) {
            toggleButton(0);
            checkOtherButtons(0);
        }

        void BtnQuarrierClick(object sender, EventArgs e) {
            toggleButton(1);
            checkOtherButtons(1);
        }

        void BtnFarmClick(object sender, EventArgs e) {
            toggleButton(2);
            checkOtherButtons(2);
        }

        void BtnMineClick(object sender, EventArgs e) {
            toggleButton(3);
            checkOtherButtons(3);
        }

        void GlobalTimerTick(object sender, EventArgs e) {
            GlobalData.tickCounter += 1;
            resourceCollection(0, btnGather, lblWoodAmount, lblWoodRate, pbWood);
            resourceCollection(1, btnQuarrier, lblStoneAmount, lblStoneRate, pbStone);
            resourceCollection(2, btnFarm, lblFoodAmount, lblFoodRate, pbFood);
            resourceCollection(3, btnMine, lblMetalAmount, lblMetalRate, pbMetal);

            currentTime();
        }
    }
}
```

```
yearCounter();
startWar();

// Call the housing and science functions to update values if needed
    housing();
science();
}

// Fun temp stuff (might actually add into finished game)
private void btnPauseGame_Click(object sender, EventArgs e) {
    if (isPauseToggled) {
        globalTimer.Start();
        combatTimer.Start();
        isPauseToggled = false;
    } else {
        globalTimer.Stop();
        combatTimer.Stop();
        isPauseToggled = true;
    }
}
#endregion

#region Procedures
void yearCounter() {
    // If tick counter has reached a multiple of 60
    if ((GlobalData.tickCounter % 60) == 0) {
        // Increment year by 1
        GlobalData.year++;
        // If the year is negative
        if (GlobalData.year < 0) {
            // Get the positive year version
            GlobalData.actualYear = (GlobalData.year * -2) + GlobalData.year;
            // Set the era to BC
            GlobalData.era = "BC";
        } else {
            GlobalData.actualYear = GlobalData.year;
            // Set the era to AD
            GlobalData.era = "AD";
        }
        rtxtStoryLogs.Text += ("\nA new year has begun: " + GlobalData.actualYear + GlobalData.era);
    }
}
```

```
        }

    }

/// <summary>
/// Calculates how much each resource will be increased by each tick
/// </summary>
/// <param name="arraySlot">The resource type</param>
/// <param name="buttonType">The button type</param>
/// <param name="labelAmounts">The resource type's labelAmount</param>
/// <param name="labelRates">The resource type's labelRate</param>
/// <param name="pBarType">The progress bar type</param>
void resourceCollection(int arraySlot, Button buttonType, Label labelAmounts, Label labelRates, ProgressBar pBarType) {
    int currentRate = GlobalData.resourcesData[1][arraySlot];
    int currentAmount = GlobalData.resourcesData[0][arraySlot];
    int currentCapacity = GlobalData.resourcesData[2][arraySlot];

    // If button is toggled off
    if (!btnToggled[arraySlot]) {
        // Set its colour to grey
        buttonType.BackColor = SystemColors.ControlLight;
    } else {
        // Set its colour to brown, and increase the currentRate by gatherMultiplier
        currentRate += GlobalData.resourcesData[3][arraySlot];
        buttonType.BackColor = Color.Chocolate;
    }

    // If the currentAmount = 0, set it to 1 just in case it is divided by 0 (which would crash program)
    amountCheck();

    // Set the text of label rates
    labelRates.Text = ("+" + currentRate + "/sec");
    // Set the progress bar maximum to the capacity of the resource
    pBarType.Maximum = currentCapacity;

    // If there is space to increase the value of the bar
    if (currentAmount <= (currentCapacity - currentRate)) {
        currentAmount += currentRate;
    } else {
        currentAmount = currentCapacity;
    }
}
```

```
// Update the progress bar
pBarType.Value = currentAmount;
// Set the new current amount to show in the output label
GlobalData.resourcesData[0][arraySlot] = currentAmount;
labelAmounts.Text = currentAmount.ToString();
}

void housing() {
    // Make locally scoped variables for easier access
    int currentHousingCapacity = GlobalData.totalHousing;
    int currentHousingRemaining = GlobalData.housingRemaining;

    // Set the current housing amount to the amount in global data
    lblHousingCapacity.Text = currentHousingCapacity.ToString() + " housing capacity";
    lblHousingSpace.Text = currentHousingRemaining.ToString() + " housing remaining";
    // Set the value and maximum of the housing progress bar
    pbHousing.Maximum = currentHousingCapacity;
    pbHousing.Value = currentHousingRemaining;
}

void science() {
    // Make locally scoped variables for easier access
    int currentScience = GlobalData.scienceData;

    // Set the current science amount in the global data
    lblScienceAmount.Text = currentScience.ToString();
}

    /// <summary>
    /// Makes sure the amount of a resource NEVER goes below 1
    /// </summary>
void amountCheck() {
    for (int i = 0; i < 4; i++) {
        if (GlobalData.resourcesData[i][0] < 1) {
            GlobalData.resourcesData[i][0] = 1;
        }
    }
}
```

```
/// <summary>
/// When a button is clicked, the button is toggled on, or if it is already on, toggle it off
/// </summary>
/// <param name="arraySlot">The array slot of the btnToggled array</param>
void toggleButton(int arraySlot) {
    // If button is toggled on, set it to false
    if (btnToggled[arraySlot]) {
        btnToggled[arraySlot] = false;
    // If button is toggled off, set is to true, then make sure all other buttons are off (calling checkOtherButtons)
    } else {
        btnToggled[arraySlot] = true;
        checkOtherButtons(arraySlot);
    }
}

/// <summary>
/// When a button is toggled on, make sure all other buttons are toggled off
/// </summary>
/// <param name="arraySlot">The array slot of the btnToggled array</param>
void checkOtherButtons(int arraySlot) {
    //Check if the button being clicked is being toggled on
    if (btnToggled[arraySlot]) {
        //Loop through all the buttons
        for (int i = 0; i < btnToggled.Length; i++) {
            // Set all the buttons to false
            btnToggled[i] = false;
        }
        // Then set currently clicked button to true
        btnToggled[arraySlot] = true;
    }
}
#endregion
}
```

Upgrades.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace ThroughoutHistory {
    public partial class MainForm {
        #region Buttons
        /// <summary>
        /// All storage buttons
        /// </summary>
        private void btnWoodStorage_Click(object sender, EventArgs e) {
            buyingStorage(GlobalData.upgradesCosts[0][0], 0, 0, 1000, lblWoodStorageCost);
        }

        private void btnStoneStorage_Click(object sender, EventArgs e) {
            buyingStorage(GlobalData.upgradesCosts[0][1], 0, 1, 1000, lblStoneStorageCost);
        }

        private void btnFoodStorage_Click(object sender, EventArgs e) {
            buyingStorage(GlobalData.upgradesCosts[0][2], 0, 2, 1000, lblFoodStorageCost);
        }

        private void btnMetalStorage_Click(object sender, EventArgs e) {
            buyingStorage(GlobalData.upgradesCosts[0][3], 0, 3, 1000, lblMetalStorageCost);
        }

        /// <summary>
        /// All workers buttons
        /// </summary>
        private void btnGatherer_Click(object sender, EventArgs e) {
            buyingWorkers(GlobalData.upgradesCosts[1][0], 2, 0, 5, lblGathererCost);
        }

        private void btnQuarrying_Click(object sender, EventArgs e) {
            buyingWorkers(GlobalData.upgradesCosts[1][1], 2, 1, 2, lblQuarrierCost);
        }

        private void btnButcher_Click(object sender, EventArgs e) {
            buyingWorkers(GlobalData.upgradesCosts[1][2], 2, 2, 5, lblButcherCost);
        }
    }
}
```

```
}

private void btnMiner_Click(object sender, EventArgs e) {
    buyingWorkers(GlobalData.upgradesCosts[1][3], 2, 3, 1, lblMinerCost);
}

private void btnTroop_Click(object sender, EventArgs e) {
    buyingTroops(GlobalData.upgradesCosts[4][3], 3, lblTroopCost);
}

/// <summary>
/// All housing buttons
/// </summary>
private void btnShack_Click(object sender, EventArgs e) {
    buyingHousing(GlobalData.upgradesCosts[3][0], 1, 0, lblShackCost);
}

private void btnBootcamp_Click(object sender, EventArgs e) {
    buyingHousing(GlobalData.upgradesCosts[3][1], 1, 1, lblBootcampCost);
}

private void btnBarracks_Click(object sender, EventArgs e) {
    buyingHousing(GlobalData.upgradesCosts[3][2], 1, 2, lblBarracksCost);
}

/// <summary>
/// All combat buttons
/// </summary>
private void btnHealth_Click(object sender, EventArgs e) {
    buyingCombat(GlobalData.upgradesCosts[4][0], 3, 0, 100, lblHealthCost);
}

private void btnBlock_Click(object sender, EventArgs e) {
    buyingCombat(GlobalData.upgradesCosts[4][1], 3, 1, 250, lblBlockCost);
}

private void btnAttack_Click(object sender, EventArgs e) {
    buyingCombat(GlobalData.upgradesCosts[4][2], 3, 2, 150, lblAttackCost);
}
```

```
/// <summary>
/// Research buttons
/// </summary>
private void btnAquaducts_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][0], 0, lblAquaductsCost);
}

private void btnStampMill_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][1], 1, lblStampMillCost);
}

private void btnTripHammer_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][2], 2, lblTripHammerCost);
}

private void btnHushing_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][3], 3, lblHushingCost);
}

private void btnVilla_Click(object sender, EventArgs e) {
    buyingResearch(GlobalData.upgradesCosts[2][4], 4, lblVillaCost);
}
#endregion

#region Procedures
/// <param name="cost">The cost of the storage</param>
/// <param name="typeToBuy">What resource is needed to buy it</param>
/// <param name="typeToBuyFor">What resource storage it is being bought for</param>
/// <param name="storageIncrease">The amount of storage the upgrade increased by</param>
void buyingStorage(int cost, int typeToBuy, int typeToBuyFor, int storageIncrease, Label labelCosts) {
    // If the player does not have enough wood
    if (GlobalData.resourcesData[0][typeToBuy] < cost) {
        // Print to upgrades tab in logs
        rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough wood for this upgrade");
    } else {
        // Deduct the cost from the number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase the cost of this upgrade
        GlobalData.upgradesCosts[0][typeToBuyFor] *= GlobalData.costMultipliers[0];
        // Increase the storage amount by the amount specified
    }
}
```

```
        GlobalData.resourcesData[2][typeToBuyFor] += storageIncrease;
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[0][typeToBuyFor] + " wood");
    }
}

/// <param name="cost">The cost of the workers</param>
/// <param name="typeToBuy">What resource is needed to buy it</param>
/// <param name="typeToBuyFor">The resource worker it is being bought for</param>
void buyingWorkers(int cost, int typeToBuy, int typeToBuyFor, int rateIncrease, Label labelCosts) {
    // If the player does not have enough food OR enough housing space
    if (GlobalData.resourcesData[0][typeToBuy] < cost || GlobalData.housingRemaining == 0) {
        // Print to upgrades tab in logs
        rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough food for this upgrade");
    } else {
        // Deduct the cost from the number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase the cost of this upgrade
        GlobalData.upgradesCosts[1][typeToBuyFor] *= GlobalData.costMultipliers[1];
        // Increase the rate of the resource
        GlobalData.resourcesData[1][typeToBuyFor] += rateIncrease;
        // Decreasing the housing space by 1
        GlobalData.housingRemaining -= 1;
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[1][typeToBuyFor] + " food, 1 housing");
    }
}

/// <param name="cost">Cost of the troop</param>
/// <param name="typeToBuy">What resource is needed to buy the troop</param>
void buyingTroops(int cost, int typeToBuy, Label labelCosts) {
    // If there is not enough metal or housing space
    if (GlobalData.resourcesData[0][typeToBuy] < cost || GlobalData.housingRemaining == 0) {
        // Print to upgrades tab in logs
        rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough metal or housing for this upgrade");
    } else {
        // Deduct cost from number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase cost of this upgrade
        GlobalData.upgradesCosts[4][3] *= GlobalData.costMultipliers[1];
```

```
// Increase the number of troops in the army
GlobalData.combatData[3][0] += 1;
// Decrease the housing space by 1
GlobalData.housingRemaining -= 1;
// Print the new cost to the cost label
labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[4][3] + " metal, 1 housing");
// Print the new number of troops to upgrades logs
rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": Troop number increased by 1, total is " +
GlobalData.combatData[3][0].ToString());
}

/// <param name="cost">Cost of upgrade</param>
/// <param name="typeToBuy">Resource type needed to buy upgrade</param>
/// <param name="housingType">Type of housing being upgraded</param>
void buyingHousing(int cost, int typeToBuy, int housingType, Label labelCosts) {
    // If there are enough resources to buy the upgrade
    if (GlobalData.resourcesData[0][typeToBuy] < cost) {
        // Print to upgrades tab in logs
        rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough stone for this upgrade");
    } else {
        // Deduct cost from number of resources
        GlobalData.resourcesData[0][typeToBuy] -= cost;
        // Increase cost of this upgrade
        GlobalData.upgradesCosts[3][housingType] *= GlobalData.costMultipliers[3];
        // Increase the total housing and housing remaining this upgrade provides
        GlobalData.totalHousing += GlobalData.housingData[housingType];
        GlobalData.housingRemaining += GlobalData.housingData[housingType];
        // Print the new cost to the cost label
        labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[3][housingType] + " stone");
    }
}

/// <param name="cost">Cost of upgrade</param>
/// <param name="typeToBuy">Type of resource needed to buy upgrade</param>
/// <param name="combatType">Type of combat being bought for</param>
/// <param name="buff">Amount the combat type is being buffed by</param>
void buyingCombat(int cost, int typeToBuy, int combatType, int buff, Label labelCosts) {
    // If the player does not have enough resources
    if (GlobalData.resourcesData[0][typeToBuy] < cost) {
```

```
// Print message into upgrades logs
    rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough metal for this upgrade");
} else {
    // Deduct cost from number of resources
    GlobalData.resourcesData[0][typeToBuy] -= cost;
    // Increase cost of this upgrade
    GlobalData.upgradesCosts[4][combatType] *= GlobalData.costMultipliers[4];
    // Buff the combat type
    GlobalData.combatData[combatType][0] += buff;
    // Print the new cost to the cost label
    labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[4][combatType] + " metal");
    switch (combatType) {
        case 0:
            // Print health increase to logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime + " Troop health increased by 100, total is " +
GlobalData.combatData[0][0].ToString());
            break;
        case 1:
            // Print block increase to logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime + " Troop block increased by 250, total is " +
GlobalData.combatData[1][0].ToString());
            break;
        case 2:
            // Print attack increase to logs
            rtxtCombat.Text += ("\n" + GlobalData.currentTime + " Troop attack increased by 150, total is " +
GlobalData.combatData[2][0].ToString());
            break;
    }
}

/// <param name="cost">Cost of upgrade</param>
/// <param name="researchType">Type of research being bought</param>
void buyingResearch(int cost, int researchType, Label labelCosts) {
    // If the player does not have enough science
    if (GlobalData.scienceData < cost) {
        // Print in upgrades logs
        rtxtUpgrades.Text += ("\n" + GlobalData.currentTime + ": You do not have enough science for this upgrade");
    } else {
        // Deduct cost from total science
    }
}
```

```
GlobalData.scienceData -= cost;
// Increase cost
GlobalData.upgradesCosts[2][researchType] *= GlobalData.costMultipliers[2];
// Output new cost
labelCosts.Text = ("Cost: " + GlobalData.upgradesCosts[2][researchType] + " science");

switch (researchType) {
    // Aqueducts
    case 0:
        GlobalData.resourcesData[1][0] *= 5;
        GlobalData.resourcesData[1][2] *= 5;
        rtxtStoryLogs.Text += ("\n" + "You have learned the art of aqueducts! These towering structures made of stone
automate the carrying of water into your farms.");
        break;
    // Stamp-mill
    case 1:
        GlobalData.resourcesData[1][1] *= 2;
        rtxtStoryLogs.Text += ("\n" + "You have created the blueprints for stamp mills! These crush metal ore by
pounding them to bits, making it faster to extract metal!");
        break;
    // Trip-hammer
    case 2:
        GlobalData.resourcesData[1][3] *= 2;
        rtxtStoryLogs.Text += ("\n" + "You have invented the trip-hammer! This will help you to shape your metals
into more usable shapes!");
        break;
    // Hushing
    case 3:
        GlobalData.resourcesData[1][3] *= 5;
        rtxtStoryLogs.Text += ("\n" + "You have stolen the secret of hushing from another civilisation! It allows you
to use floods of water to reveal mineral veins.");
        break;
    // Villa
    case 4:
        GlobalData.totalHousing *= 3;
        rtxtStoryLogs.Text += ("\n" + "You have earned yourself enough to construct a villa! This magnificent
structure increases your housing space by 3 times!");
        break;
}
```

[name]

Candidate No: xxxx

Centre No:xxxxx

```
    }  
    #endregion  
}  
}
```

Combat.cs

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Drawing;
using System.Runtime.InteropServices;

namespace ThroughoutHistory {
    public partial class MainForm {
        /// <summary>
        /// Runs every 5 ticks (seconds)
        /// </summary>
        private void combatTimer_Tick(object sender, EventArgs e) {
            warTicks += combatTimer.Interval / 1000;
            calculateAttack();
            winWarCheck();
        }

        #region Variables
        int maxPlayerHealth;
        int playerBlock;
        int playerDamage;

        int maxEnemyHealth;
        int enemyBlock;
        int enemyDamage;

        int enemyHealth = 100;
        int playerHealth = 100;

        bool playerTurn = true;
        int warTicks = 0;
        Random rng = new Random();
        int playerTiles = 10;
        const int ROWS = 4;
        const int COLUMNS = 5;
        const int PLAYER = 1;
        const int ENEMY = 0;
        int[,] grid = new int[4, 5]
    {
```

```
{0, 0, 0, 0, 0},  
{0, 0, 0, 0, 0},  
{1, 1, 1, 1, 1},  
{1, 1, 1, 1, 1}  
}; //Where 0 is red and 1 is green (enemy and player occupied respectively)  
#endregion  
  
/// <summary>  
/// Runs a bunch of stuff when a new war starts  
/// </summary>  
void startWar() {  
    // Loop number of wars  
    for (int i = 0; i < GlobalData.warTimes.Length; i++) {  
        // If the current tick is equivalent to the current war time  
        if (GlobalData.tickCounter == GlobalData.warTimes[i]) {  
            // New war starts  
            GlobalData.currentWar = GlobalData.warTimes[i];  
            GlobalData.warNumber++;  
            warTicks = 0;  
            // Print this in logs  
            rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": New war called " + GlobalData.warNames[i] + "  
started!");  
            lblWarDescription.Text = ("Current war: " + GlobalData.warNames[i] + ".");  
            // Reset the board  
            resetBoard();  
            // Calculate new enemy stats  
            calculateNewEnemyStats();  
            // Start the combat timer  
            combatTimer.Start();  
  
            // Update all the necessary variables  
            maxPlayerHealth = GlobalData.combatData[0][0] * GlobalData.combatData[3][0];  
            playerBlock = GlobalData.combatData[1][0] * GlobalData.combatData[3][0];  
            playerDamage = GlobalData.combatData[2][0] * GlobalData.combatData[3][0];  
  
            maxEnemyHealth = GlobalData.combatData[0][1] * GlobalData.combatData[3][1];  
            enemyBlock = GlobalData.combatData[1][1] * GlobalData.combatData[3][1];  
            enemyDamage = GlobalData.combatData[2][1] * GlobalData.combatData[3][1];  
        } else {  
            lblCurrentYear.Text = ("Year: " + GlobalData.actualYear.ToString() + GlobalData.era + ".");  
        }  
    }  
}
```

```
        }
    }

/// <summary>
/// Resets the board at the start of every war
/// </summary>
void resetBoard() {
    // Setup the graphics and brushes
    Bitmap bitmap = new Bitmap(1000, 1000);
    Graphics GFX = Graphics.FromImage(bitmap);
    SolidBrush greenBrush = new SolidBrush(Color.Green);
    SolidBrush redBrush = new SolidBrush(Color.Red);

    // Loop through each column (all are same)
    for (int x = 0; x < COLUMNS; x++) {
        // Loop through first 2 rows
        for (int y = 0; y < ROWS; y++) {
            // Paint the red tiles
            GFX.FillRectangle(redBrush, x * 65, y * 65, 60, 60);
            grid[y, x] = ENEMY;
        }
        // Loop through last 2 rows
        for (int y = 2; y < ROWS; y++) {
            // Paint the green tiles
            GFX.FillRectangle(greenBrush, x * 65, y * 65, 60, 60);
            grid[y, x] = PLAYER;
        }
    }
    // Show the image in the picture box
    pbxGrid.Image = bitmap;
}

/// <summary>
/// Calculates each attack turn
/// </summary>
void calculateAttack() {
    // If it is the player's turn
    if (playerTurn) {
        playerTurn = false;
```

```
// Add some random into the damage, using a bonus damage
int bonusDamagePlayer = rng.Next(1, 3);
int bonusDamage = 0;
switch (bonusDamagePlayer) {
    case 1:
        bonusDamage = GlobalData.combatData[2][0] / 5;
        break;
    case 2:
        bonusDamage = GlobalData.combatData[2][0] / 2;
        break;
}
// Work out how much damage the player does
int blockedPlayerDamage = (playerDamage + bonusDamage) - enemyBlock;
enemyHealth -= blockedPlayerDamage;

// If the enemy health is 0 or less
if (enemyHealth <= 0) {
    // Give the player a tile
    playerTiles += 1;
    int enemyTiles = 20 - playerTiles;
    lblEnemyTiles.Text = ("Enemy tiles: " + enemyTiles);
    lblPlayerTiles.Text = ("Allied tiles: " + playerTiles);

    int i = 19;
    // While the current tile is owned by the player
    while(grid[i / 5,i % 5] == PLAYER) {
        i--;
    }
    // Next tile must be next in line to be claimed, thus set this to player
    grid[i / 5, i % 5] = PLAYER;

    updateBoard();

    // Reset player and enemy health back to normal
    enemyHealth = maxEnemyHealth;
    playerHealth = maxPlayerHealth;
}
// 0/0 Health, 0 attack, 0 block
lblEnemyStats.Text = String.Format("{0}/{1}, {2} attack, {3} block", enemyHealth, maxEnemyHealth, enemyDamage,
enemyBlock);
```

```
lblPlayerStats.Text = String.Format("{0}/{1}, {2} attack, {3} block, +{4} bonus!", playerHealth, maxPlayerHealth,
playerDamage, playerBlock, bonusDamage);
rtxtTempOutput.Text += String.Format("\nPlayer's turn. New enemy health is {0} out of {1} total health, player tiles
is {2}.", enemyHealth, maxEnemyHealth, playerTiles);
} else {
    playerTurn = true;
    // Add some random into the damage, using a bonus damage
    int bonusDamageEnemy = rng.Next(1, 3);
    int bonusDamage = 0;
    switch (bonusDamageEnemy) {
        case 1:
            bonusDamage = GlobalData.combatData[2][0] / 5;
            break;
        case 2:
            bonusDamage = GlobalData.combatData[2][0] / 10;
            break;
    }
    // Work out how much damage the enemy does
    int blockedEnemyDamage = (enemyDamage + bonusDamage) - playerBlock;
    playerHealth -= blockedEnemyDamage;

    // If the player health is 0 or less
    if (playerHealth <= 0) {
        // Give the enemy the tile
        playerTiles -= 1;
        int enemyTiles = 20 - playerTiles;
        lblEnemyTiles.Text = ("Enemy tiles: " + enemyTiles);
        lblPlayerTiles.Text = ("Allied tiles: " + playerTiles);

        int i = 0;
        // While the current tile is owned by the enemy
        while (grid[i / 5, i % 5] == ENEMY) {
            i++;
        }
        // Next tile must be next in line to be claimed, thus set this to enemy
        grid[i / 5, i % 5] = ENEMY;

        updateBoard();

        // Reset player and enemy health to normal
    }
}
```

```
        playerHealth = maxPlayerHealth;
        enemyHealth = maxEnemyHealth;
    }
    // 0/0 Health, 0 attack, 0 block
    lblEnemyStats.Text = String.Format("{0}/{1}, {2} attack, {3} block, +{4} bonus!", enemyHealth, maxEnemyHealth,
enemyDamage, enemyBlock, bonusDamage);
    lblPlayerStats.Text = String.Format("{0}/{1}, {2} attack, {3} block", playerHealth, maxPlayerHealth, playerDamage,
playerBlock);
    rtxtTempOutput.Text += String.Format("\nEnemy's turn. New player health is {0} out of {1} total health, player tiles
is {2}.", playerHealth, maxPlayerHealth, playerTiles);
}
}

/// <summary>
/// Updates the board based off the grid 2D array
/// </summary>
void updateBoard() {
    // Setup graphics and brushes
    Bitmap bitmap = new Bitmap(1000, 1000);
    Graphics GFX = Graphics.FromImage(bitmap);
    SolidBrush[] brushes = { new SolidBrush(Color.Red), new SolidBrush(Color.Green) };
    const int GRIDSIZE = 65;

    // Loop through each column
    for (int x = 0; x < COLUMNS; x++) {
        // Loop through each row
        for (int y = 0; y < ROWS; y++) {
            // Redraw the grid according to what the values in the grid variable are
            GFX.FillRectangle(brushes[grid[y, x]], Rectangle.FromLTRB(x * (GRIDSIZE), y * (GRIDSIZE), x * (GRIDSIZE) + 60, y
* (GRIDSIZE) + 60));
        }
    }
    pbxGrid.Image = bitmap;
}

/// <summary>
/// Checks if a war has been won or not
/// </summary>
void winWarCheck() {
    // A side wins all tiles, or the war timer has run out
```

```
if (playerTiles == 0 || playerTiles == 20 || warTicks == 600) {
    // Reset allied and enemy tiles count labels
    lblPlayerTiles.Text = "Allied tiles: 10";
    lblEnemyTiles.Text = "Enemy tiles: 10";
    lblPlayerStats.Text = "No war running";
    lblEnemyStats.Text = "No war running";
    lblWarDescription.Text = "Current war: None - Check loot and combat logs.";
    resetBoard();
    // If both sides have the same number of tiles
    if (playerTiles == 10) {
        // Work out new enemy stats
        int randomNum = rng.Next(3, 5);
        for (int i = 0; i < 3; i++) {
            GlobalData.combatData[i][1] *= randomNum;
        }
        GlobalData.combatData[3][1] += randomNum;
        // Print draw into logs
        rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": Draw");
        // Work out how much loot player receives
        calculateLoot(0);
        // If the enemy has more tiles than the player
    } else if (playerTiles == 0) {
        // Work out new enemy stats
        int randomNum = rng.Next(3, 7);
        for (int i = 0; i < 3; i++) {
            GlobalData.combatData[i][1] *= randomNum;
        }
        GlobalData.combatData[3][1] += randomNum;
        // Print enemy wins into logs
        rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have lost the war, the enemy will get much
stronger :(");
        // Work out how much loot player receives
        calculateLoot(1);
        // If the player has more tiles than the enemy
    } else if (playerTiles == 20) {
        // Work out new enemy stats
        int randomNum = rng.Next(3, 5);
        for (int i = 0; i < 3; i++) {
            GlobalData.combatData[i][1] *= randomNum;
        }
```

```
        GlobalData.combatData[3][1] += randomNum;
        // Print player wins into logs
        rtxtCombat.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have won the war! The enemy will not get
that much stronger!");
        // Work out how much loot player receives
        calculateLoot(2);
    }
    // Stop the combat timer until new war starts
    combatTimer.Stop();
}

/// <summary>
/// Calculates loot player gains/loses based on war outcome
/// </summary>
/// <param name="outcome">Value of outcome</param>
void calculateLoot(int outcome) {
    switch (outcome) {
        case 0:
            // Gets 200* war number for each resource
            const int MULTIPLIERDRAW = 200;
            for (int i = 0; i < 4; i++) {
                GlobalData.resourcesData[0][i] += MULTIPLIERDRAW * GlobalData.warNumber;
            }
            // Gets 1000 * war number of science
            const int MULTIPLIERWAR = 1000;
            GlobalData.scienceData += MULTIPLIERWAR * GlobalData.warNumber;
            // Print values to logs
            rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have gained " + (MULTIPLIERDRAW *
GlobalData.warNumber).ToString() + " of each resource + " + (MULTIPLIERWAR * GlobalData.warNumber).ToString() + " science!");
            break;
        case 1:
            // Loses 5-10 troops to the enemy
            int randomNum = rng.Next(5, 10);
            GlobalData.combatData[3][0] -= randomNum;
            if (GlobalData.combatData[3][0] < 1) {
                GlobalData.combatData[3][0] = 1;
            }
            GlobalData.combatData[3][1] += randomNum;
            // Gets 500* war number of each resource
```

```
const int MULTIPLIERLOSS = 100;
for (int i = 0; i < 4; i++) {
    GlobalData.resourcesData[0][i] += MULTIPLIERLOSS * GlobalData.warNumber;
}
// Gets 1000 * war number of science
GlobalData.scienceData += MULTIPLIERWAR * GlobalData.warNumber;
// Print values to logs
rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have lost " + randomNum.ToString() + " troops
to the enemy, but gained " + (MULTIPLIERLOSS * GlobalData.warNumber).ToString() + " of each resource + " + (MULTIPLIERWAR *
GlobalData.warNumber).ToString() + " science!");
break;
case 2:
// Gets half number of troops enemy had
int noEnemyTroops = GlobalData.combatData[3][1];
// If it is odd, take 1 then half it
if (noEnemyTroops % 2 != 0) {
    noEnemyTroops -= 1;
}
GlobalData.combatData[3][0] += (noEnemyTroops / 2);
// Gets 500* war number of each resource
const int MULTIPLIERWIN = 500;
for (int i = 0; i < 4; i++) {
    GlobalData.resourcesData[0][i] += MULTIPLIERWIN * GlobalData.warNumber;
}
// Gets 1000 * war number of science
const int MULTIPLIERWARWIN = 5000;
GlobalData.scienceData += MULTIPLIERWARWIN * GlobalData.warNumber;
// Print values to logs
rtxtLoot.Text += ("\n" + GlobalData.currentTime.ToString() + ": You have gained half of the enemy's troops, and
gained " + (MULTIPLIERWIN * GlobalData.warNumber).ToString() + " of each resource + " + (MULTIPLIERWARWIN *
GlobalData.warNumber).ToString() + " science!");
break;
}
}

/// <summary>
/// Calculates the stats of the enemies on new war start
/// </summary>
void calculateNewEnemyStats() {
// Enemy health 100-200% of player's (1x-2x)
```

```
int randomHealth = rng.Next(1, 3); // MaxVal is exclusive
GlobalData.combatData[0][1] = GlobalData.combatData[0][0] * randomHealth;
// Enemy block 100-200% of player's (1x-2x)
int randomBlock = rng.Next(1, 3);
GlobalData.combatData[1][1] = GlobalData.combatData[1][0] * randomBlock;
// Enemy damage 50-100% of player's (0.5x-1x)
int randomAttack = rng.Next(1, 3);
// Get around having to use random doubles
if (randomAttack == 1) {
    GlobalData.combatData[2][1] = GlobalData.combatData[2][0] / 2;
} else {
    GlobalData.combatData[2][1] = GlobalData.combatData[2][0];
}
}
}
```

Logs.cs

```
using System;

namespace ThroughoutHistory {
    /// <summary>
    /// Does all logs based processing
    /// </summary>
    public partial class MainForm {
        void clearText() {
            // Get index of selected tab
            int selectedTab = tabControl2.SelectedIndex;
            switch (selectedTab) {
                // Story
                case 0:
                    rtxtStoryLogs.Text = "";
                    break;
                // Upgrades
                case 1:
                    rtxtUpgrades.Text = "";
                    break;
                // Combat
                case 2:
                    rtxtCombat.Text = "";
                    break;
                // Loot
                case 3:
                    rtxtLoot.Text = "";
                    break;
                // Saves
                case 4:
                    rtxtSaves.Text = "";
                    break;
            }
        }

        private void btnClearLogs_Click(object sender, EventArgs e) {
            clearText();
        }
    }
}
```

FileHandling.cs

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Collections.Generic;

namespace ThroughoutHistory {
    public partial class MainForm {
        bool isAutosaveToggled = false;
        int autosaveTick = 0;

        private void autosaveTimer_Tick(object sender, EventArgs e) {
            autosaveTick++;
            // If a multiple of 300 seconds (or 5 mins) has been reached
            if ((autosaveTick % 300) == 0) {
                // Call saving function with autosave setting
                saveToFile(true);
                rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": The game has been autosaved!");
            }
        }

        #region Buttons
        private void btnSave_Click(object sender, EventArgs e) {
            saveToFile(false);
        }

        private void btnLoad_Click(object sender, EventArgs e) {
            loadFromFile();
        }

        private void btnAutosaveOption_Click(object sender, EventArgs e) {
            if (isAutosaveToggled) {
                autosaveTimer.Stop();
                isAutosaveToggled = false;
                lblAutosaveStatus.Text = ("Autosaving off.");
            } else {
                autosaveTimer.Start();
                isAutosaveToggled = true;
                lblAutosaveStatus.Text = ("Autosave interval: 5 minutes");
            }
        }
    }
}
```

```
}

private void btnDelete_Click(object sender, EventArgs e) {
    deleteSave();
}

#endregion

/// <summary>
/// Uses the selected file to load info into the game
/// </summary>
void loadFromFile() {
    // Set filter to only text files
    openDialogue.Filter = "Text|*.txt";
    // Show dialog
    DialogResult result = openDialogue.ShowDialog();
    // If user presses OK button
    if (result == DialogResult.OK) {
        // Get file to load
        string selectedFile = openDialogue.FileName;
        // Load data from file
        try {
            using (StreamReader sr = File.OpenText(selectedFile)) {
                string s = "";
                int lineNum = 1;
                // While not end of file
                while ((s = sr.ReadLine()) != null) {
                    // Split each part into this currentSplits array
                    string[] currentSplits = s.Split('#');
                    // Write to global data variables based on which line is on
                    switch (lineNum) {
                        // Amount
                        case 1:
                            for (int i = 0; i < currentSplits.Length; i++) { GlobalData.resourcesData[0][i] =
Convert.ToInt32(currentSplits[i]); }
                            break;
                        // Rate
                        case 2:
                            for (int i = 0; i < currentSplits.Length; i++) { GlobalData.resourcesData[1][i] =
Convert.ToInt32(currentSplits[i]); }
                    }
                }
            }
        }
    }
}
```

```
        break;
    // Capacity
    case 3:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.resourcesData[2][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Gather multiplier
    case 4:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.resourcesData[3][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Science data
    case 5:
        GlobalData.scienceData = Convert.ToInt32(currentSplits[0]);
        break;
    // Total housing
    case 6:
        GlobalData.totalHousing = Convert.ToInt32(currentSplits[0]);
        break;
    // Housing remaining
    case 7:
        GlobalData.housingRemaining = Convert.ToInt32(currentSplits[0]);
        break;
    // Storage costs
    case 8:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.upgradesCosts[0][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Workers costs
    case 9:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.upgradesCosts[1][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Science costs
    case 10:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.upgradesCosts[2][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Housing costs
    case 11:
```

```
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.upgradesCosts[3][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Combat costs
    case 12:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.upgradesCosts[4][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Cost multipliers
    case 13:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.costMultipliers[i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Health
    case 14:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.combatData[0][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Block
    case 15:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.combatData[1][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Damage
    case 16:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.combatData[2][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // No of troops
    case 17:
        for (int i = 0; i < currentSplits.Length; i++) { GlobalData.combatData[3][i] =
Convert.ToInt32(currentSplits[i]); }
        break;
    // Tick counter
    case 18:
        GlobalData.tickCounter = Convert.ToInt32(currentSplits[0]);
        break;
    // Current war
    case 19:
        GlobalData.currentWar = Convert.ToInt32(currentSplits[0]);
```

```
        break;
    // War number
    case 20:
        GlobalData.warNumber = Convert.ToInt32(currentSplits[0]);
        break;
    // Year
    case 21:
        GlobalData.year = Convert.ToInt32(currentSplits[0]);
        break;
    // Actual year
    case 22:
        GlobalData.actualYear = Convert.ToInt32(currentSplits[0]);
        break;
    // Era
    case 23:
        GlobalData.era = currentSplits[0];
        break;
    }
    lineNum++;
}
}
rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": " + selectedFile + ".txt has been loaded.");
} catch (Exception ex) {
    rtxtSaves.Text += ("\n" + ex.Message);
}
}
}

/// <summary>
/// Save all global data to a new file with user's name
/// </summary>
void saveToFile(bool isAutosave) {
    string selectedFile = "";
    bool goAhead = false;
    // If being called by autosave timer
    if (isAutosave) {
        selectedFile = "./autosave.txt";
        goAhead = true;
    // Being called by button press
    } else {
```

```
// Set dialog filter to text files only
saveDialog.Filter = "Text|*.txt";
// Open saving dialog
DialogResult result = saveDialog.ShowDialog();
// If user clicks OK button
if (result == DialogResult.OK) {
    selectedFile = saveDialog.FileName;
    goAhead = true;
}
if (goAhead) {
    // Stop global timer so that all values are paused at the correct place
    globalTimer.Stop();
    // Now that validation is done, make new file in directory, open it and write to it
    Stream cs = File.Create(selectedFile);
    using (StreamWriter sw = new StreamWriter(cs)) {
        // Write resourcesData to the file
        for (int i = 0; i < 4; i++) {
            sw.WriteLine(GlobalData.resourcesData[i][0] + "#" + GlobalData.resourcesData[i][1] + "#" +
GlobalData.resourcesData[i][2] + "#" + GlobalData.resourcesData[i][3]);
        }

        sw.WriteLine(GlobalData.scienceData);
        sw.WriteLine(GlobalData.totalHousing);
        sw.WriteLine(GlobalData.housingRemaining);

        // Write upgradesData to file
        for (int i = 0; i < 2; i++) {
            sw.WriteLine(GlobalData.upgradesCosts[i][0] + "#" + GlobalData.upgradesCosts[i][1] + "#" +
GlobalData.upgradesCosts[i][2] + "#" + GlobalData.upgradesCosts[i][3]);
        }
        sw.WriteLine(GlobalData.upgradesCosts[2][0] + "#" + GlobalData.upgradesCosts[2][1] + "#" +
GlobalData.upgradesCosts[2][2] + "#" + GlobalData.upgradesCosts[2][3] + "#" + GlobalData.upgradesCosts[2][4]);
        sw.WriteLine(GlobalData.upgradesCosts[3][0] + "#" + GlobalData.upgradesCosts[3][1] + "#" +
GlobalData.upgradesCosts[3][2]);
        sw.WriteLine(GlobalData.upgradesCosts[4][0] + "#" + GlobalData.upgradesCosts[4][1] + "#" +
GlobalData.upgradesCosts[4][2] + "#" + GlobalData.upgradesCosts[4][3]);

        // Write costMultipliers to file
    }
}
```

```
        sw.WriteLine(GlobalData.costMultipliers[0] + "#" + GlobalData.costMultipliers[1] + "#" +
GlobalData.costMultipliers[2] + "#" + GlobalData.costMultipliers[3] + "#" + GlobalData.costMultipliers[4]);

        // Write combatData to file
        for (int i = 0; i < 4; i++) {
            sw.WriteLine(GlobalData.combatData[i][0] + "#" + GlobalData.combatData[i][1]);
        }

        sw.WriteLine(GlobalData.tickCounter);
        sw.WriteLine(GlobalData.curretWar);
        sw.WriteLine(GlobalData.warNumber);
        sw.WriteLine(GlobalData.year);
        sw.WriteLine(GlobalData.actualYear);
        sw.WriteLine(GlobalData.era);
    }
    // Restart the global timer now that the saving is finished
    globalTimer.Start();
    rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": File has been saved to " + selectedFile);

}
}

/// <summary>
/// Deletes user selected file
/// </summary>
void deleteSave() {
    // Set dialog filter to text files only
    openDialogue.Filter = "Text|*.txt";
    // Show dialogue
    DialogResult result = openDialogue.ShowDialog();
    // If user pressed OK button
    if (result == DialogResult.OK) {
        string selectedFile = openDialogue.FileName;
        // Delete file
        File.Delete(selectedFile);
        rtxtSaves.Text += ("\n" + GlobalData.currentTime.ToString() + ": " + selectedFile + " has been deleted
permanently.");
    }
}
}
```

[name]

Candidate No: xxxx

Centre No:xxxxx