

H446 A-Level Computer Science

C03 Programming Project

Hugh Beaumont

Candidate Number: 5066

Hereford Sixth Form

Centre Number: 24175

Documentation of the planning, development and testing
of a town-builder game named ‘Fief’.

WORD COUNT 69,314

PAGE COUNT 326

CONTENTS PAGE

Page Number	Contents
1	Title page
2	Contents page
4	Analysis of the problem Problem identification
8	Stakeholders
8	Research the problem
25	Specify the proposed solution
32	Pre-iterative development note
32	First iteration Discussion with stakeholders
33	Specify the proposed solution (2)
34	Decomposing the problem
35	Describe the solution (1)
40	Describe the solution (2)
42	Approach to testing
44	Describe the solution (3)
47	Coding the solution
60	Testing to inform development
62	Stakeholder review and sign-off
63	Second iteration Discussion with stakeholders
63	Specify the proposed solution (2)
64	Decomposing the problem
65	Describe the solution (1)
67	Describe the solution (2)
69	Approach to testing
72	Describe the solution (3)
90	Coding the solution
112	Testing to inform development
133	Stakeholder review and sign-off

Page Number	Contents
	Third iteration
134	Discussion with stakeholders
134	Specify the proposed solution (2)
135	Decomposing the problem
136	Describe the solution (1)
137	Describe the solution (2)
139	Approach to testing
140	Describe the solution (3)
145	Coding the solution
160	Testing to inform development
163	Stakeholder review and sign-off
	Fourth iteration
164	Discussion with stakeholders
164	Specify the proposed solution (2)
165	Decomposing the problem
166	Describe the solution (1)
172	Describe the solution (2)
173	Approach to testing
173	Describe the solution (3)
180	Coding the solution
210	Testing to inform development
211	Stakeholder review and sign-off
	Evaluation
212	Testing to inform evaluation
231	Evaluation of the solution
231	Success of the solution
242	Describe the final product
256	Maintenance and development
259	Appendix

Hugh Beaumont
Candidate Number: 5066
Centre Number: 24175

Computer Science C03 Programming Project

ANALYSIS OF THE PROBLEM

Problem Identification

This dossier will describe the development of a project to design and construct a town-builder game based in the medieval period. The game will be created in the Visual Studio 2015 IDE suite using the Visual Basic language. The city-builder genre has plenty of potential for a project due to the diverse nature of it.

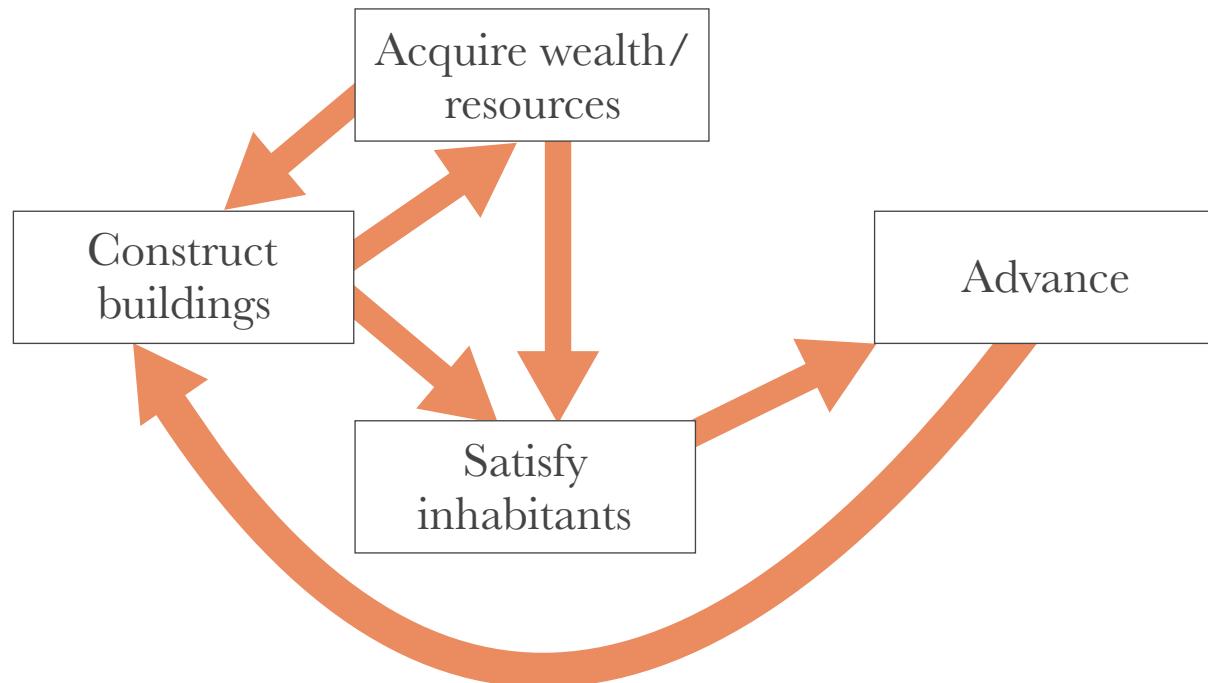
The general goal of these games is often to settle and satisfy inhabitants of the virtual city/town in order to advance and generate virtual revenue to then further advance repeatedly until some sort of goal or limit is reached. Construction of buildings require both resources and funding, the available buildings range from housing to trade guilds to commercial infrastructure. Certain buildings yield benefits and provide a return in investment in various ways, which could include satisfying citizen needs, producing resources or providing public service.

The requirements of progression could include the acquisition of certain resources or virtual wealth; the construction of certain buildings; a technological level; or a population level. There has to be incentive to advance other than for the sake of progression: for example a more advanced town/city could have a larger taxation level or a higher score. There also has to be some strategy to the town design, for example building a mill near water, or a marketplace near housing.

Random events are also a common feature of games of this genre. These events put the user to the test and either punish or reward the user's town depending upon the decision taken. For example a user may decide to build a town with little defences in a bid to save

money; the gamble may or may not pay off depending upon whether or not the city is attacked. Other random events are generated simply to give the user a push forward or stagnate growth for the sake of the game's longevity. Since this is based in the medieval period, this could include events such as plague or famine.

The visual below summarises the game structure in its most trivial form.



The genre of city builder games can potentially be aimed at any age group. Certain mobile games are aimed at younger children whilst others are designed for adults. This game in particular however will be aimed at 13 - 21 year olds, although it could be played by anyone. The game will be aimed at anyone in this age range with interest in the medieval period and enjoy city-builder games. The platform will be computer based and since screen resolution and monitor size varies from person to person, I will need to ensure that the game can cater for all monitor sizes, which could be done by allowing the user to vary the size of the game window. The user should be allowed to pause the game and be able to save the game to load it again at a later date.

The problem must be approached in a computational thought paradigm, and must be amenable to such an approach. The solution will need to be appropriately abstracted to make the problem easier to solve. In a town builder, some information is vital for the formation of

a solution that is realistic enough to provide the user with an experience akin to managing an actual town. Managing a real-life medieval town would have been far more complicated to run than it will be to run in the solution. Thus it is important to identify which parts of the solution will be a simplified version of reality, and what elements of reality will be ignored completely or more accurately represented. The graphics of the solution will be a vastly depleted representation of reality. The graphics will be detailed well enough so that users can understand what the graphic represents and identify individual entities, but will be curtailed at best. The sounds of the town and the ambient sounds will be reduced to reduce complexity of the solution and artificial sounds will be introduced for actions, for example, clicking on a building may produce an unnatural sound which is appropriate for the building (this would help inform the user that they just performed an action and the game is responding). The social complexities of medieval towns that arise from the social structures will be simplified so that the user can better understand how actions can influence effects. For example, the fact that a family may be poor would in real-life medieval towns would have been more likely to steal, but in-game that aspect of reality would be ignored and replaced with another way of punishing the user for having poor families in the town. On the other hand social structures such as the Feudal hierarchy may be accurately represented in the game to make it feel more realistic to the user. In real-life medieval towns collecting wealth and resources would have been complicated tasks, but in the game these elements of reality will likely be abstracted to make it much simpler for the user to progress. For example in real life collecting lumber would require felling specific trees, careful management of the forest and processing of the timber into useful tools or construction material; in the solution it is likely that such a task would be simplified into requiring a single building to collect and process resources such as lumber. One final major example of abstraction in a solution to the problem is the passage of time. In real life a town would take decades to be built up from a hamlet to a village and into a town. Collecting materials and constructing buildings would be significantly time-consuming. In the solution the time will be vastly compressed such that if the passage of years takes minutes or even seconds. Buildings in the solution will likely be constructed instantly and consume the required materials immediately, as opposed to reality where it would take days or weeks to build any sort of building.

At this stage and throughout development of a solution it is important to identify the user inputs and the data that will be required for the solution to function. Town-builder games work by receiving user input and use known data to create an output which affects the

town. For example the user may input using a button the a building they want to construct, and the user would then input where the building should be located. The program should then process the implications of constructing the building. The program will require data on aspects such as each of the buildings to understand how each building affects the town and the requirements of the citizens and how user inputs affect the happiness of the citizens. If there is a settings menu, then the program needs to be able to cope with certain user inputs regarding user preference.

It would be extremely complicated to try to construct a solution including all the features in one single iteration. The problem must be decomposed and tackled element by element. Whilst the mechanics of town builder games are very interconnected, it is possible to break down the problem and solve the individual component parts. For example it would be possible to create a solution to the user interface aspect of the problem independently of creating a solution to the mechanics of constructing buildings or map generation. The order in which these individual elements of the problem are solved should not matter, but to make the progression of solving the problem more understandable, it would be logical to tackle each element in a particular order.

For almost every user action the program will need to make a decision based on data or user input and will require the program to branch or iterate through statements. For example if the user tries to place a building on a specific tile, the program would need to compare the list of valid building tiles to the target tile and either construct the building or output an error if the tile in question was invalid for that building; the program would also have to check if the user has enough materials to construct the building. An example of the program making use of iteration is when the map is being generated: the program will need to iterate through the co-ordinates and create a grid before assigning the tiles landscape types.

Stakeholders

There are three people who are interested in my solution who have elected to become stakeholders. My stakeholders will be Sibyl Beaumont, Alex Dain and James Megenis. I decided on three stakeholders as it will decrease the chances of indecisive answers to questionnaires. Sibyl plays town builder games and is interested in how city builder games generate maps randomly, and so would like the solution to generate maps randomly. The solution is likely to contain a random map generator aspect to it, thus the solution is appropriate for Sibyl. James has an interest in strategy games and history and would like a game that fits his interest. The solution will be appropriate for James as town-builders are a part of the strategy game genre and the game will be set in the medieval period. Alex also plays town builder games and the development of games in the towns builder genre such as the implementation of random events interests him, so this project is ideally suited to him. All my stakeholders, especially James are interested in the historical aspect of the solution and wish the see that reflected in the final product. My stakeholders intend to use the solution for entertainment and are keen to help inform production decisions.

Research the Problem

In order to help inform the direction of the solution to my solution, I will research two successful games related to the problem and study the game mechanics. This will broaden my understanding of how city-builders are played and inspire the basic game mechanics I should include in my solution. In each existing game I will study the features divided into three categories: gameplay (achievement and progress), interface and unique themes in each game.

Anno 1404, Dawn of Discovery

The first solution I will study is the game '*Anno 1404, Dawn of Discovery*', a game developed by Related Design and Blue Byte Software and released in 2009. '*Anno 1404*' is the 4th instalment in the '*Anno*' series.

Gameplay

The game is set on an archipelago (the map is randomly generated in sandbox mode) where an Emperor has granted the user a fief to build a town on and thrive. The user must settle inhabitants and provide them with materials, infrastructure and luxury goods so that they prosper and advance. The inhabitants are taxed to provide the user with capital to expand and fulfil more of the population's needs. There are often computer players who

interact with the player in aggressive and co-operative ways and compete for economic or military supremacy. There are also AIs who are neutral and interact with the player with advice and trade. The player can settle multiple islands to gain control over strategic resources to satisfy their population and progress. The inhabitants have levels of civilisation, and the user goal is to advance to higher levels of civilisation. The four levels of civilisation in order of attainment are: peasants, citizens, patricians and noblemen. Each level requiring more goods and buildings to advance, whilst generating larger amounts of money for taxation. The ultimate goal of each play through, depending on the settings. In the campaign mode there are always objectives to fulfil to complete the level.

In sandbox mode, where the user can play potentially indefinitely; the user can choose goals.

Objectives can include military domination, acquiring a global population size, generating a sum of capital or settling a certain number of noblemen. In-game there are a variety of quests that the computer players offer the

A small Occidental settlement that has reached the Patrician level of civilisation.

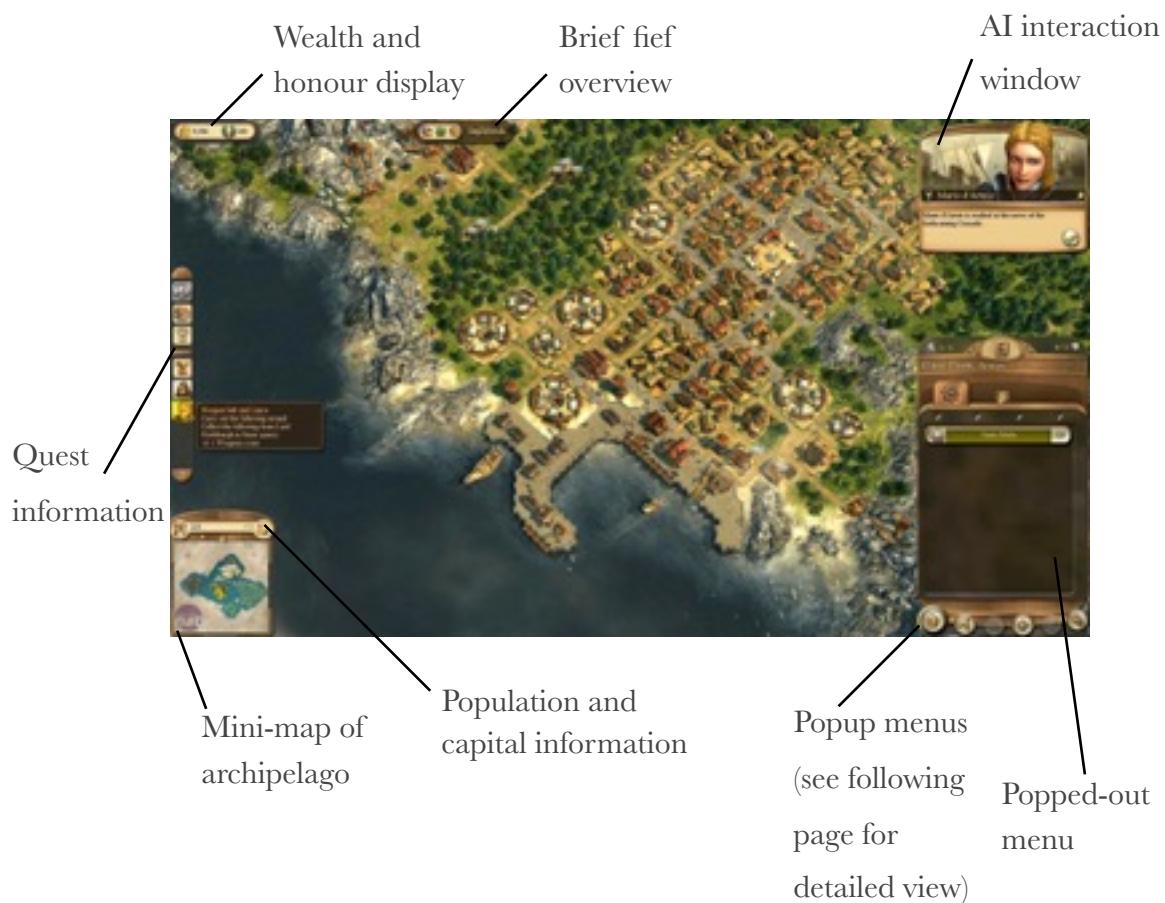


user. These can include privateering, rescue missions or tributing the quest giver certain resources. These can be competitive as they are also offered to other computer players to complete. A completed quest will reward the player with goods, special items and honour, which is a pseudo-resource which can be used to buy special items or goods. There are two main cultures in the game: the Occident and the Orient. The Occident is the main culture which is inspired by medieval/renaissance European Christian cultures such as the Holy Roman Empire, England and the Italian states. The Orient culture is based upon the Islamic Ottoman Empire and one of the neutral AIs is the Grand Vizier of the Sultan, Al Zahir. From him you can gain the ability to settle Nomads who can progress to Envoys in a similar fashion to the four main civilisation levels.

Interface

The interface of this game is very simple and intuitive. There are icons which describe the available menus. The display also provides the user with valuable information such as

stockpile of key resources; quest information; wealth and honour; capital gain through taxation; population size; and a mini-map. As the mouse-pointer hovers over a menu icon, text describes what the menu is, which helps with clarification for new players. The icons for the different buildings and resources are clear and are aesthetically pleasing. There is also a window for AI interaction which displays an animation of the characters talking for authenticity. At the top of the screen is a brief summary of the island: the name and the resources that can be cultivated on the island.



The user is presented with a display that is simple and easy to follow. All the important information is either always viewable or is only one click away. This is the kind of presentation which is suitable to the city-builder genre, as there is often many things the user has to consider at the same time.



The large house icon on the far left is the button for the buildings menu. This displays all the buildings available to the player. The buildings are categorised by the civilisation level they apply to. For example on the right are the buildings available for the second tier of civilisation (citizens). Each icon displays a picture which illustrates what building it is. Hovering over the icon displays the building name and resources required for construction. This makes gameplay very informative and easy to use for new players.

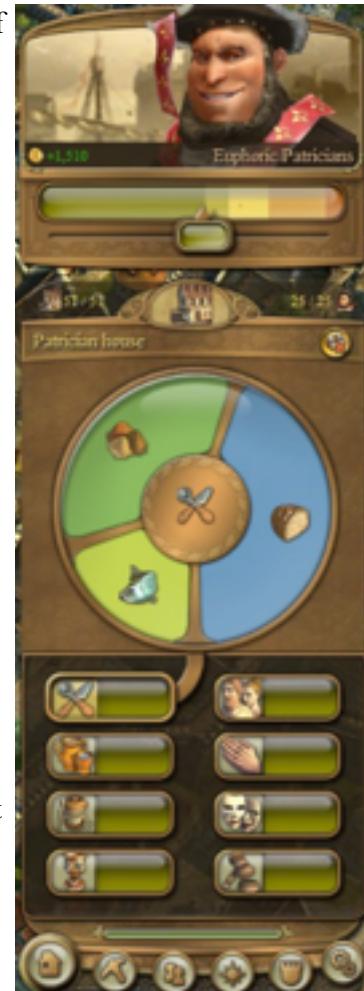
The pickaxe icon second from the left is simply a demolish tool which is used to remove unwanted structures and roads. This is an appropriate place for the tool as it would be used commonly and is therefore in an easy to find location.



The compass icon just right of centre is a trade overview. This allows the player to create and manage trade routes easily. When the mouse pointer hovers over a trade route in the list the route is shown on the map. The final important icon is the cog button which reveals an advanced menu which gives access to more management options such as a bailiwick registry.



When the player clicks on a house, it shows the type of inhabitant and their needs. The icons depict what the inhabitants are demanding and a bar shows the level of satisfaction by each requirement. Clicking on the demand will give an expanded and more detailed pie chart of the constituent needs of the demand. For example the pie chart depicts the food demand of this patrician, who requires fish, bread and spices. In this menu there is also the option to change the level of taxation of this type of inhabitant. Higher levels of tax is beneficial to the player, but too high tax (i.e. in the orange and red zone) will have detrimental effects on the town (riots for example). The use of colours in this menu is very useful and makes the user experience very intuitive. One will know off hand that too high taxation on the patrician will have negative effects from the colour of the tax boundary.



Unique Themes

This game has a strong focus on capitalism. Trade is a central part of the game to generate extra revenue from excess product. There is an emphasis on investing for a return in profit, for example investing in settling a new island to secure more resources and connecting the island to your trade network in return for a happier settlement and a greater return in capital.

Anno 1404 also has a strong emphasis on computer player interaction and random events. Random events can range from relatively trivial (a plague or fire breaking out in a household) to pivotal (invasion from computer players or corsairs). The random events contribute hugely to the overall reusability of the game, as no two games are similar.

Another aspect of the game is the combat mechanics. The game encourages the construction of a fleet to defend trade vessels and to assert influence on other players. It is possible to send an army of musket men to conquer an adversary to gain monopolies on strategic resources, or just to simply remove competition. The frequent corsair raids and privateer missions given by AI players forces the player into investing in a military.

This added to the overall enjoyment of the game massively and made the game more riskier if one decided to opt out of a military to conserve money.

Overall from this game I have learned that a clear interface is vital to the success of a solution to this sort of problem. The user must find it convenient to manage a town and have important information displayed in a fashion that does not overwhelm the user, else the player will be put off. The inclusion of random events and varied gameplay is also vital to keep the user interested. If the gameplay is too linear then it is unlikely the user will return to the game again after first use.

Age of Empires II: The Age of Kings

The second solution I will study is the 1999 game '*Age of Empires II: Age of Kings*' developed by Ensemble Studios. This game is not a town-builder, but a Real Time Strategy game, but there are some mechanics in this game which are appropriate to consider in relation to my solution.

Gameplay

Each game the user starts with a main Town Centre building, a scout and 3 villagers. The aim of the game is to build up a settlement and defeat your opponent(s) before they defeat you. Villagers are used to collect resources and construct buildings which are used for various purposes including training an army or allowing research of technology.

One main component of the game is the Age system. There are four Ages in the game: the Dark Age, the Feudal Age, the Castle Age and the Imperial Age. The player is incentivised to advance to higher ages as the higher ages provide better technology, a larger variety of buildings and a set of more powerful military units. The player must make balanced decisions throughout the game whether to advanced and postpone boosting military size to gain a power boost from

A Frankish town in the Feudal Age



technological superiority over the opponent; or remain in the current age to ensure that the town is properly equipped to fend off attack or so that the player may take the offensive earlier. This important aspect of making balanced decisions and taking risks because of or in spite of potential circumstances is an aspect that must be included in order to make any solution successful. The Age element of this game is similar to the Levels of Civilisation progression system in '*Anno 1404*' and it is clear that the incorporation of this element in a solution will improve its appeal. This is because it gives the user clear goals and rewards the player for advancing, but can also punish the player for advancing in inappropriate circumstances.

'Age of Empires II: Age of Kings' also has a very expansive technology tree. Technologies are researched from almost every building in the game, but most notably from the blacksmith and the university buildings which are dedicated to research. The technologies either improve units and buildings or unlock new units and other buildings. The technology feature in this game adds a huge amount of depth to the strategy of the game. The inclusion of a technology system would be hugely beneficial to my solution.

A key dynamic to this game is the wide range of civilisations available. The civilisations vary from Western European cultures such as the Spanish, Celts and the Britons; through

Summary of Aztec civilisation



A Castle Age Briton university with the available technology icons displayed in the bottom left.



Mesopotamian civilisations: Byzantines, Turks and Saracens; to Mesoamerican empires: Aztecs, Mayans, Incas. A full list is shown on the following page. Each civilisations have their own bonuses; strengths and weaknesses; and can all field at least one unique unit. For example, as seen left the Aztecs can create military units 15% faster, and can field the jaguar warrior unique unit, however the aztecs cannot create cavalry, as horses did not exist in America until the Spanish arrived.

Each civilisation also has its own architecture sets, unit textures and unit accents. By including a huge variety of civilisations into the game, every single game is different as players try play according to their own civilisation's strengths and exploiting their opponents weaknesses. There is therefore almost no meta-game in '*Age of Empires II: Age of Kings*', which makes the game extremely repayable, hence its great longevity, and after nearly 2 decades still has a huge community base. It would therefore be very sensible to allow this successful game to help inform the final solution.

Interface

The game's interface lends itself well to intuitive gameplay. The icons for units and buildings are very descriptive and the User Interface is attractively minimised and simple.

Resource levels: wood, food, gold and stone icons with their respective stockpile size shown clearly.

Text box showing current Age.

Various menus including map information, tech tree, team chat, diplomacy and options



Research window for blacksmith technologies

Decorative bar with simple medieval design. Each design is unique to a civilisation.

Mini-map with various display options



Building icon and name with health point information

Player civilisation and name



Unlike '*Anno 1404*', the menus are not very dynamic. There are no pop-up menus, and there is designated space for options and menus. This is very suitable to real-time strategy games as often the player is under pressure and thus does not want to be distracted by menu windows. In real-time strategy, the user will only be concerned with one task at one point, and such only one menu is needed. However in town builder games, the user may want to have multiple windows open at any one point as they consider various factors as they make a decision. As my solution is purely a town builder, pop-up menus are much more appropriate as there will be less pressure and more options should be available to the user at the same time.

Icons used are used to clearly describe to the user what each action does. In the building menu the icons clearly describe what each building does. For example below the icon depicting an anchor (in the red box) is very obviously a dock or harbour of some description, the icon showing a drawn bow (in the green box) is clearly something to do with archers.



By using descriptive icons the user has a much more intuitive experience and reduces any learning curve the game might otherwise have. I must include clear and descriptive icons in the final solution.

Unique Themes

Warfare is the main focus of this game, which attracts many players. Throughout all four ages the user has military units available, and the very best players will attack in every age. I feel it is in the interest of the solution's success to include some aspect including military, but it is something I will have to consult my stakeholders about. In each game of '*Age of Empires II: Age of Kings*', the player will eventually win or lose in a black or white fashion. The main way of victory is conquest.

A large battle taking place.



There are however alternative ways of winning including capturing relics from around the map, building a wonder, or in a specific game type by killing the enemy king. There are various ways of winning in the campaign mode, but the campaign victory conditions are very specific.

‘Age of Empires II: Age of Kings’ bases many of the campaigns and civilisations on historical facts and stereotypes. For example the Britons are given the best archers in the game due to the historical use of the English/Welsh longbows, and the campaigns follow historical figures such as Barbarossa and Saladin.

After examining this game, I have learned that elements such as the ageing progression system and the large tech tree give incentive to advance and gives greater variety to each play-through. The use of a variety of civilisations lends to a strong solution as it makes each play-through unique and encourages different play styles. I feel that including these features in my solution will produce a greater user satisfaction and will make them more likely to replay the game. I will consult my stakeholders if they would like to see these components such as ageing, technologies and several factions/civilisations included in the solution. Age of Empires II: Age of Kings also has a clear and minimised user interface, which helps reduce the learning curve and makes the game less overwhelming to play.

After conducting research into existing solutions and demonstrating these games to my stakeholders I will now produce a questionnaire for my stakeholders so I can take inspiration from the features of the existing solutions that my stakeholders favoured and dismiss features that my stakeholders disliked. This will help decide the initial direction of the project. The questionnaire questions were as follows:

Gameplay

- 1). How long should a game take to play?
 - a) Less than 1 hour
 - b) 1-2 hours
 - c) Greater than 2 hours
- 2). What sort of inventory items would you prefer?
 - a) Resources
 - b) Wealth (skip to Q4)
 - c) Both

3). How many unique resources (including wealth) are you expecting to see?

- a) 1-5
- b) 6-10
- c) 10<

4). Should there be a scoring mechanism?

- a) Yes
- b) No

5). Should populations have a progression (i.e. should they be able to advance)

- a) Yes
- b) No
- c) Yes, but proportionately

6). Should there be an ultimate goal? If yes, please select which.

- a) No
- b) Yes, Score objective
- c) Yes, Population objective
- d) Yes, Wealth objective
- e) Yes, Progression objective (reaching a high level of civilisation)

7) Should there be random events?

- a) Yes
- b) No

8) Would you like any military aspect at all?

- a) Yes
- b) No

9) Would you like sound effects/background music?

- a) Yes
- b) No

10) Would you like the game to incorporate historically accurate features?

- a) Yes
- b) No

11) Would you like there to be an option to choose different civilisations/factions?

- a) Yes
- b) No

Graphics and visuals

1). Would you like the buildings to be diamond shaped or square shaped? (i.e would you prefer seeing the buildings head on or the corner?)

- a) Diamond
- b) Square

2). What kind of terrain are you expecting to see?

- a) Grassland/Forests (similar to rural England)
- b) Desert/Oasis (similar to Mesopotamia/Arabia)
- c) Snowy/Tundra (similar to Scandinavia)

3). Should the graphics look realistic or pixel graphics?

- a) Realistic
- b) Pixel

4). Would you like animations (i.e smoke from chimneys, running rivers, etc)

- a) Yes
- b) No

5). How large should each tile be? (keeping in mind buildings may take up different number of tiles)

a) 20x20 pixels



b) 30x30 pixels



c) 40x40 pixels



d) 50x50 pixels



e) 60x60 pixels



6). What percentage of the playable map should you be able to see at once?

- a) <20%
- b) 20-39%
- c) 40-59%
- d) 60-79%
- e) 80-100%

7). Would you prefer a minimalist interface of popup menus or full display?

- a) Minimalist
- b) Full display

Content miscellaneous

1). Would you like to be able to manage professions? (e.g Reeve, blacksmith, etc.)

- a) Yes
- b) No
- c) Don't mind

2). How many unique buildings are you expecting?

- a) At least 5
- b) At least 10
- c) More than 15

3). Finally please tick the boxes of this table if you consider these buildings essential, preferable or not preferable.

Building	Essential	Preferable	Not preferable
House			
Farm			
Mill			
Church			
Manor house			
Market			
Blacksmith			
Tower			
Wall			
Mine			
Pasture			
Public house			
Healer house			
Bakery			
Butcher			
Barracks			
Town Square			
Guard quarters			
Guild houses			

Questionnaire results

The following table indicates feedback questionnaire responses.

Question	Option selected				
	A	B	C	D	E
Gameplay					
1	1	2			
2			3		
3			3		
4	3				
5	1		2		
6		2			1
7	3				
8	3				
9	3				
10	3				
11	3				
Graphics					
1	1	2			
2	3				
3	1	2			
4	3				
5		1	2		
6			1		2
7	3				
Content Misc.					
1			3		
2		2	1		

Final question response on next page.

3).

Building	Essential	Preferable	Not preferable
House	3		
Farm	3		
Mill	3		
Church	2	1	
Manor house	1	2	
Market	3		
Blacksmith	2	1	
Tower	1	2	
Wall	1	1	1
Mine	3		
Pasture	2	1	
Public house	3		
Healer house	2	1	
Bakery	1	2	
Butcher	1	2	
Barracks	3		
Town Square	3		
Guard quarters	2	1	
Guild houses	3		

Now that I have collected preferences from my stakeholders it is much easier for me to identify the essential features of my solution and discuss some of the limitations that I will face in the creation of the project.

It is clear that my stakeholders are not interested in a game that will drag out for longer than 2 hours. I need to ensure the game plays quickly enough to finish within 2 hours. I will have to implement a method of saving and loading so that my users can stop playing at any time and come back later. It is also important that the games aren't too rapid. The majority of my stakeholders preferred games that lasted longer than one hour. Features such as random events could increase the complexity of the game and make the games longer.

My stakeholders are also very keen to have a large variety of unique resources, with all of them answering that they would prefer over 10 different unique resources, as well as a wealth factor and a scoring mechanism.

It is also clear that the civilisation needs to demonstrate evolution. All my stakeholders wanted to see their towns advancing, with a majority preferring a more realistic proportional advance of civilisation (i.e a certain percentage of the population can advance).

All my stakeholders were keen that there should be an ultimate goal. Whilst there is a possibility for more than one objective it seems that a score objective is the most preferred option. This makes a scoring mechanism very essential to the solution.

Random events will be an essential aspect of the solution as it will help diversify the gameplay and all my stakeholders were eager to see this as a feature. A military aspect to the game appears to also be very popular, and my stakeholders suggested that the random events and military aspects could be closely linked.

Sounds effects are also a very desired feature of the solution, thus I should dedicate time and resource into ensuring the game has appropriate sound effects to enhance the gameplay.

All of my stakeholders said at the start that they were keen on history, so it is unsurprising to see that they were unanimous that the game should include historically accurate features, this will become integral to the solution.

My stakeholders want to see a variety of buildings available to them, with a majority requesting at least 10 individual buildings. I will try to implement as many of the buildings that my stakeholders considered essential as time will allow.

Limitations

The Visual Basic language in the 2015 Visual Studio IDE suite is not suitable for advanced graphics such as animations, as Visual Studio does not include support for OpenMP, which optimises run-time behaviour. In previous projects using this IDE I have found that images of picture boxes and bitmaps take time to load, so attempts at animations would cause flashing, which is wholly undesirable. Thus I will not be able to implement high level graphics. It is likely I will implement multiple civilisations for the user to play the game as, which will change the user's approach to the game. As time is limited, I will not be able to implement a large variety of civilisations, else I will not have the time to implement them all. A later version could include a larger variety of available civilisations. Ideally each civilisation would have its own texture set (i.e. individual textures for buildings and landscape) however as

graphics is very time consuming, there will be no time to implement this feature, thus all civilisations will have the same in-game texture set. My stakeholders are keen on the realism of the solution, but as I discussed when I covered abstraction in problem identification, certain events will be abstracted to an extreme level. This is to reduce the complexity of the solution to a standard which is realistic to code. This abstraction will limit the realism of the solution, which my stakeholders, having discussed this problem with them, have agreed to compromise on this. The number of possible buildings and levels of civilisations will have to be severely limited from the number in real life, as it would not be possible to implement in the proposed time period. Perhaps in later versions, more buildings and levels of civilisations could be added. My stakeholders are keen that I include background music and sound effects into the game. In Visual Studio there are two main ways to implement multiple sounds into a solution: scheduling and multithreading, the former only allowing one sound at once and the latter allowing multiple sounds to be played simultaneously. Implementing sounds using multithreading is incredibly complex, and time would not allow for it. Thus I will use scheduling. This means I will only be limited to one sound at a time. I will need to write a scheduling procedure else I will not be able to implement music as sounds effects will take priority. The general theme is that there is a limitation to every aspect because of the available time. I would like to fully implement every aspect of the solution accurate to reality as my stakeholders would prefer, but I will have to compromise and sacrifice some verisimilitude in exchange for a greater variety of aspects to the solution which will improve the user experience and the replay-ability of the solution.

Specify The Proposed Solution

The solution will be judged by my stakeholders according to success criteria that are listed below in a table:

Success Criteria	Description	Justification
SC1	The user must be able to play as three different civilisations.	As discussed in the research, a variety of civilisations force the user into playing differently each game play-through. This will increase replay-ability of the solution.
SC2	Each civilisation must have different bonuses/disadvantages, buildings and technologies.	This will directly affect the strategy of the play through. Different civilisation attributes will force the user into playing differently and according to the strengths of the chosen civilisation.

Success Criteria	Description	Justification
SC3	The user must be able to choose the name of the town.	This will make the solution more personalised to each user.
SC4	The map on which the user will build the town must be randomly generated.	One of my stakeholders is particularly keen on learning how developers make maps that randomly generate. A variety of maps to play on will increase the replay-ability of the game.
SC5	There must be a help menu which gives the user all the information necessary to play the game which must be accessible from the main menu and in-game.	The help menu exists to teach the user how to play the game. Without it the user will not be able to play the game properly. The user must have access to it from in-game as the user may want to quickly look something up without having to exit the game.
SC6	The user must be able to save at least one game to be reloaded later.	As the game will likely take 1 - 2 hours, the user may not be able to complete a play through in one sitting. Therefore in order for the game to be fully playable, the user must be able to save the game to return to it later.
SC7	The initial in-game screen must display the most fundamental information about the town.	This gives the user a display which provides them with all the most basic information about the town. This will allow the user to make decisions easier.
SC8	The in-game screen must include buttons to open up an expanded menu with options related to town management.	My stakeholders made clear that they preferred minimalist user interface utilising pop-up menus. This produces a straightforward user interface which is easier for new users to use.
SC9	There must be a specific menu available on the in-game screen which allows the user to save the game and return to the main menu.	The user may decide that the map that has been generated is not desirable and thus wishes to exit or the user may wish to save the game to return to it later. Therefore these options must be made available to the user at the in-game screen.
SC10	Navigation between menus must be flexible and at any point the user must be able to return to any other menu easily.	The user may reconsider certain settings (e.g. civilisation choice) and may wish to return to previous menus to change settings.
SC11	User must have access to at least 10 unique buildings which can be placed on the map in appropriate locations.	The cornerstone of town-builder games is the ability to build structures wherever the user wants. In the interests of realism, certain buildings will not be allowed in some locations (e.g. a harbour on a mountainside). The minimum of 10 was chosen as a majority of my stakeholders desire at least 10 unique buildings.
SC12	Each level of civilisation must provide the user with a reward/benefit upon being reached.	This gives progression through the levels of civilisation appeal as the user will be closer to winning the game.

Success Criteria	Description	Justification
SC13	There must be a scoring mechanism which calculates the current score based upon different aspects of the town. The score should be used as a victory condition.	This will provide a standard for a victory condition based upon score and will provide to the user a simple measure of progress and success.
SC14	The solution should generate random events at random time periods.	This will make the game more unpredictable and will force the user into making decisions where to invest to safeguard against negative random events.

Each of these success criteria have requirements which must be satisfied in order for each success criteria to be fulfilled. Each requirement is listed in the table below with the success criteria it relates to, a description of the requirement and justification.

All requirements of each success criterion are the product of discussion with my stakeholders.

Requirement	Success Criteria	Description	Justification
SCR1.1	SC1	The user must have the option to play as the Seljuk Turks.	My stakeholders wanted a civilisation from Arabia/Mesopotamia so I chose the Seljuk Turks.
SCR1.2	SC1	The user must have the option to play as the Kingdom of England.	My stakeholders wanted a European civilisation so the Kingdom of England was chosen.
SCR1.3	SC1	The user must have the option to play as the Yuan Dynasty	My stakeholders wanted an Asian civilisation as an option so I picked the most well known Chinese Empire.
SCR2.1	SC2	There must be at least one unique building for each civilisation. See note 1.	This will emphasise diversity in the civilisations and vary the gameplay.
SCR2.2	SC2	The technology tree should vary for each civilisation (i.e. availability of standard technologies and unique technologies). See note 1.	See SCR2.1.
SCR2.3	SC2	Each civilisation should have at least one bonus and one disadvantage. See note 1.	See SCR2.1.
SCR3.1	SC3	The solution must contain a menu before the start of the game where the user can input the name of the town.	This will allow the user to personalise the name of the town.

Requirement	Success Criteria	Description	Justification
SCR3.2	SC3	The in-game screen must include a display for the name of the town.	This will implement the input described in SCR3.1 to make the solution more personalised to the user.
SCR4.1	SC4	The solution must randomly generate a river feature.	For the sake of realism, and given that most towns are settled near rivers, it makes sense to include a river in the map generation.
SCR4.2	SC4	The solution must randomly generate a water feature (i.e. a lake or the sea).	As a river will be present, my stakeholders suggested I include a lake or a sea which the river runs into, rather than running to the bottom of the screen.
SCR4.3	SC4	The solution must randomly generate mountain ranges.	My stakeholders requested that I include mines as one of the building type, thus I require mountains where the mines will be allowed to be built.
SCR4.4	SC4	The solution must randomly generate forests.	A basic material for buildings will be wood, thus the user requires somewhere to get the wood from.
SCR4.5	SC4	The solution must randomly generate hills.	My stakeholders suggested that there should be terrain where certain buildings get bonuses. A good example is a tower on hills, thus I decided to include hills into the map generation.
SCR4.6	SC4	The solution must randomly generate flora.	My stakeholders want to see the plains terrain varied so the graphics are not bland.
SCR4.7	SC4	The solution must generate plains in the remaining tiles.	All remaining land should be occupied by plains which the majority of buildings will be allowed to be placed.
SCR5.1	SC5	Each major aspect of the game must have instructions in the help menu.	This will allow the user to play the game without any external aid.
SCR5.2	SC5	The help menu must be accessible from the main menu.	This will allow the user to have access to a tutorial before playing the game.
SCR5.3	SC5	The help menu must be accessible from in-game.	This will allow the user to check a guide of the game whilst playing easily.
SCR6.1	SC6	There must be an option to save the game whilst in-game which will write all necessary information to a file.	This will allow the user to save games to be returned to later.
SCR6.2	SC6	There must be an option to load a saved game from the main menu.	This will allow the user to return to a saved game later on.

Requirement	Success Criteria	Description	Justification
SCR6.3	SC6	The user must have the option to delete saved games from the main menu.	This will allow the user to save another game later on.
SCR7.1	SC7	The in-game screen must display population count.	This is fundamental information which the user should have access to at all times.
SCR7.2	SC7	The in-game screen must display wealth and income.	This will tell the user what buildings and other investments they can afford, and give the user an idea as to whether their town is growing or is in recession.
SCR7.3	SC7	The in-game screen must display the date of the game.	The date will give the user an appreciation of in-game time to help model strategies.
SCR8.1	SC8	There must be a button to access a menu for building construction options.	This will allow the user to access a menu where they can select buildings to place.
SCR8.2	SC8	There must be a button to access a menu for town management.	This will allow the user to access a menu where they can implement decisions about management of the town (e.g. raise or lower taxes).
SCR8.3	SC8	There must be a button to access a menu for detailed town information.	This will allow the user to access a menu where they can see information to make decisions about town management.
SCR9.1	SC9	There must be a button to access a menu where the user can save the game or return to the main menu.	This will give the user access to a menu where the game can be saved or the user can return to the main menu.
SCR9.2	SC9	There must be a button which commences saving the game.	This will allow the user to save a game to return to later.
SCR9.3	SC9	There must be a button which returns to user to the main menu.	This will allow the user to exit the game for any reason.
SCR10.1	SC10	All menus before the in-game menu must have a path back to the main menu.	This will allow the user to restart the setup of the game at any point during setup.
SCR11.1	SC11	The user must be able to place at least 10 unique buildings.	My stakeholders voted to have at least 10 unique buildings. This will make the gameplay more diverse and give it more depth.
SCR11.2	SC11	Each building must yield a different bonus. See note 1.	This will give the user incentive to build each building.
SCR11.3	SC11	Certain buildings must not be allowed to be placed in certain locations. See note 1.	In the interests of realism, this must be fulfilled. An example of this is preventing a harbour being on a mountain.

Requirement	Success Criteria	Description	Justification
SCR11.4	SC11	Inverse of SC11.3: Certain buildings must be allowed to be placed in certain locations. See note 1.	See SCR11.3. An example of this is allowing a mine to be built on mountains.
SCR11.5	SC11	If a building is related to technology research, there must be a button to bring up the technology menu related to that building.	This will allow the user to research technology to yield bonuses from the technologies.
SCR12.1	SC12	There must be 3 levels of civilisation: peasant, freemen and yeomen.	This will give the user meaningful feedback on progress.
SCR12.2	SC12	Certain buildings must only be available once the user has met the criteria to advance to the next level of civilisation.	This will give the user incentive to advance.
SCR12.3	SC12	Upon reaching the next level of civilisation the user should be rewarded. See note 1.	See SCR12.2.
SCR13.1	SC13	The score must take population into account. See note 1.	This will give the user a higher score depending on size of the town. Gives incentive to increase the size of the town.
SCR13.2	SC13	The score must take number of buildings of each type into account. See note 1.	See SCR13.1.
SCR13.3	SC13	The score must take wealth and income into account. See note 1.	This will give the user more incentive to amass wealth.
SCR13.4	SC13	The score must take level of civilisation into account. See note 1.	Gives further incentive to advancing civilisation.
SCR13.5	SC13	The score should take decisions made about random events into account. See note 1.	This will give significance to random events.
SCR13.6	SC13	There should be a victory requirement based upon score. See note 1.	This will give the user incentive to get higher scores. This will also give the user an idea of how successful their town is.
SCR14.1	SC14	Each random event should take place at random time intervals. See note 1.	Makes random events unpredictable, thus the user cannot directly prepare for random events, making gameplay more challenging.
SCR14.2	SC14	The random events should either produce a positive or negative effect. See note 1.	Gives the random events more significance.
SCR14.3	SC14	Some random effects must give the user an option on how to respond, the decision should then have some meaningful effect on the town, positive or negative.	Makes the solution interact with the user, making the solution more immersible.

Note 1: Details of this requirement cannot be specified at this time as the effect on gameplay cannot be accurately measured until a prototype solution is beta-tested.

Fulfilling these requirements will satisfy the success criteria which will determine whether the solution is a success or a failure.

Hardware Requirements

As I am using Visual Studio 2015 for this project the users will need the following to play the game:

- 1.6GHz or faster processor
- 1 GB of RAM/1.5GB of RAM on virtual machine
- 5400 RPM hard disk drive with 4GB of space.
- DirectX 9-capable video card (1024 x 768 or higher resolution)
- A keyboard for hotkeys and entering into text fields (not essential)
- A mouse
- A monitor

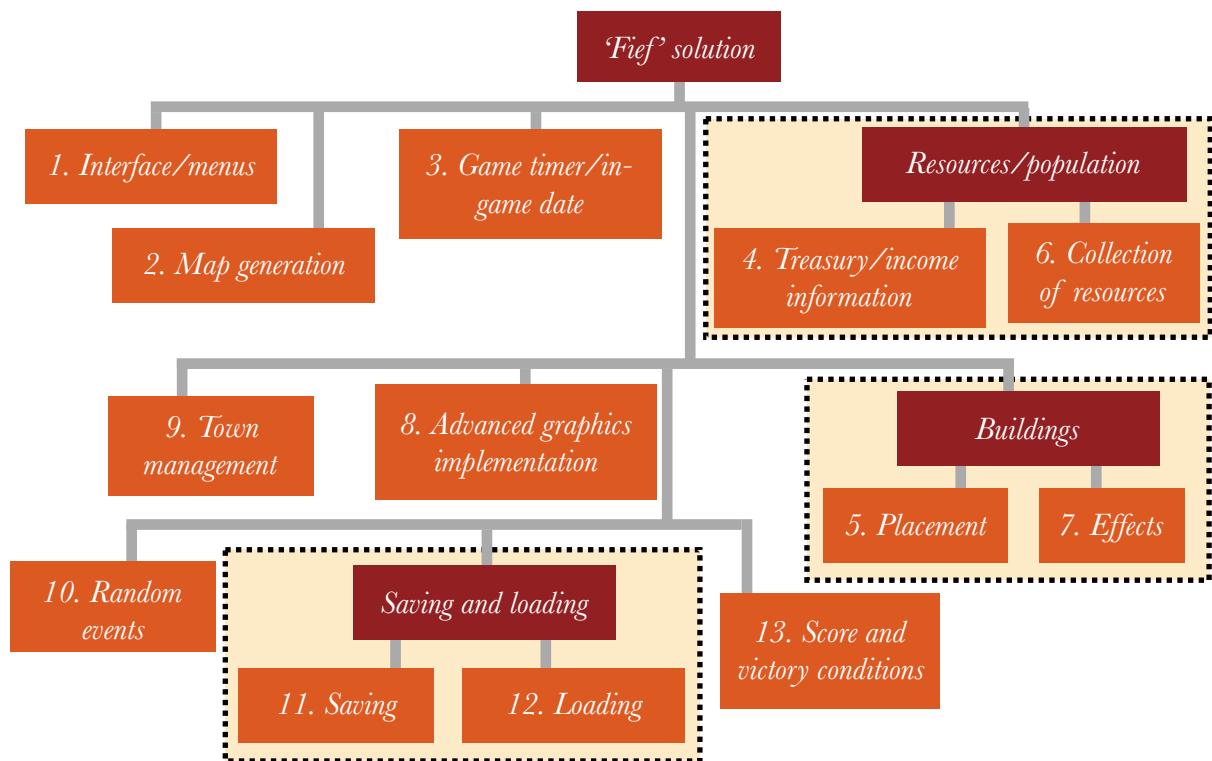
Software Requirements

- One of the following Operating Systems:
 - Windows Server 2008 R2 SP1
 - Windows Server 2012
 - Windows Server 2012 R2
 - Windows 7 SP 1
 - Windows 8
 - Windows 8.1
 - Windows 10

END OF ANALYSIS OF THE PROBLEM

PRE-ITERATIVE DEVELOPMENT NOTE

Below is a diagram of how the problem will be broken down into individual iterations. In each iteration an extract of this diagram will be shown detailing each iteration individually and the planned order of implementation. The number of iterations described here by the diagram is not a definite figure. The number may increase according to changing user requirements. It is very unlikely the number of iterations described here will decrease. Each leaf of the tree notates a single iteration.



FIRST ITERATION

Discussion with Stakeholders

For the first iteration I will be implementing the user interface and menus. This includes the main menu, in-game user interface and options. I decided with my stakeholders what the main menu should include, what should be displayed in-game, and what in-game menus are available. From the questionnaire in my research, I found that my users preferred using pop-up menus to manage the town. My stakeholders also agreed with me on a name for the game which will be displayed in the title screen: '*Fief*', which is what a landholding is called in Feudalism.

Specify the problem: requirements of menus and game interface

Specific requirements of iteration

- R1.1.** Clicking start game button must load the civilisations screen where the user can select a civilisation.
- R1.2.** Clicking the start button on the civilisation menu must load a menu to enter the Town Name. There must be a button to start the game after entering town name.
- R1.3.** There must be in-game displays which always show population of town; amount of wealth; magnitude of income; name of town and the in-game date.
- R1.4.** There must be buttons to access menus including: building menu; treasury menu; town management menu; start menu and help menu.
- R1.5.** Menu buttons must open pop-up menus which the user can interact with using more buttons.
- R1.6.** Stakeholders must be content with the design of the menus and interface.
- R1.7.** Main menu must have the options to start the game and to manage saved games.
- RO1.1.** My stakeholders suggested I try make the main menu image change each time it is loaded, this will be considered as an optional element of this iteration as it is not vital to the solution.

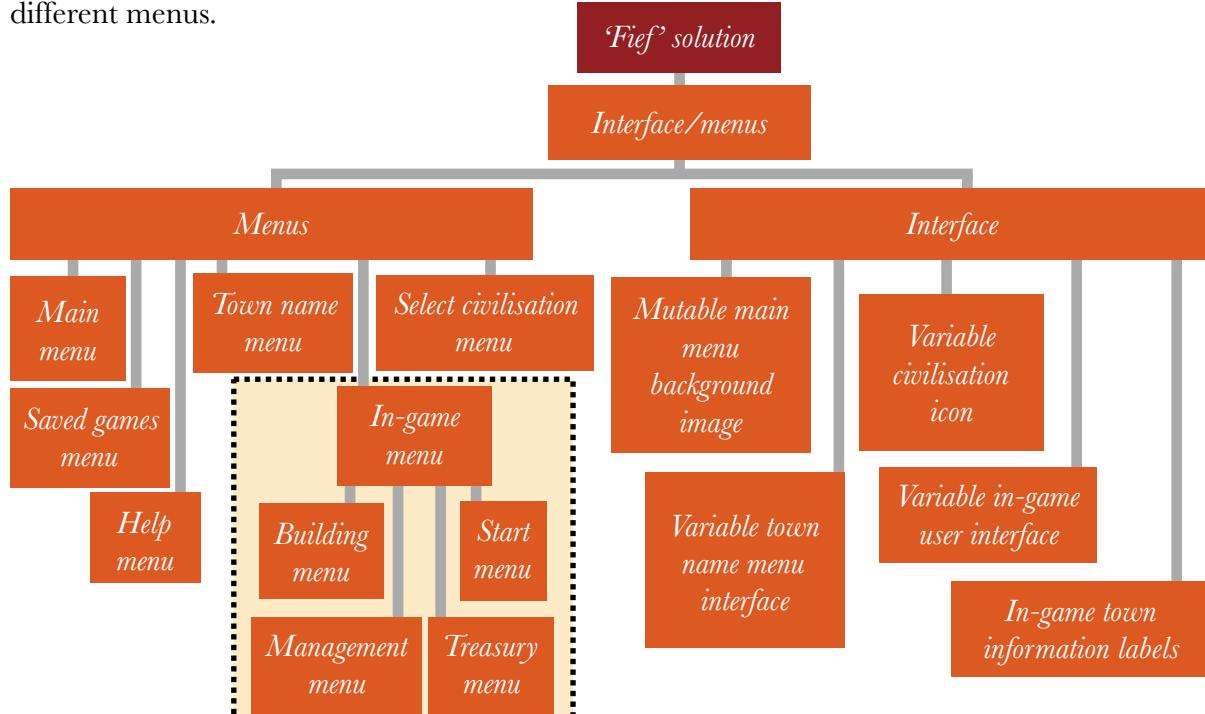
Success criteria and requirements the iteration addresses:

Success Criteria	Requirement	How success criteria requirements are addressed
SC1	SCR1.1	A button on the select civilisation menu will allow the user to select the Seljuk Turks.
SC1	SCR1.2	A button on the select civilisation menu will allow the user to select the Kingdom of England.
SC1	SCR1.3	A button on the select civilisation menu will allow the user to select the Yuan Dynasty.
SC3	SCR3.1	A menu will exist where the user will be asked to input the name of the town.
SC3	SCR3.2	There will be a label in the in-game screen which will display the town name.
SC5	SCR5.2	There will be a button on the main menu to access the help menu.
SC5	SCR5.3	There will be a button on the in-game display to access the help menu.
SC7	SCR7.1	The in-game screen will include a label to display the population level.
SC7	SCR7.2	The in-game screen will include labels to display wealth and income.
SC7	SCR7.3	The in-game screen will include a label for in-game date.

Success Criteria	Requirement	How success criteria requirements are addressed
SC8	SCR8.1	A button will be added to the in-game screen to access building construction options.
SC8	SCR8.2	A button will be added to the in-game screen to access town management options.
SC8	SCR8.3	A button will be added to the in-game screen to access town information.
SC9	SCR9.1	There will be a button on the in-game to access a menu to save the game and/or return to the main menu.
SC9	SCR9.2	A button will be included in the menu described directly above to save the game.
SC9	SCR9.3	A button will be included in the menu described two rows above to return to the main menu.
SC10	SCR10.1	Each menu will have a button to return to the previous menu which provides a link for every menu to the main menu.

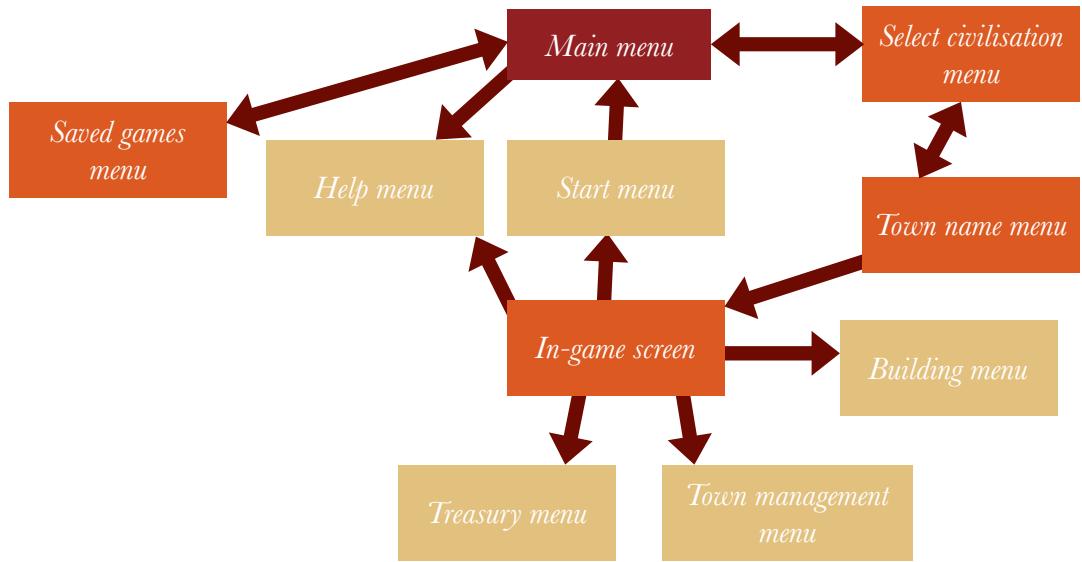
Decomposing the Problem

The diagram below elucidates the problem presented in this iteration by showing the components of this part of the solution to the problem. This iteration was broken down using stepwise refinement. The iteration was first broken down into two general groups: menus and interface because the basic functions of each menu should be tackled separately from the detailed interface of each menu. The menus section has been further subdivided into individual menus as each menu's function is unique and thus each should be developed individually. The pale yellow box denotes menus related to the in-game screen. The interface section has been broken into five further problems as each problem is unique and relate to different menus.



Describe the solution (Usability features)

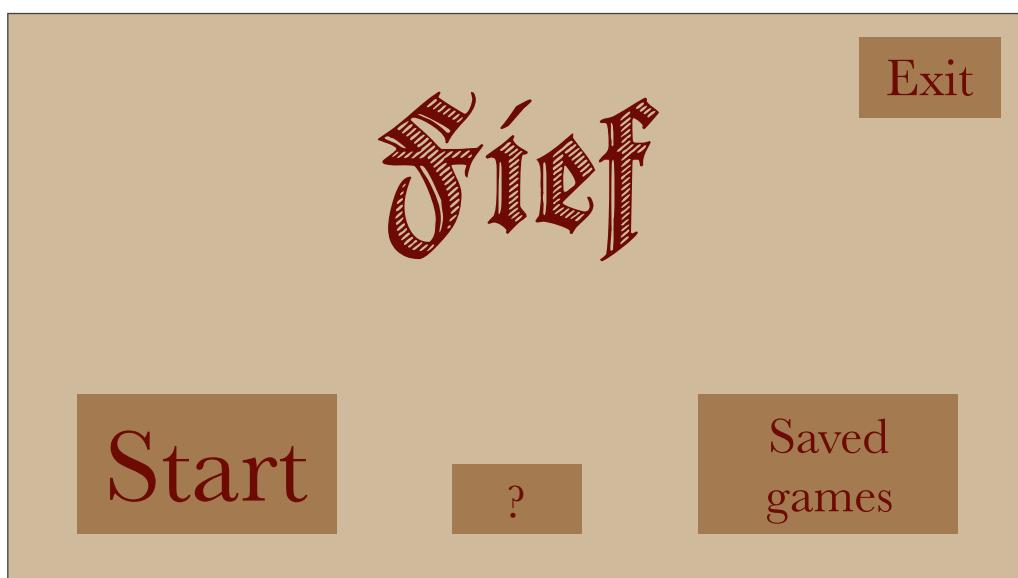
Each menu will be able to navigate to at least two other menus. A diagram of the menu navigation is shown below. Main menu is shown in red as all other menus must have a path back to the main menu as specified by SCR10.1. Menus shown in pale brown are pop-up menus, and do not close any other menus upon being opened.



I designed a series of menus and agreed on the layout with my stakeholders.

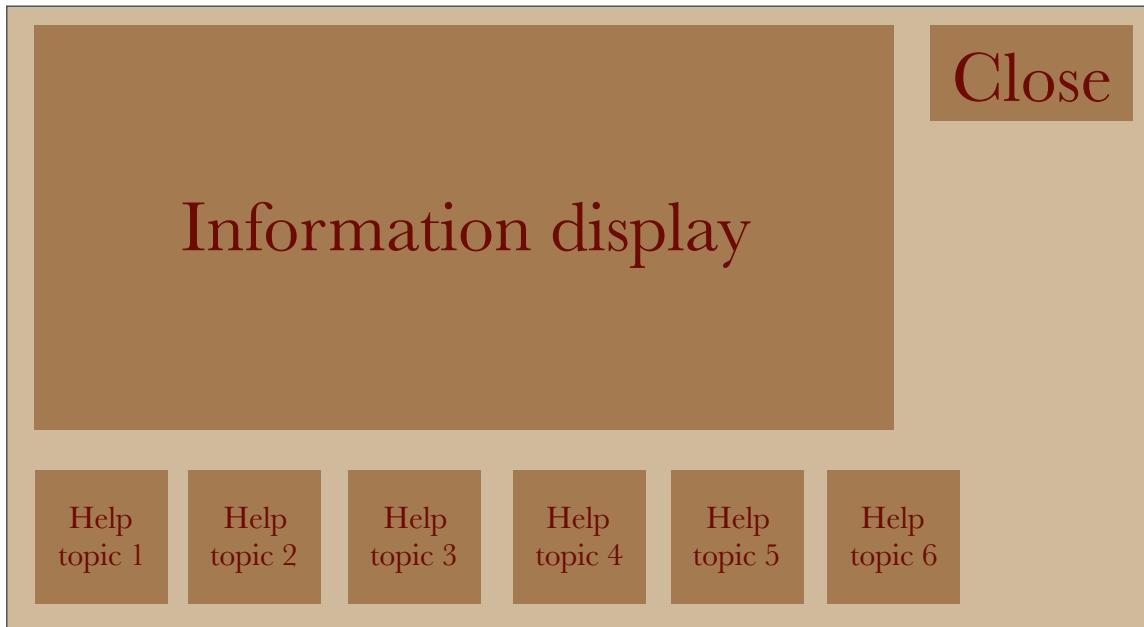
Main menu

Below is the wireframe design of the main menu. In the background will be an image related to the game. The main menu is comprised of four buttons: one to begin setup of game (Start); one to open the help menu; one to exit the game and one to access saved games. The main menu has a minimalist design and only has a few buttons as it needs to be easy for the user to access the various menus.



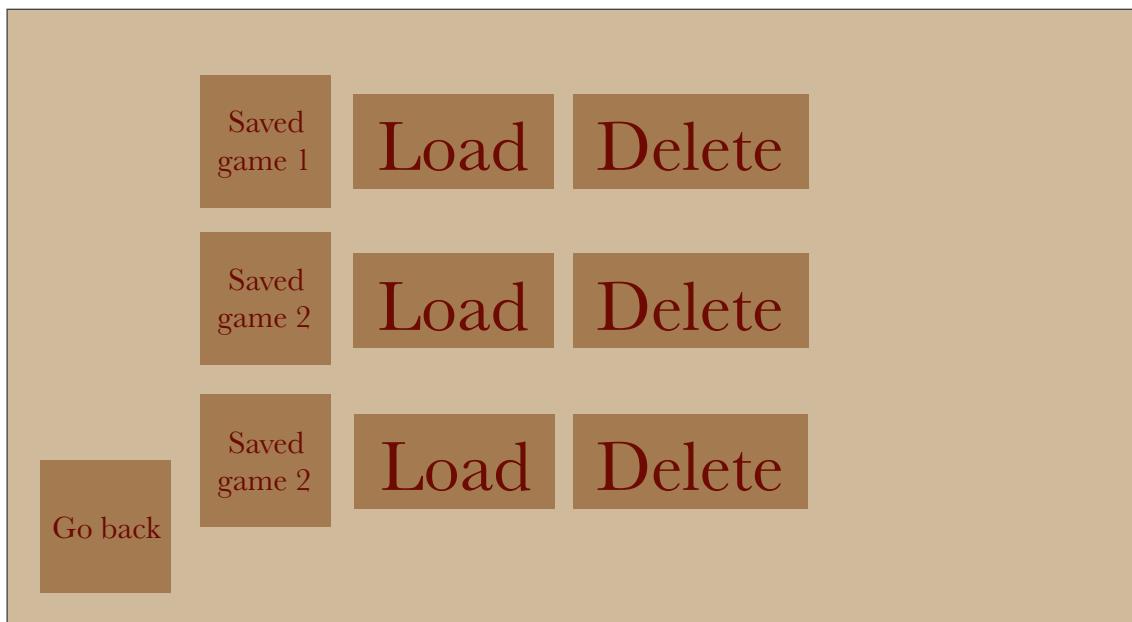
Help Menu

Below is the wireframe design of the help menu. Each button on the bottom will display the help information for the specific aspect of the game related to the button when pressed. This makes it easy for users to find information on the topic they query, rather than having to scroll through pages of unrelated information.



Saved games menu

Below is the wireframe design of the manage saved games menu. The user is shown the saved game(s) available and is presented with options to load the save or delete the save. This design makes it very clear what options the user has with each saved game. Each saved game can be titled so the user can discern multiple saved games.



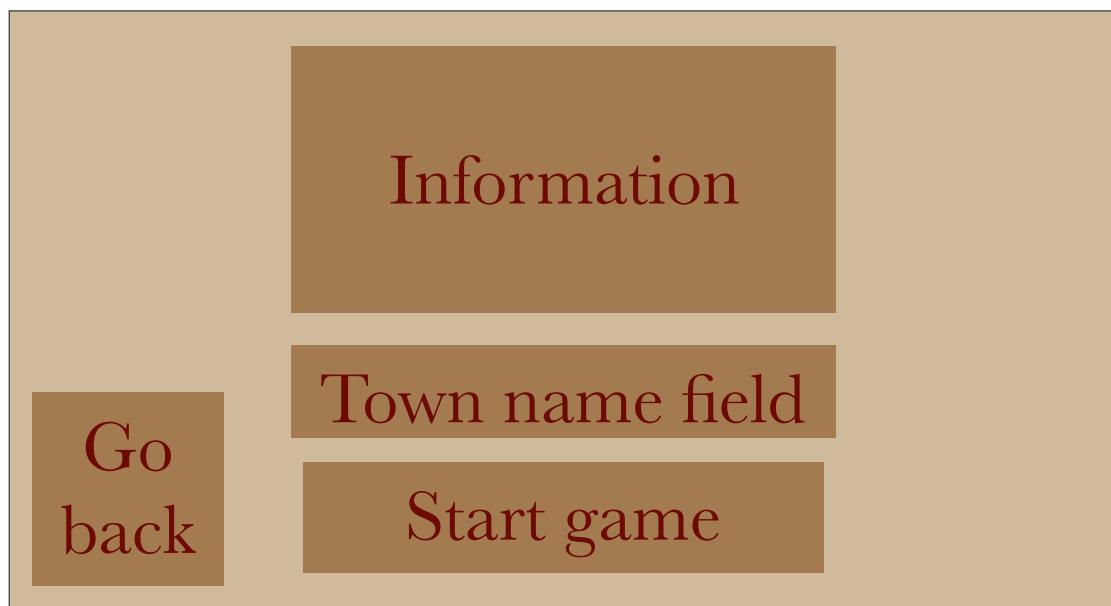
Select civilisation menu

Below is the wireframe of the select civilisation menu, the user will have the option of the civilisations specified in the success criteria requirements. They will be able to choose the civilisation they will play as and they will receive visual feedback through a display above the options this is to demonstrate to the user that the program has accepted their input. The user will not be able to progress until they have selected a civilisation to prevent errors when loading the interface.



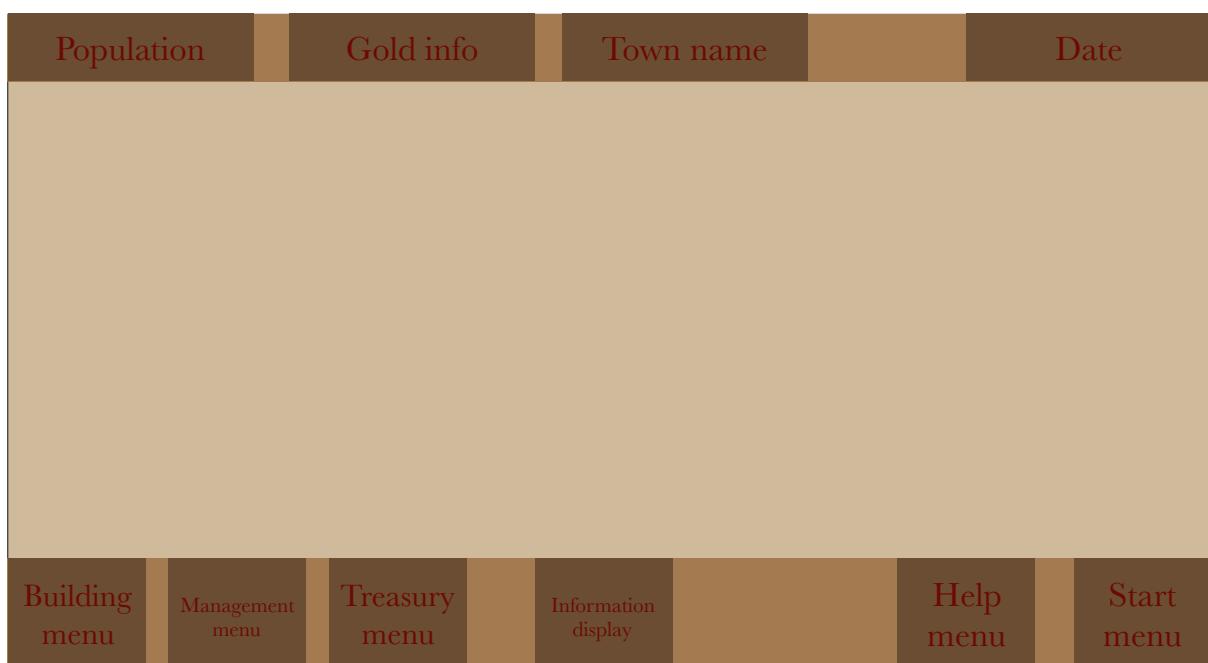
Town name menu

Below is a wireframe design of the town name menu. The user will be presented with a text box entry field to type the name of the town. There will be a maximum character limit of 14 to prevent graphical oddities in the in-game label. There is an information field to provide closure of the setup before starting the game.



In-game screen

Below is the wireframe design of the in-game user interface. The top display bar shows all of the most important information including population level, gold income and total wealth; the town name and the in-game date; so that the user has easy access to them. The bottom bar has buttons to access the building menu; the town management menu; the treasury menu, the help menu and the start menu. This addresses SC7, SC8 and SC9. There is also a general purpose information display to show relevant information such as building costs so that the user will know where to find relevant information.



Clicking the five menu buttons on the bottom display bar open their respective menus. All four of the menu's wireframe designs are shown either below or on the following pages (the help menu button will link back the help menu shown previously).

Start menu

This menu directly addresses SC9 directly. This menu includes a button to save the game and exit the game. There is also the option to close the menu (as this appears as a pop-up menu).

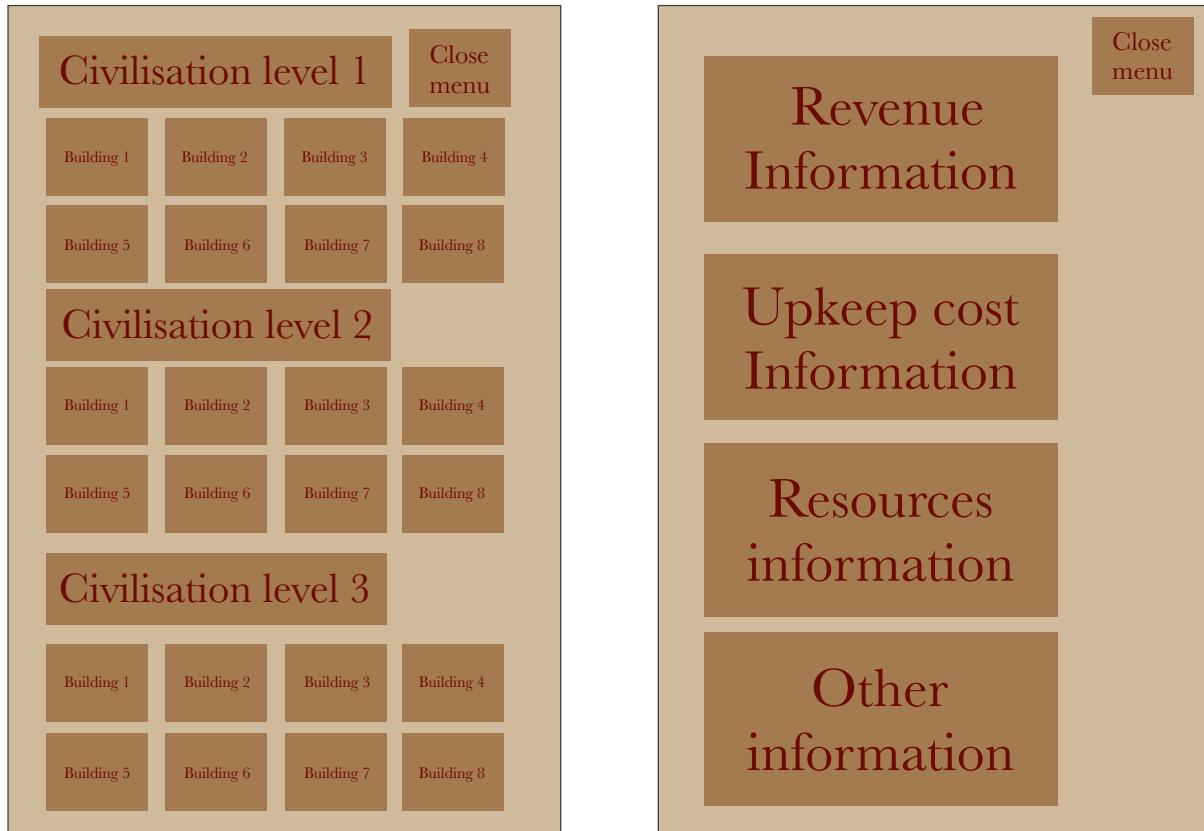


Building menu

On the following page on the left is a wireframe for the building menu. Buildings are split into the three levels of civilisation which will be implemented in a later iteration. The user will be able to select a building to place by clicking the appropriate button.

The layout has been designed so that the user knows clearly which levels of civilisation each building is related to. This wireframe does not include the exact number of buildings that will be available, but as stated by SC11.1, there will be at least 10.

Hovering over each building button will display costs and other relevant information in the information display from the in-game user-interface. This facilitates SC8.1

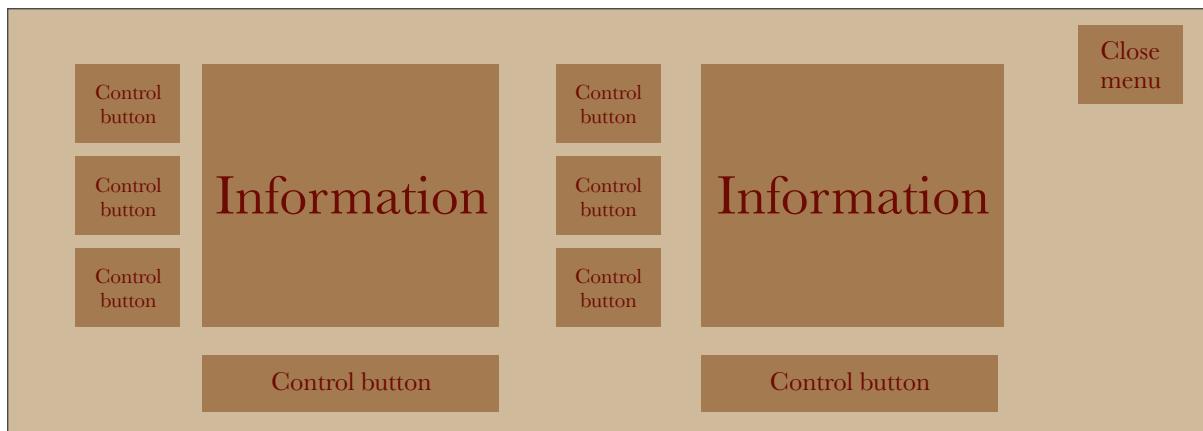


Treasury menu

Directly above on the right is a wireframe for the treasury menu. The menu is comprised solely of information as there are no action the user can perform here. Each information field will include a title to make it clear to the user what each statistic is related to. Information will include resource levels and tax income/upkeep costs information. This addresses SC8.3 directly.

Management menu

On the next page is a wireframe design of the town management menu. The user will have management options available to them, such as the ability to raze or lower taxes. The information relevant the control buttons is displayed in picture boxes. There will be information boxes of information relevant to the options available. This is to allow the user to make decisions easily. This menu addresses SC8.2.



Describe the solution (Key variable, data structures, classes and validation checks)

For the development of the user interface there are some variables which will be required. Key variables required for this iteration shown in the table below with the variable name, type and justification included as column headings.

Variable Name	Variable Type	Justification
intCivilisation	Integer	Will be used to select civilisation and will be used in this iteration to provide the user with different user interfaces that reflect the culture of the chosen civilisation.
strTownName	String	Will be used by the user to customise the name of their town. Will be displayed in-game.
intRandomNumber	Integer	For the main menu form where the background image is randomly generated, a variable is required which will be assigned a random integer which will be used to choose a background image.

This iteration will utilise multiple form classes to provide the user with many menus to make information and navigation much clearer to the user. Form classes that are required are shown below with the name and the justification.

Form Class Name	Justification
FormMainMenu	User requires a form which acts as the centre of navigation which is loaded first so the user can easily access menus and start the game.
FormHelpMenu	User will require help and information about the game and game mechanics to aid them build their town. A form is required for the user to query relevant information.

Form Class Name	Justification
FormSavesMenu	Users must have easy access to manage saved games from the main menu to ensure navigation to the game is clear, swift and does not hinder experience. Using a separate form makes managing saved games easy.
FormCivilisationMenu	After pressing the start button on the main menu, user options related to creating the town should be processed in steps to make the options clear to the user. A separate form for civilisation selection allows the user to easily make a decision.
FormTownName	Following on from FormCivilisationMenu, a separate form for entering the town name makes it clear to the user what the program is requesting of them.
FormInGame	A separate form for the game itself is essential as the user interface must be clear and not cluttered to ensure the user experience is not hindered by an incomprehensible user interface. This combined with the use of pop-up menus for the building, management and treasury menus ensure that the game will be easy to use.
FormBuildingMenu	Having a separate form for the building menu allows the user to open and close it while the in-game form is open which produces a simple user interface.
FormManagementMenu	Having a separate form for the management menu allows the user to open and close it while the in-game form is open which produces a simple user interface.
FormTreasuryMenu	Having a separate form for the treasury menu allows the user to open and close it while the in-game form is open which produces a simple user interface.
FormStartMenu	Having a separate form for the start menu allows the user to open and close it or make commands while the in-game form is open which produces a simple user interface.

Navigation of the menus will be accomplished using buttons. Thus buttons are an essential part of the solution. Information will be displayed using labels and picture boxes; as the solution requires the user to have access to information about the town at any time, the user interface must include labels and picture boxes. A text box will be used in the town name form to enter the name of the town; this is required if the user is to be allowed to choose the name of the town.

Two validation checks will be necessary in this iteration. The first will be to ensure the user has chosen a civilisation before progressing. This will be accomplished by making the progression-to-next-form button disabled unless a civilisation is selected. The second will be to check the town name entered does not exceed 14 characters. This can be achieved by exploiting the character limit property of the text box by setting it to 14, this will limit the number of characters without any developer coding.

Approach to testing (Usability features)

It is necessary for my stakeholders to test to the code this iteration implements to ensure it addresses the success criteria this iteration addresses correctly.

Feature number	Usability test description
UF1.1	Does each civilisation have unique theme for the user interfaces?
UF1.2	Is the town name fully customisable within reason (of reasonable length)
UF1.3	Is there a help menu available to the user from important locations? (i.e. the main menu and from in-game)
UF1.4	Does the initial in game screen have a user interface that provides the user with the most fundamental information about the town?

Approach to testing (During development)

The table below lists the tests that will be done to inform development.

Test number	Description of test	Expected Result
DD1.1	Check that the button linking the select civilisation menu to the town name menu does nothing until a civilisation is chosen.	Nothing happens upon being clicked.
DD1.2	Check that the buttons that allow the user to select a civilisation work and provide meaningful feedback.	Picture box displaying icon updates whilst the mouse is hovering over the button and updates when the corresponding buttons are pressed and value of intCivilisation is updated.
DD1.3	Check that the in-game user interface is changed according to the civilisation chosen.	Background image updates to corresponding civilisation.
DD1.4	Check that upon returning to the select civilisation menu, no civilisation is selected from the previous game or previous selections.	User is unable to progress to town name menu and the civilisation picture box is empty.
DD1.5	The navigation buttons on each menu work correctly.	Upon clicking the navigation buttons the respective menu is opened.

Approach to testing (Post development)

After development is completed my stakeholders will perform post-development tests to test the program's robustness and to ensure that it meets the success criteria requirements this iteration addresses. These tests are shown on the following page in a table.

Test number	Description of test	Expected Result
PD1.1	Testing that clicking the select Seljuk Turks button on the select civilisation menu correctly updates the intCivilisation variable.	intCivilisation is assigned the value associated with the Seljuk Turk civilisation.
PD1.2	Testing that clicking the select Kingdom of England button on the select civilisation menu correctly updates the intCivilisation variable.	intCivilisation is assigned the value associated with the Kingdom of England civilisation.
PD1.3	Testing that clicking the select Yuan Dynasty button on the select civilisation menu correctly updates the intCivilisation variable.	intCivilisation is assigned the value associated with the Yuan Dynasty civilisation.
PD1.4	Testing that upon entering the town name into the town name menu, the value of strTownName is updated.	The value of strTownName is assigned the text in the text box of the town name menu.
PD1.5	Testing that the text of the in-game label displaying the town name takes on the value of strTownName.	The text of the label displaying the town name is equal to strTownName.
PD1.6	Testing that the button to open to the help menu from the main menu functions correctly.	The help menu is opened without closing any other form.
PD1.7	Testing that the button to open to the help menu from the in-game screen functions correctly.	The help menu is opened without closing any other form.
PD1.8	Testing that the label displaying the population level is displayed correctly.	The in-game screen contains a label denoting the population level.
PD1.9	Testing that the label displaying wealth and income is displayed correctly.	The in-game screen contains a label denoting wealth and income.
PD1.10	Testing that the label displaying the in-game date is displayed correctly.	The in-game screen contains a label stating the in-game date.
PD1.11	Testing that the building menu button opens the construction menu correctly.	The building menu opens upon this button being clicked.
PD1.12	Testing that the management menu button opens the management menu correctly.	The management menu opens upon this button being clicked.
PD1.13	Testing that the treasury button opens the treasury menu correctly.	The treasury menu opens upon this button being clicked.
PD1.14	Testing that the start button opens the start menu correctly.	The start menu opens upon this button being clicked.
PD1.15	Testing that the save game button is displayed correctly in the start menu.	The save game button appears in the start menu.
PD1.16	Testing that the return to main menu functions correctly.	The main menu is opened and the in-game screen is closed.
PD1.17	Testing that the navigation buttons on each menu work correctly.	The previous menu visited by the user is opened upon clicking the back button.

Some of these tests may be made redundant as they are very similar to the during development tests.

Describe the solution (Pseudo-code algorithms)

(PC1.1) Pseudo-code for navigation (opening and closing forms):

```
Procedure LoadTargetFormButton.Click
    CurrentForm.Hide
    TargetForm.Show
End Procedure
```

(PC1.2) Pseudo-code for random number generator (to choose an image for main menu):

```
Procedure SelectImage
    Variable RandomNumber as integer = 0
    RandomNumber = Randomise(1 to 3)
    Select case RandomNumber
        Case 1
            BackgroundImage = Image1
        Case 2
            BackgroundImage = Image2
        Case 3
            BackgroundImage = Image3
    End select RandomNumber
End Procedure
```

(PC1.3) Pseudo-code for selecting a civilisation:

```
Procedure SelectCivilisation1Button.Click
    // CivilisationVariable is a global variable
    CivilisationVariable = 1
    PictureBoxDisplayCivilisationIcon.Image = Civilisation1Icon.Image
End Procedure
```

(PC1.4) Pseudo-code for mouse over events for the civilisation buttons:

```
Procedure SelectCivilisation1Button.MouseOver
    PictureBoxDisplayTemporaryCivilisation.Image = Civilisation1Icon.Image
End Procedure
```

(PC1.5) Pseudo-code for mouse leave events for the civilisation buttons:

```
Procedure SelectCivilisation1Button.MouseLeave
    PictureBoxDisplayTemporaryCivilisation.Image = Nothing
End Procedure
```

(PC1.6) Pseudo-code to verify that a civilisation has been selected when clicking the button to load the town name menu from the select civilisation menu:

```
Procedure LoadNextMenuButton.Click
    // CivilisationVariable is a global variable
    If CivilisationVariable != 0 Then
        CurrentForm.Hide
        TargetForm.Show
    End If
End Procedure
```

(PC1.7) Pseudo-code to select a user interface according to the chosen civilisation:

```
Procedure ApplyUserInterface
    // CivilisationVariable is a global variable
    Select CivilisationVariable
        Case 1
            UserInterface = UserInterfaceImage1
        Case 2
            UserInterface = UserInterfaceImage2
        Case 3
            UserInterface = UserInterfaceImage3
    End select
End Procedure
```

(PC1.8) Pseudo-code to change the name of the town according to an entry field:

```
Procedure EnterTownNameButton.Click
    // TownNameVariable is a global variable
    TownNameVariable = TownNameEntryField.Text
End Procedure
```

(PC1.9) Pseudo-code to display the name of the town:

```

Procedure DisplayTownName
    // TownNameVariable is a global variable
    LabelTownName.Text = TownNameVariable
End Procedure

```

Below is a table which describes how each algorithm forms part of the solution and which success criteria and requirements they fulfil.

Pseudo-code algorithm	Description of how algorithm forms part of solution	Success criteria/requirements fulfilled	Justification of how algorithms fulfils criteria
PC1.1	This algorithm will be used very frequently to navigate through the forms/menus	R1.1 R1.2 R1.4 R1.5	Algorithm allows navigation between menus
PC1.2	Will be used to generate the background image for the main menu (more appealing user interface)	R1.6 R01.1	Algorithm is dedicated to fulfilling this option requirement
PC1.3	Will be used to give a value to the Civilisation variable which will be used to determine user interface appearance and aspects of gameplay	R1.1	Algorithm allows the user to select a civilisation
PC1.4	Will help make the user interface clearer by providing feedback	R1.1 R1.6	Produces a clearer interface for the user to select a civilisation
PC1.5	Will help make the user interface clearer by providing feedback	R1.1 R1.6	Produces a clearer interface for the user to select a civilisation
PC1.6	Ensures that the user must pick a civilisation before continuing	R1.1	Ensures that the user must pick a civilisation to start the game
PC1.7	Provides a clear interface relevant to the civilisation	R1.1 R1.2 R1.6	Improves the appearance of the user-interface making usage easier
PC1.8	Allows the user to customise their town name, providing a better experience	R1.3	Algorithm enables the display of important information
PC1.9	Displays the town name, making the user interface more personalised	R1.3	Algorithm displays important information

Developing the solution (Coding)

I have decided to code each menu in the order that the user would encounter them whilst using the program. The order I will program the menus in is as follows:

1. Main menu
2. Help menu
3. Saves menu
4. Select civilisation menu
5. Town name menu
6. In-game screen
7. Building menu
8. Treasury menu
9. Management menu
10. Start menu

After all the menus are programmed I will implement graphics for each menu and test the functionality of each menu. Some of the forms' code includes references to a form called FormImgRef: this form contains several picture boxes with images that will be used throughout the solution. This form will be shown in this dossier towards the end of the project since the form will be constantly manipulated throughout all iterations.

Note: all variables, data structures, entities and classes are named in the format:
typeName: (for example a picture box would be named as pbMonthDisplay).

Code for the main menu form:

```
Public Class FormMainMenu
    Private Sub FormMainMenu_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Subroutine to choose a background image
        MainMenuImgGen()
    End Sub

    Private Sub btnStartGame1_Click(sender As Object, e As EventArgs) Handles btnStartGame1.Click
        'Closes the main menu and loads the civilisation menu:
        Me.Hide()
        FormCivilisationMenu.Show()
    End Sub

    Private Sub btnManageSaves_Click(sender As Object, e As EventArgs) Handles btnManageSaves.Click
```

```

'Closes the main menu and loads the manage saved games menu:
Me.Hide()
FormSavesMenu.Show()
End Sub

Private Sub btnHelp_Click(sender As Object, e As EventArgs) Handles
btnMainMenuHelp.Click
    'Loads the help menu, but does not close the main menu as the help menu is
    'also be accessible from in game:
    FormHelpMenu.Show()
End Sub

Private Sub btnExitGame_Click(sender As Object, e As EventArgs) Handles
btnExitGame.Click
    'Stops the game by closing the main menu.
    Me.Close()
End Sub

Public Sub MainMenuImgGen()
    Dim RandomNumber As Integer
    'Generate a random number between 1 and 3:
    Randomize()
    RandomNumber = CInt(Int((3 * Rnd()) + 1))
    'Use the random number in a Select Case to choose a background image:
    Select Case RandomNumber
        Case 1
            Me.BackgroundImage = FormImgRef.pbMainMenu1.Image
        Case 2
            Me.BackgroundImage = FormImgRef.pbMainMenu2.Image
        Case 3
            Me.BackgroundImage = FormImgRef.pbMainMenu3.Image
    End Select
End Sub
End Class

```

The main menu is fully coded. This menu will act as a hub for all other menus and the user can access all other forms within one or a few clicks.

Code for the help menu form:

Public Class FormHelpMenu

Private Sub btnExitHelp_Click(sender As Object, e As EventArgs) Handles
btnExitHelp.Click

'Closes the help menu, no need to open another form as the various
routes the help menu do not close any forms:

Me.Close()

End Sub

End Class

In regards to the help menu, this iteration will only implement the opening and closing of the menu. When a feature such as buildings are added in a later iteration, the help menu will be updated with the relevant information. Upon being closed, the help menu does not open any other forms as it is a pop-up menu, so that if it is accessed from within the game, it can be viewed whilst the user is playing the game.

Code for the saved games menu form:

Public Class FormSavesMenu

Private Sub btnReturnSavesMain_Click(sender As Object, e As EventArgs)
Handles btnReturnSavesMain.Click

'Close the form, load the main menu and generate a random image for
the main menu:

Me.Close()

FormMainMenu.MainMenuImgGen()

FormMainMenu.Show()

End Sub

End Class

This iteration does not address the code behind saving and loading games, and such the code above only loads the main menu and closes the saved games menu.

Code for the select civilisation menu form:

Public Class FormCivilisationMenu

Private Sub FormCivilisationMenu_Load(sender As Object, e As EventArgs)
Handles MyBase.Load

'Initialise civilisation variable, the civilisation display icon and show the
user that they cannot press the start game button.

```
Me.pbCivdisplay.Image = Nothing  
FormInGame.intCivilisation = 0  
btnStartGame2.Cursor = Cursors.No  
End Sub
```

'The code for the select England interface and buttons.

```
Private Sub btnSelectEngland_Click(sender As Object, e As EventArgs)  
Handles btnSelectEngland.Click  
    'Provide the user with feedback that they have selected England using  
    pbCivDisplay:  
    Me.pbCivdisplay.Image = FormImgRef.pbSelectEnglandIcon.Image  
    'Provide a backup of the image of pbCivDisplay for the mouse hover/  
    leave events:  
    FormImgRef.pbImageTempSave.Image =  
    FormImgRef.pbSelectEnglandIcon.Image  
    'Update the civilisation variable to the value for England (1):  
    FormInGame.intCivilisation = 1  
    'Show the user they can now start the game:  
    btnStartGame2.Cursor = Cursors.Hand  
End Sub
```

```
Private Sub btnSelectEngland_MouseHover(sender As Object, e As EventArgs)  
Handles btnSelectEngland.MouseHover  
    'Provide the user with confirmation that they are selecting the  
    civilisation of their preference by updating pbCivDisplay:  
    Me.pbCivdisplay.Image = FormImgRef.pbSelectEnglandIcon.Image  
End Sub
```

```
Private Sub btnSelectEngland_MouseLeave(sender As Object, e As EventArgs)  
Handles btnSelectEngland.MouseLeave  
    'Show the user that they are no longer in the button to choose a  
    civilisation by clearing pbCivDisplay.  
    Me.pbCivdisplay.Image = FormImgRef.pbImageTempSave.Image  
End Sub
```

'The code for the select Seljuk Turks interface and buttons.

```
Private Sub btnSelectTurk_Click(sender As Object, e As EventArgs) Handles  
btnSelectTurk.Click  
    'Provide the user with feedback that they have selected Seljuk Turks  
    'using pbCivDisplay:  
    Me.pbCivdisplay.Image = FormImgRef.pbSelectTurkIcon.Image  
    'Provide a backup of the image of pbCivDisplay for the mouse hover/  
    'leave events:  
    FormImgRef.pbImageTempSave.Image =  
        FormImgRef.pbSelectTurkIcon.Image  
    'Update the civilisation variable to the value for Seljuk Turks (2):  
    FormInGame.intCivilisation = 2  
    'Show the user they can now start the game:  
    btnStartGame2.Cursor = Cursors.Hand  
End Sub
```

```
Private Sub btnSelectTurk_MouseHover(sender As Object, e As EventArgs)  
Handles btnSelectTurk.MouseHover  
    'Provide the user with confirmation that they are selecting the  
    'civilisation of their preference by updating pbCivDisplay:  
    Me.pbCivdisplay.Image = FormImgRef.pbSelectTurkIcon.Image  
End Sub
```

```
Private Sub btnSelectTurk_MouseLeave(sender As Object, e As EventArgs)  
Handles btnSelectTurk.MouseLeave  
    'Show the user that they are no longer in the button to choose a  
    'civilisation by clearing pbCivDisplay.  
    Me.pbCivdisplay.Image = FormImgRef.pbImageTempSave.Image  
End Sub
```

'The code for the select Yuan interface and buttons.

```
Private Sub btnSelectChina_Click(sender As Object, e As EventArgs) Handles  
btnSelectChina.Click  
    'Provide the user with feedback that they have selected Yuan using  
    'pbCivDisplay:  
    Me.pbCivdisplay.Image = FormImgRef.pbSelectChinaIcon.Image  
    'Provide a backup of the image of pbCivDisplay for the mouse hover/  
    'leave events:  
    FormImgRef.pbImageTempSave.Image =  
        FormImgRef.pbSelectChinaIcon.Image  
    'Update the civilisation variable to the value for Yuan (3):  
    FormInGame.intCivilisation = 3
```

```

'Show the user they can now start the game:
btnStartGame2.Cursor = Cursors.Hand
End Sub

Private Sub btnSelectChina_MouseHover(sender As Object, e As EventArgs)
Handles btnSelectChina.MouseHover
    'Provide the user with confirmation that they are selecting the
    'civilisation of their preference by updating pbCivDisplay:
    Me.pbCivdisplay.Image = FormImgRef.pbSelectChinalIcon.Image
End Sub

Private Sub btnSelectChina_MouseLeave(sender As Object, e As EventArgs)
Handles btnSelectChina.MouseLeave
    'Show the user that they are no longer in the button to choose a
    'civilisation by clearing pbCivDisplay.
    Me.pbCivdisplay.Image = FormImgRef.pbImageTempSave.Image
End Sub

'Code for the return to main menu button.
Private Sub btnReturnCivMain_Click(sender As Object, e As EventArgs)
Handles btnReturnCivMain.Click
    'Close the form, load the main menu and generate a random image for
    'the main menu:
    Me.Close()
    FormMainMenu.MainMenuImgGen()
    FormMainMenu.Show()
End Sub

'Code for the start game button.
Private Sub btnStartGame2_Click(sender As Object, e As EventArgs) Handles
btnStartGame2.Click
    'Check that the user has selected a civilisation:
    If FormInGame.intCivilisation > 0 Then
        'If the user has not selected a civilisation they cannot start the
        'game, this can be checked by comparing the value of
        'intCivilisation
        '(which will have the values 1, 2 or 3 if a civilisation has been
        'selected) to zero.
        'Returns pbImageTempSave to empty state:
        FormImgRef.pbImageTempSave.Image = Nothing
        'Close the form and open town name form:
        Me.Close()

```

```

        FormTownName.Show()
    End If
End Sub
End Class

```

The three buttons which the user uses to choose the civilisation interact with the intCivilisation to tell the solution which civilisation has been selected. In its most trivial form, that is all this form requires, however after showing this to my stakeholders, they wanted more visual feedback to which civilisation they had selected, thus I introduced a picture box which displays icons for each civilisation when selected. This picture box also changes according to the mouse-hover event which makes the form more interactive. The solution prevents the user from progressing to the next form if no civilisation has been selected by only allowing the code within the start game button to be executed if the intCivilisation variable is greater than 0 (i.e. if a civilisation has been selected) by using an if statement.

In order to implement the variable picture box to display the civilisation icon upon the mouse-hover event being triggered, I needed to add another picture box (pbImageTempSave) to temporarily store the previous image of the display picture box whilst it was overwritten. This was stored in FormImgRef so that the user would not see it.

Code for the town name menu form:

```

Public Class FormTownName
    Private Sub FormTownName_Load(sender As Object, e As EventArgs)
        Handles MyBase.Load
            'Picks the appropriate form user interface for the individual civilisations
            Select Case FormInGame.intCivilisation
                Case 1 'England
                    'Uses a relevant background image for the form:
                    Me.BackgroundImage = FormImgRef.pbNameMenu1.Image
                    'Shows an appropriate UI for the town name entry:
                    Me.pbCivdisplay.Image =
                        FormImgRef.pbTownEnglandIcon.Image
                    'Displays a brief appropriate for the civilisation:
                    Me.pbCivBrief.Image = FormImgRef.pbCivBrief1.Image
                Case 2 'Turks
                    'Uses a relevant background image for the form:
                    Me.BackgroundImage = FormImgRef.pbNameMenu2.Image
                    'Shows an appropriate UI for the town name entry:

```

```

Me.pbCivdisplay.Image = FormImgRef.pbTownTurkIcon.Image
'Displays a brief appropriate for the civilisation:
Me.pbCivBrief.Image = FormImgRef.pbCivBrief2.Image
Case 3 'China
'Uses a relevant background image for the form:
Me.BackgroundImage = FormImgRef.pbNameMenu3.Image
'Shows an appropriate UI for the town name entry:
Me.pbCivdisplay.Image = FormImgRef.pbTownChinaIcon.Image
'Displays a brief appropriate for the civilisation:
Me.pbCivBrief.Image = FormImgRef.pbCivBrief3.Image
End Select
End Sub

```

```

Private Sub btnReturnCivMain_Click(sender As Object, e As EventArgs)
Handles btnReturnCivMain.Click
'Button is to be pressed if the user has reconsidered the chosen
civilisation, thus the civilisation menu should return to its original state:
'pbCivDisplay and pbImageTempSave returned to empty state:
FormCivilisationMenu.pbCivdisplay.Image = Nothing
FormImgRef.pbImageTempSave.Image = Nothing
'Initialises the intCivilisation variable:
FormInGame.intCivilisation = 0
'Reminds the user to select a civilisation to continue:
FormCivilisationMenu.btnStartGame2.Cursor = Cursors.No
'Closes the form and opens the Civilisation menu:
Me.Close()
FormCivilisationMenu.Show()
End Sub

```

```

Private Sub btnSettleTown_Click(sender As Object, e As EventArgs) Handles
btnSettleTown.Click
'Update the strTownName variable to the value of the entered text of
tbTownName:
FormInGame.strTownName = tbTownName.Text
Me.Close()
'Closes the form and opens the in-game menu:
FormInGame.Show()
End Sub
End Class

```

The solution displays a brief to the user to introduce the game. This changes depending upon the selected civilisation. Upon clicking the settle town button, the input of tbTownName is assigned to the variable strTownName. This allows SC3 to be realised.

Code for the in-game form:

Public Class FormInGame

'Create some global variables that must be accessible in multiple forms:

Public intCivilisation **As Integer**

Public strTownName **As String**

'Code to be executed upon loading to provide clear UI:

Private Sub FormInGame_Load(sender **As Object**, e **As EventArgs**) **Handles**
MyBase.Load

'Position the in-game menu menu to a convenient location, assuming in-game form is not moved (start location is centre and is repositioned):

Me.Top -= 150

Me.Show()

'Choose the UI for each civilisation by using the intCivilisation variable
in a select case statement

Select Case intCivilisation

Case 1

Me.BackgroundImage = FormImgRef.pbEnglandUI.Image

Case 2

Me.BackgroundImage = FormImgRef.pbTurkUI.Image

Case 3

Me.BackgroundImage = FormImgRef.pbChinaUI.Image

End Select

'Display the town name in the label lblTownName using the value of
strTownName and display the month in pbMonthDisplay

lblTownName.Text = strTownName

pbMonthDisplay.Image = FormImgRef.pbJanuary.Image

End Sub

Private Sub btnInGameHelp_Click(sender **As Object**, e **As EventArgs**) **Handles**
btnInGameHelp.Click

'Loads the help menu, but does not close the main menu as the help
menu is also be accessible from the main menu:

FormHelpMenu.Show()

End Sub

```
Private Sub btnStartMenu_Click(sender As Object, e As EventArgs) Handles  
btnStartMenu.Click
```

```
    'Opens the start menu with options to exit the game and to save the  
    game:
```

```
    FormStartMenu.Show()
```

```
End Sub
```

```
Private Sub btnBuildingMenu_Click(sender As Object, e As EventArgs)
```

```
Handles btnBuildingMenu.Click
```

```
    'Displays the building menu where construction options are available:
```

```
    FormBuildingMenu.Show()
```

```
End Sub
```

```
Private Sub btnTreasuryMenu_Click(sender As Object, e As EventArgs)
```

```
Handles btnTreasuryMenu.Click
```

```
    'Displays the treasury menu where information about resource levels and  
    income are described:
```

```
    FormTreasuryMenu.Show()
```

```
End Sub
```

```
Private Sub btnManagementMenu_Click(sender As Object, e As EventArgs)
```

```
Handles btnManagementMenu.Click
```

```
    'Displays the management menu where options for managing the  
    population and town are available.
```

```
    FormManagementMenu.Show()
```

```
End Sub
```

```
End Class
```

In its current state, the in-game screen solely hosts buttons linking to other op-up menus.

In later iterations this form will include a view of the map and the ability to interact with objects in the map. To emphasise each civilisation's theme, each civilisation has its own user interface design. The different interfaces are implemented when the form is loaded with a select case using the intCivilisation variable. The form also inputs the town name into a label when loaded. The in-game date will be implemented in a later iteration, but the form has a label for this purpose and a picture box to display an image for each month to illustrate the time of year. From here the user can access all the menus necessary to play the game.

Code for the building menu form:

Public Class FormBuildingMenu

 Private Sub FormBuildingMenu_Load(sender As Object, e As EventArgs)

 Handles MyBase.Load

 'Position the building menu to a convenient location, assuming in-game
 form is not moved (start location is centre and is repositioned):

 Me.Top -= 150

 Me.Left -= 715

 End Sub

 Private Sub btnExitBuildingMenu_Click(sender As Object, e As EventArgs)

 Handles btnExitBuildingMenu.Click

 'Hides the building menu:

 Me.Close()

 End Sub

End Class

This form consists of all the buttons to place buildings. Upon loaded it is repositioned to be exactly to the left of the in-game form, however it can be repositioned. In a later iteration clicking the individual building buttons will let the user place the respective building, and hovering over the buttons will display the costs and information related to the building.

Code for the treasury menu form:

Public Class FormTreasuryMenu

 Private Sub FormTreasuryMenu_Load(sender As Object, e As EventArgs)

 Handles MyBase.Load

 'Position the treasury menu to a convenient location, assuming in-game
 form is not moved (start location is centre and is repositioned):

 Me.Top -= 150

 Me.Left += 715

 End Sub

 Private Sub btnExitTreasuryMenu_Click(sender As Object, e As EventArgs)

 Handles btnExitTreasuryMenu.Click

 'Hide the treasury menu:

 Me.Close()

 End Sub

End Class

It is not possible at this time to know the exact details of what information will display. In later iterations I will consult my stakeholders about what exactly should be displayed in this menu. At this time it is known that this form will contain information about resources, but as neither my stakeholders or I know exactly what resources will be in the game, it is not reasonable to implement labels for each resource at this time. For now the form is programmed to be positioned directly right of the in-game form, and there is a button to close the menu. I will return to this form in later iterations.

Code for the management menu form:

Public Class FormManagementMenu

Private Sub FormManagementMenu_Load(sender As Object, e As EventArgs)
Handles MyBase.Load

'Position the management menu to a convenient location, assuming in-game form is not moved (start location is centre and is repositioned):
Me.Top += 403 '411

End Sub

Private Sub btnExitManagementMenu_Click(sender As Object, e As EventArgs) Handles btnExitManagementMenu.Click

'Hides the management menu:

Me.Close()

End Sub

End Class

In the same way that the details of the treasury menu cannot be discerned at this point, the management options cannot be detailed at this stage of development. I will return later to this form in a later iteration to fully implement the management options. Once again this form is repositioned to a convenient location (directly beneath the in-game form). There is a button which closes the menu.

Code for the start menu form:

Public Class FormStartMenu

Private Sub FormStartMenu_Load(sender As Object, e As EventArgs) Handles MyBase.Load

'Position the start menu to a convenient location, assuming in-game form is not moved (start location is centre and is repositioned):

```

    Me.Left += 480
    Me.Top += 82
End Sub

Private Sub btnExitStartMenu_Click(sender As Object, e As EventArgs)
Handles btnExitStartMenu.Click
    'Hides the start menu:
    Me.Close()
End Sub

Private Sub btnExitInGame_Click(sender As Object, e As EventArgs) Handles
btnExitInGame.Click
    'Close all the menus relevant to the in-game form and load the main
    menu:
    Me.Close()
    FormInGame.Close()
    FormBuildingMenu.Close()
    FormManagementMenu.Close()
    FormTreasuryMenu.Close()
    FormMainMenu.Show()
End Sub
End Class

```

This form serves two main purposes: to return to the main menu and saving the game. The saving game mechanic will be implemented in a much later iteration. Upon clicking the return to main menu button, all forms that could be opened from the in-game form are closed and the main menu form is opened. This form is positioned exactly where the button to open it is, making it convenient. One can close the start menu if it is not required.

The coding for this iteration is completed. On the following pages are the images of the forms. Some forms have a variety of appearances on account of the user interface varying for each civilisation (or in the case of the main menu on account of the image being randomly chosen).

Developing the solution (Testing to inform development):

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
DD1.1	Check that the button linking the select civilisation menu to the town name menu does nothing until a civilisation is chosen.	No test data. The next button on the select civilisation menu is clicked without having selected a civilisation.	Nothing happens upon being clicked.	Yes.
DD1.2	Check that the buttons that allow the user to select a civilisation work and provide meaningful feedback.	No test data. The mouse cursor will hover over the buttons for the civilisations and check that the civilisation picture box updates.	Picture box displaying icon updates whilst the mouse is hovering over the button and updates when the corresponding buttons are pressed and value of intCivilisation is updated.	Yes.
DD1.3	Check that the in-game user interface is changed according to the civilisation chosen.	No test data. Each civilisation button will be clicked and the image of the civilisation picture box will be checked.	Background image updates to corresponding civilisation.	Yes.
DD1.4	Check that upon returning to the select civilisation menu, no civilisation is selected from the previous game or previous selections.	No test data. Test that when the civilisation menu is opened, no civilisation is selected.	User is unable to progress to town name menu and the civilisation picture box is empty.	Yes.
DD1.5	The navigation buttons on each menu work correctly.	No test data, the buttons for navigation will be clicked and the menu that is loaded will be checked.	Upon clicking the navigation buttons the respective menu is opened.	Yes.

Evidence of testing:

All tests for this iteration were recorded using a screen recording software called Grabilla.

DD1.1

This test can be viewed in the file DD1.1.mp4. At 2 seconds in the video the next button is pressed, and nothing happens. This is the expected result and the test is a success.

DD1.2

This test can be viewed in the file DD1.2.mp4. The important timestamps are as follows:

- 3 seconds the mouse hover event of the select Seljuk Turks button is tested
- 6 seconds the mouse hover event of the select England button is tested
- 9 seconds the mouse hover event of the select Yuan Dynasty button is tested
- 15 seconds the mouse click event of the select England button is tested
- 19 seconds the mouse click event of the select Seljuk Turks button is tested
- 22 seconds the mouse click event of the select Yuan Dynasty button is tested

All of the effects of the events being tested were the expected result (the civilisation picture box was updated in the correct way). As the civilisation picture box was updated correctly, intCivilisation has been assigned the correct value in all cases.

DD1.3

This test can be viewed in the file DD1.3.mp4. The important timestamps are as follows:

- 3 seconds the town name UI for the Seljuk Turks is tested
- 6 seconds the in-game UI for the Seljuk Turks is tested
- 15 seconds the town name UI for the Kingdom of England is tested
- 17 seconds the in-game UI for the Kingdom of England is tested
- 24 seconds the town name UI for the Yuan Dynasty is tested
- 27 seconds the in-game UI for the Yuan Dynasty is tested

The correct user interface for each civilisation was loaded in all menus, thus this test is a success.

DD1.4

This test can be viewed in the file DD1.4.mp4. At 2 seconds the select civilisation menu is opened from the town name menu. The civilisation picture box is empty indicating that no civilisation is selected. This test is a success as a result.

DD1.5

This test can be viewed in the file DD1.5.mp4. The important timestamps are as follows:

- 2 seconds the help menu is successfully opened from the main menu
- 6 seconds the help menu is successfully closed
- 8 seconds the saves menu is successfully opened from the main menu
- 9 seconds the main menu is successfully opened from the saves menu
- 11 seconds the select civilisation menu is successfully opened from the main menu
- 16 seconds the main menu is successfully opened from the select civilisation menu
- 24 seconds the town name menu is successfully opened from the select civilisation menu
- 28 seconds the in-game screen is successfully opened from the town name menu
- 33 seconds the building menu is successfully opened from the in-game screen
- 35 seconds the management menu is successfully opened from the in-game screen
- 37 seconds the treasury menu is successfully opened from the in-game screen
- 40 seconds the building menu is successfully closed
- 41 seconds the management menu is successfully closed
- 43 seconds the treasury menu is successfully closed
- 46 seconds the help menu is successfully opened from the in-game screen
- 51 seconds the start menu is successfully opened from the in-game screen
- 55 seconds the start menu is successfully closed
- 57 seconds the main menu is successfully opened from the start menu
- 59 second the main menu is successfully closed

Stakeholder review and sign-off of iteration

After development of this iteration I asked my stakeholders to test the program against the requirements of this iteration. As a result my stakeholders have all agreed that all the iteration requirements, including the optional requirement have been fulfilled and that no further changes need to be made to the menus implemented in this version. They have allowed me to continue to the next iteration, where map generation will be implemented.

END OF FIRST ITERATION

SECOND ITERATION

Discussion with Stakeholders

The second iteration will implement the random map generation part of the solution which will take place as the in-game form is loaded and before the user is allowed to interact with the in-game form. I have discussed with my stakeholders what kind of landscape the map should generate and I have received the feedback from them. My stakeholders are keen that there should be a river; forests; plains (on which to build the town); a lake/sea and mountains/hill. They are also expecting each civilisation to have different textures for the landscape. I presented my stakeholders with the map size in tile dimensions (15 x 27 - calculated from available space (600 x 1080) ÷ tile size (40)) and they have reconsidered some previous decisions and would like me to implement a map four times larger than this and have four regions of the map which can be viewed individually. This will require me to have another user interface iteration later on in implement a mini-map and scrolling features. This iteration will implement very basic graphics for the map.

Specify the problem: requirements of map generation

The iteration must fulfil the following requirements:

R2.1. The solution must automatically generate a random map as soon as the in-game form is loaded.

R2.2. The solution must produce a map with space where the user is allowed to build (plains).

R2.3. The map generator must generate forests.

R2.4. The map generator must generate a river.

R2.5. The map generator must generate mountains.

R2.6. The map generator must generate a water feature (lake or sea)

R2.7. The map generator must randomly place hills on the plains tiles.

RO2.1. Optional: the plains tiles display a variety of graphics (will be implemented if time allows).

RO2.2. Optional: The map generator must be able to implement different terrain graphics for the different civilisations (see RO2.3 on implementation itself).

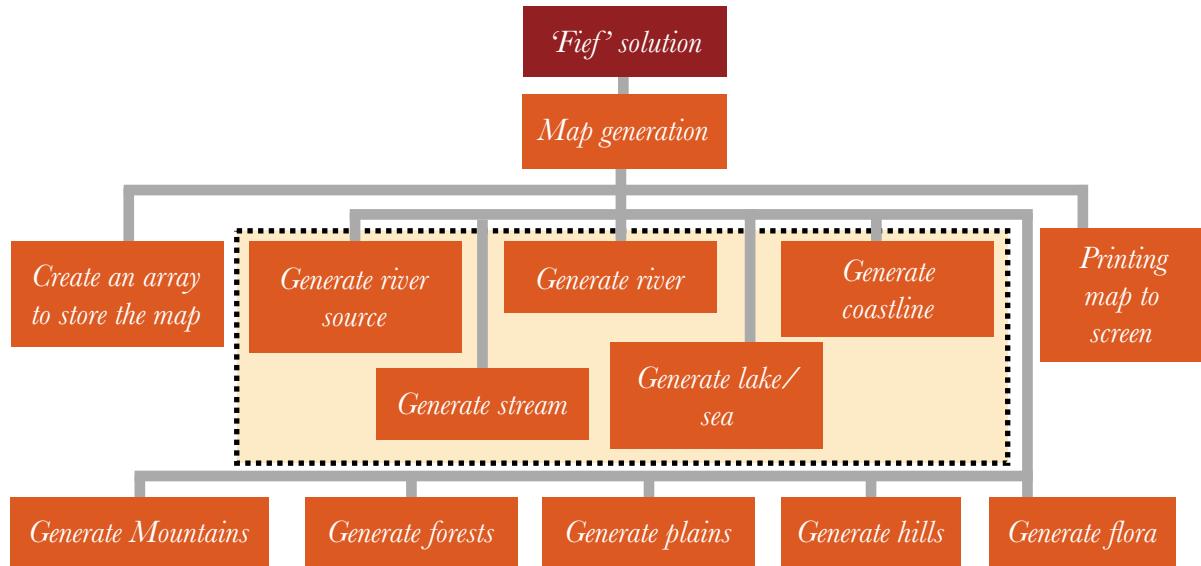
RO2.3. Optional: implement individual terrain texture sets for each civilisation (will be implemented if time allows).

Success criteria and requirements the iteration addresses:

Success Criteria	Requirement	How success criteria requirement is addressed
SC4	SCR4.1	The solution will generate a river.
SC4	SCR4.2	The solution will generate a lake or sea.
SC4	SCR4.3	The solution will generate mountains.
SC4	SCR4.4	The solution will generate forests.
SC4	SCR4.5	The solution will generate hills.
SC4	SCR4.6	The solution will generate flora.
SC4	SCR4.7	The solution will generate plains to the remaining empty tiles.

Decomposing the Problem

The diagram below illustrates the decomposition of the problem presented in this iteration. The map generation will be coded in a separate module which will be called upon when the in-game form loads. This module will be coded in chunks, each chunk containing a particular map feature: rivers, forests, etc. All the code will be included in a single module, but repeated code will be put into subroutines wherever possible (for example a random number generator). Once the map has been generated it will be printed to the in-game screen.

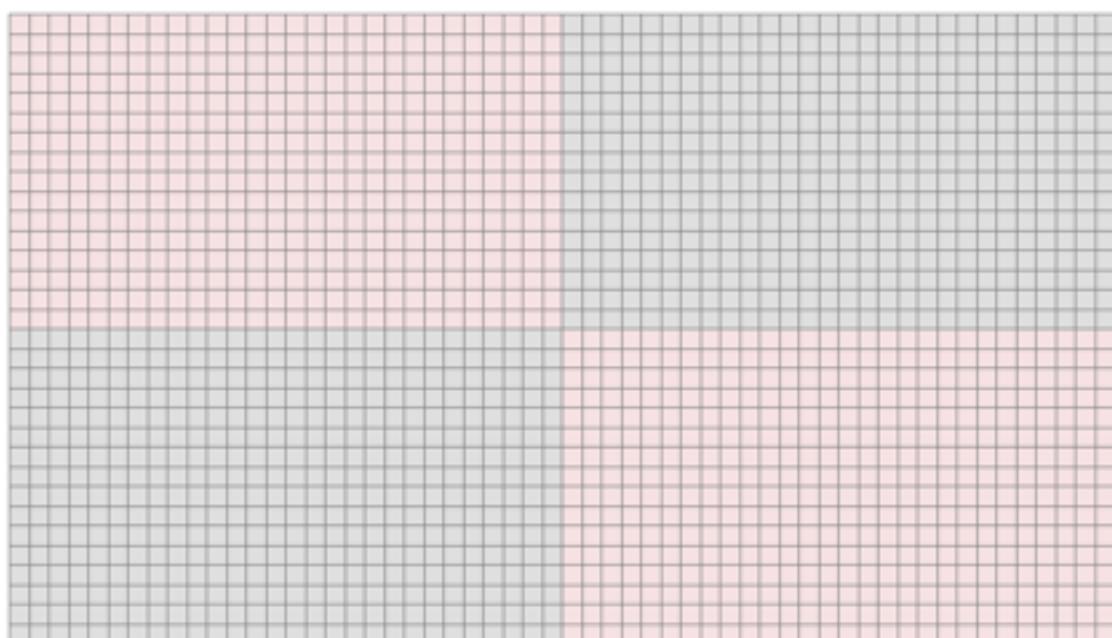


Elements that are within the yellow box are generated directly in sequence, using the previous feature as an anchor to generate the next feature (e.g the lake/sea will be generated around the river).

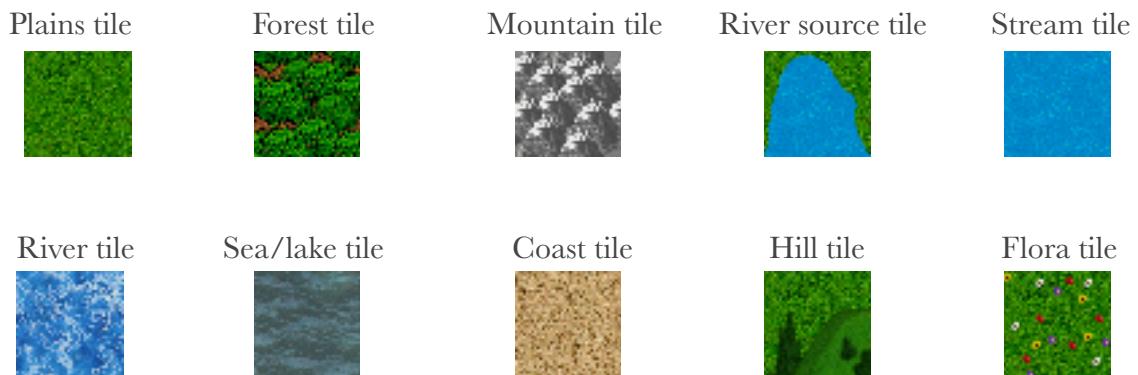
Describe the solution (Usability features)

Each possible kind of tile will be made into an object that inherits properties from a map tile class. They will have two properties: tile type and building type that is build upon them. The map will stored by a two-dimensional array with 30 rows and 54 columns. This array will be an object array consisting of map-tile objects. The array will be filled by running a series of procedures that insert the different map features into each array location. Each map feature will be added in sequence. I have decided to use object oriented programming to implement this iteration as it allows me to easily create different types of map-tiles and will allow me to store the necessary information about each tile as properties in each map-tile object. I have decided to use an array to store the map as it will be far easier to visualise the map grid and it will be simpler to reference individual map tiles when performing an action on them (e.g. placing a building). As previously mentioned, the original map size was calculated from available space \div tile size $(600 \div 40, 1080 \div 40) = (15, 27)$ but my stakeholders have demanded a larger map size thus the new map size has double the dimensions (4x the area) thus the map dimensions are 30 tiles x 54 columns hence the size of the two-dimensional array. The new dimensions allows for 1620 tiles for map generation features and building.

Below is a spreadsheet representation of the map. Each of the four coloured areas displayed represents one 15x27 area (size of original map). As the size of the in-game screen only allows for a 15x27 grid of tiles (given tile dimensions are 40x40 pixels, this means that only 25% of the screen will be available at one time).



The coding process will take place in a console application so that I will be able to see the map being generated in the same format as the spreadsheet on the previous page. The code will then be transferred to a separate module in the main program and instead of assigning the array character values, it will assign objects to each location of the array. A separate procedure will print the map to the screen using a basic texture set. A later iteration will manipulate graphics to improve the appearance of the map. Below I have displayed the basic texture set I will use to print the map to the in-game screen.



Below is an image of what I predict the in-game screen will look like after the map has been printed to the screen.



Describe the solution (Key variable, data structures, classes and validation checks)

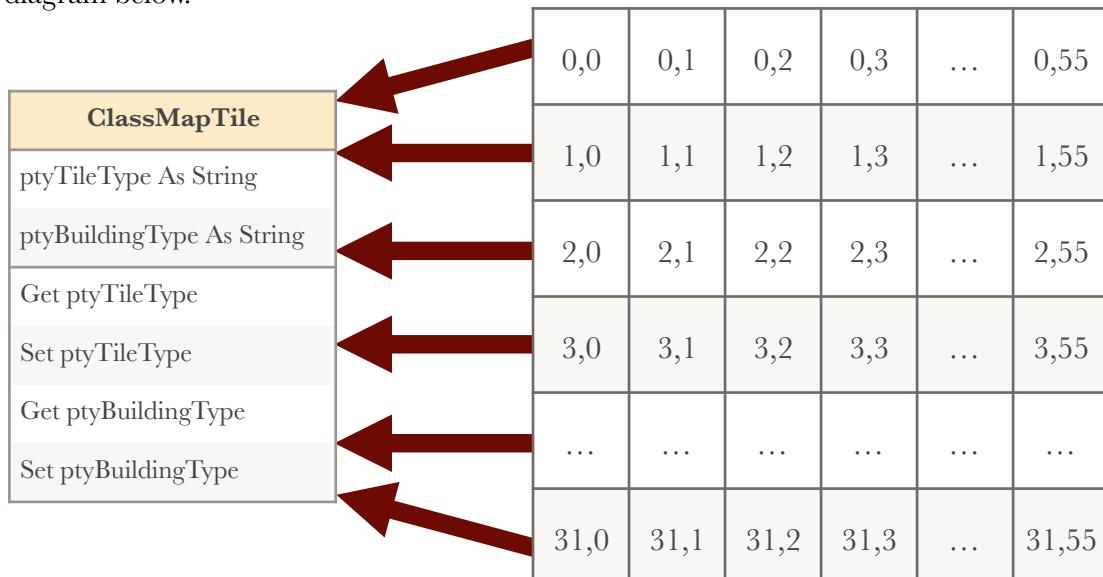
In order to generate the map, values will need to be saved to variables. These key variables are shown below with the data type and justification for the variable.

Variable Name	Variable Type	Justification
intArrYCoord	Integer	Will be used to reference rows in the array, this combined with intArrXCoord can be used to reference locations in the array.
intArrXCoord	Integer	Will be used to reference columns in the array, this combined with intArrYCoord can be used to reference locations in the array.
intTemp	Integer	Used to temporarily store variables throughout the solution.
intGraphX	Integer	Will be used in a function to graph the outline of the lake/sea.
intGraphY	Integer	Will be used in a function to graph the outline of the lake/sea.
intProbability	Integer	Required to store the probability of certain features being generated.
intMagRiverMeandering	Integer	Will be used when generating the river as a counter to determine when the river should meander.
intRiverX1	Integer	Used to store the X co-ordinate of one side of the river.
intRiverX2	Integer	Used to store the X co-ordinate of the other side of the river.
intRiverY	Integer	Required to store the Y co-ordinate of the tile where a river is being generated.
intLakeHeight	Integer	Will be used when generating a lake to graph a function to draw the outline of the lake.
intLakeWidth	Integer	Will be used when generating a lake to graph a function to draw the outline of the lake.
intLakeRight	Integer	Will be used to determine the X-co-ordinate of the maximum extent of the lake to the right of the river.
intLakeLeft	Integer	Will be used to determine the X-co-ordinate of the maximum extent of the lake to the left of the river.
intLakeLeftMax	Integer	Will be used to distribute the width of the lake either side of the river such that there will always be part of the lake generated to the right of the river. Note: there is no need for intLakeRightMax as this value will be subtracted from another variable to calculate intLakeRight.
intLakeRightWidth	Integer	Will be used in a function to graph the outline of the lake.
intLakeLeftWidth	Integer	Will be used in a function to graph the outline of the lake.
intSeaHeight	Integer	Will be used in a function to graph the outline of the sea.
dblLakeShape	Double	Will be used in a function to graph the outline of the lake. This will determine the shape of the lake.
dblSeaShape	Double	Will be used in a function to graph the outline of the sea. This will determine the shape of the sea.

Variable Name	Variable Type	Justification
intMountainProbability	Integer	Will be used to determine the frequency and size of mountain ranges.
intForestProbability	Integer	Will be used to determine the frequency and size of forests.
intFloraProbability	Integer	Will be used to determine the frequency of flora.
gfxInGameView	Graphics	Used to print map to screen (declared as System.Drawing.Graphics).
rectSource	Rectangle	Used to define the area of a bitmap image that is to be printed.
rectDestination	Rectangle	Used to define the area of gfxInGameView that is to be printed to.
bmpImage	Bitmap	Used to source an image that is to be printed to gfxInGameView.

The only data structure required for this iteration is a 2-dimensional array to store the map (will be called arrMapBlueprint). This will be an array of objects, where the objects to filled are map tiles. The size of the map as previously mentioned is 30x54 tiles, however I will make the array 32x56 in size so that the map will have a border around the map which will not be visible to the user. This is so that whenever the solution needs to check around all the locations around a visible map tile, it will not exceed the bounds of the array whilst checking around edge tiles. This data structure must be accessible throughout the solution.

This iteration will introduce a map tile class (will be called ClassMapTile). At this stage the class will have the properties: ptyTileType and ptyBuildingType, where ptyTileType is the terrain type and ptyBuildingType is the building that is built on the tile. The methods are the get and set methods for these attributes. The array arrMapBlueprint will be filled with Map tile objects, which inherit the properties and methods of the map class tile. This is illustrated in the class diagram below.



As described in the describe the solution (usability features) section of this iteration, I decided to use a class to implement the map tiles as it will allow me to store the information about each tile as attributes of the tile.

The table below displaying justification for decisions on ClassMapTile's properties and methods:

Property/method	Justification
ptyTileType	This could have been declared as any data type that has a multitude of possible values such as an integer, however I decided to make the ptyTileType a string as it makes the code much easier to understand. It is clear when I'm assigning a tile a tile type, more so than if I assigned an array location a number.
ptyBuildingType	In the same way that ptyTileType could have been assigned a variety of different data types, I chose string as it will make it easier to identify in the code when a certain building type is being used.
Get ptyTileType	It will be necessary at various points in the game to retrieve the tile type of a tile in question, for example if a building is being deleted, the graphic of the tile will need to be replaced with the terrain texture, which will require the tile type to be known.
Set ptyTileType	This will be required whenever a tile is being assigned a tile type. This will happen when the map is being generated.
Get ptyBuildingType	This will be required whenever a calculation involving buildings is required. For example when the upkeep of the buildings of the town is being calculated, the number of each building will be required, this will require a get method for this property.
Set ptyBuildingType	This will be required whenever a user wants to construct a building on a tile. The building type of the tile will need to be changed when a building is constructed.

Using this method of filling the Map array with objects inheriting from the map tile class, I will be able to store all the necessary information about each tile.

This iteration will not require any validation checks as the user will not be entering any information.

Approach to testing (Usability features)

My stakeholders will perform a usability test to ensure this iteration successfully fulfils the success criteria this iteration addresses.

Feature number	Usability test description
UF2.1	Does the map generate randomly?

Approach to testing (During development)

As the map will be randomly generated, the solution will be tested by examining the output of the map generation module and checking whether each feature of the map is generated in the expected co-ordinate region. This will be accomplished by coding the solution in a console application (as described in usability features) and viewing the output of each section of code.

Test number	Description of test	Expected Result
DD2.1	Test that the random number generator returns numbers within the bounds (passed to the function via parameter passing).	The value of the returned number is within or equal to the input bounds.
DD2.2	Test that a border has been generated around the edge of the map.	Tiles of co-ordinates ($X = 0$ or $X = 55$, $Y = 0$ or $Y = 31$) are made border tiles.
DD2.3	Check that all remaining locations in the array are assigned empty map tiles.	The array is filled with empty tiles.
DD2.4	Test whether the river source generates in the expected region.	River source tile consistently generates in the region ($8 \leq X \leq 47$, $2 \leq Y \leq 3$).
DD2.5	Check that the stream tiles generate in the expected location.	Stream tiles generate directly beneath the pre-generate river source tile.
DD2.6	Ensure the dimensions of the stream are correct.	Stream should be between 2 and 4 tiles in length and 1 tile wide.
DD2.7	Check that the river generates in the correct location.	The river generates directly beneath the last stream tile.
DD2.8	Check that the river is of the correct dimensions and meanders in intervals decided by the variable intMagRiverMeandering.	The river is 2 tiles wide and runs to the bottom of the map. A section of the the river translates 1 tile left or right at intervals equal to the value of intMagRiverMeandering.
DD2.9	Check that the lake is generated in the correct location with appropriate dimensions.	The lake is generated around the bottom of the lake sized to fit within the bounds of the map array.
DD2.10	Test that the shape of the lake shape is randomly generated and that the shape of the lake is different either side of the river.	The lake outline will curve differently either side of the river.
DD2.11	Check that the water feature generation does not exceed bounds of array.	Array bounds are not exceeded.
DD2.12	Test that the sea shape is shaped correctly and in the correct location.	The sea shape is modelled as either a cosine or a sine function. The amplitude of the sine/cosine function should be a maximum of 3. Centred around a co-ordinate determined by intSeaHeight and intSeaShape.

Test number	Description of test	Expected Result
DD2.13	Check that the area enclosed by pre-generated outline (either lake or sea) is filled in by sea tiles.	All tiles within the outline are assigned the sea tile type.
DD2.14	Ensure that the edge of the water feature is enclosed by coast tiles.	All border sea tile types are lined by coast tiles.
DD2.15	Check that the river source tile is outlined by mountain tiles, with the exception of the stream tiles.	The river source tile will have mountain tiles generated in all tiles directly adjacent (except directly beneath) and in all directly diagonal tiles.
DD2.16	Test that the solution generates mountain ranges in the top of the map within the expected boundaries.	Mountain ranges are generated within the co-ordinate bounds of $(1 \leq X \leq 54, 1 \leq Y \leq 4)$
DD2.17	Check that all empty tiles in the top 3 rows are converted to forest tiles.	No empty tiles remain in the top 3 rows.
DD2.18	Test that forests are generated from the sides and bottom of the map.	Forests are generated in random shapes of varying sizes.
DD2.19	Check that all remaining empty tiles are converted to plains tiles.	No empty tiles remain.
DD2.20	Test that hills of the correct dimensions are generated randomly around the map.	Hills of size 2x2 are generated randomly around the map on plains tiles.
DD2.21	Test that flora tiles populate the map in random distribution.	Flora tiles generate randomly on-top of plains tiles.
DD2.22	Ensure that all tiles are correctly assigned the correct graphic.	All tiles will print their respective graphic to the in-game screen.

The test data for DD2.1 will be 1 for the lower bound and 10 for the upper bound. I will expect the function to return the lower bound, upper bound and one other value at least once. An invalid value will also be sought. DD2.11 will be tested using dry runs as the water feature is the only features that has the opportunity to exceed the bounds of the array. Inputs with the most extreme circumstances will be tested. These inputs will be where the river source has generated as far left or right as possible and the river has the largest possible meandering value.

Approach to testing (Post development)

My users will perform post development tests to test robustness and to ensure that this iteration fulfils the success criteria requirements this iteration addresses. The tests are shown in the table below with descriptions of the tests and expected results.

Test number	Description of test	Expected Result
PD2.1.1	Does the solution generate a river source tile?	River source tile generates in the region $(8 \leq X \leq 47, 2 \leq Y \leq 3)$.

Test number	Description of test	Expected Result
PD2.1.2	Does the solution generate a stream leading from the river source tile?	Stream of length 2 to 4 tiles generated beneath river source tile.
PD2.1.3	Is a river generated beneath the last stream tile?	River with a width of 2 tiles generates, meandering to the bottom of the map.
PD2.2.1	If a lake is generated, is it shaped randomly with varying curvature?	Lake is generated around bottom of river with varying shape either side of river.
PD2.2.2	If a sea is generated, does the shape of the sea appear curved?	Shape of sea appears modelled upon sine or cosine function.
PD2.2.3	Are the tiles surrounding the edge of the water feature replaced with coast tiles?	All tiles (except river tiles) adjacent to the edge sea tiles are replaced with coast tiles.
PD2.3	Is the top of the map populated by mountain ranges of random shape?	Mountain ranges are generated within the co-ordinate bounds of ($1 \leq X \leq 54, 1 \leq Y \leq 4$)
PD2.4	Are the remaining edges of the map filled with randomly shaped and sized forests?	Forests of varying shape and size are generated around the remaining edges of the map.
PD2.5	Does the map feature hills of size 2x2 tiles?	Map is filled randomly with hills of the specified size.
PD2.6	Does the map include randomly placed flora tiles?	Plains tiles are replaced randomly with flora tiles.
PD2.7	Are all remaining empty tiles filled with plains tiles?	No empty tiles remain after plains tiles are generated.

Describe the solution (Pseudo-code algorithms)

(PC2.1) Algorithm for random number generator:

```

Function RandomNum(ByVal LowerBound As Integer, ByVal UpperBound As Integer)
    RandomNumber = ConvertToInteger((UpperBound - LowerBound + 1) * Rnd())
    + LowerBound)
    // Rnd() is an inbuilt function in Visual Studio which generates a
    random real number between 0.000 and 1.000
    Return RandomNumber
End Function

```

(PC2.2) Algorithm to create a border around the edge of the array and fill in the centre of the map blueprint with empty map tiles:

For Y = 0 To 31

```

For X = 0 To 55
    MapArray(Y, X) = New Maptile //Create a maple object in the array
    location
    If Y = 0 Or Y = 31 Or X = 0 Or X = 55 Then
        MapArray(Y, X).ptyTileType = "Border"
    Else
        MapArray(Y, X).ptyTileType = "Empty"
    End If
    Next X
    Next Y

```

(PC2.3) Algorithm to generate a river feature.

```

//First generate river source tile
Y = RandomNum(2, 3)
X = RandomNum(8, 48)
MapArray(Y, X).ptyTileType = "RiverSource"
//Second generate a stream beneath
Counter = Y
For Y = Counter To RandomNum(Y + 2, Y + 4) //Stream 2 to 4 tiles in length
    MapArray(Y, X).ptyTileType = "Stream"
Next Y
//Third generate a larger river that meanders
RiverX1 = X
RiverY = Y
RiverMeanderFrequency = RandomNum(3, 8) //Variable decides how often the river
meanders
Select Case RandomNum(1, 2) //Decides which side of the stream the river expands
Case 1
    RiverX2 = RiverX1 - 1 //Expand left
Case 2
    RiverX2 = RiverX1 + 1 //Expand right
End Select
Counter = RiverY
While RiverY < 31 //Ensures the river does not exceed the map limits
    For RiverY = Counter To RiverY + RiverMeanderFrequency //River will
    expand downwards until it is supposed to meander

```

```

        MapArray(RiverY, RiverX1).ptyTileType = "River"
        MapArray(RiverY, RiverX2).ptyTileType = "River"
        If RiverY = 30 Then
            Exit While
        End If
        Next RiverY
        Select Case RandomNum(1, 2)
        Case 1
            RiverX1 = RiverX1 - 1 //Meander left
            RiverX2 = RiverX2 - 1 //Meander left
        Case 2
            RiverX1 = RiverX1 + 1 //Meander right
            RiverX2 = RiverX2 + 1 //Meander right
        End Select
        Counter = RiverY //Reset the counter variable to repeat the countdown to next
        meander
    End While

```

(PC2.4) Algorithm to generate a lake.

```

//Lake is generated around bottom of river, with its dimensions calculated by the X co-
ordinates of RiverX1 and RiverX2, thus RiverX1 and RiverX2 must be consistently
arranged (RiverX1 on the left and RiverX2 on the right)
If MapArray(RiverY, RiverX1 + 1).ptyTileType = "Empty" Then //Swap
    Temp = RiverX2
    RiverX2 = RiverX1
    RiverX1 = Temp
End If
//Dimensions of lake are now decided randomly
LakeWidth = RandomNum(15, 27)
LakeHeight = RandomNum(4, 10)
//The following calculations are building up to determining values for variables
LakeLeftWidth and LakeRightWidth which will be used in a function to map the
curvature of the lake
LakeLeftMax = LakeWidth - 2 //guarantees that the right side of the lake will have at
least 2 tiles
If RiverX1 - LakeWidth < 1 Then

```

```

LakeLeftMax = RiverX1 - 1 //Update LakeLeftMax to ensure generation never
exceeds bounds of array

End If

LakeLeft = RiverX1 - RandomNum(2, LakeLeftMax - 2)
LakeRight = RiverX2 + (LakeWidth - (RiverX1 - LakeLeft))
LakeLeftWidth = RiverX1 - LakeLeft
LakeRightWidth = LakeRight - RiverX2
//Now all but one variable required to graph the function of the lake shape has been
decided, the last variable is decided differently on each side of the lake. This is due to
how the graphing function works. After PC2.4 I will discuss in depth this reason and
how the function is designed†

//Create outline of lake for the left side
LakeShape = (4 * RandomNum(2, 8)) / 10
For GraphX = -LakeLeftWidth To 0
    GraphY = ConvertToInteger((LakeHeight^LakeShape) - ((GraphX / (-
        LakeLeftWidth / LakeHeight))^LakeShape))^(1/LakeShape)
    X = RiverX1 + GraphX
    Y = 30 - GraphY //Converting from graph output to array value
    MapArray(Y, X).ptyTileType = "Sea"
Next GraphX

//Create outline of lake for the right side
LakeShape = (2 * RandomNum(4, 16)) / 10
For GraphX = 0 To LakeRightWidth
    GraphY = ConvertToInteger((LakeHeight^LakeShape) - ((GraphX / (-
        LakeRightWidth / LakeHeight))^LakeShape))^(1/LakeShape)
    X = RiverX2 + GraphX
    Y = 30 - GraphY //Converting from graph output to array value
    MapArray(Y, X).ptyTileType = "Sea"
Next GraphX

//Now the outline is created, fill in the remainder of the lake
For X = LakeLeft To LakeRight
    For Y = 30 - LakeHeight To 30
        If MapArray(Y, X).ptyTileType = "Sea" Then
            Counter = GraphY
            For GraphY = Counter To 30
                MapArray(Y, X).ptyTileType = "Sea"

```

```

    Next GraphY
End If
Next Y
Next X

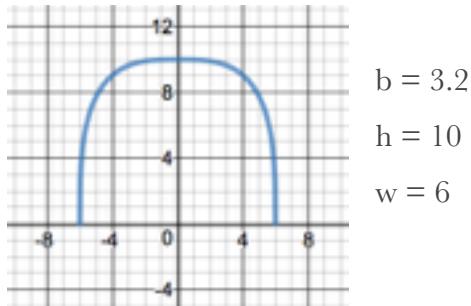
```

†I designed this function on an online graphing program called desmos. The equation is as follows:

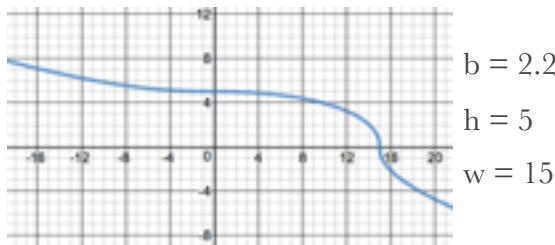
$$y = \left(h^b - \left(\frac{(x)}{\left(\frac{w}{h} \right)} \right)^b \right)^{\frac{1}{b}}$$

h = LakeHeight (The Y-intercept of the graph)
w = lakeLeftWidth/LakeRightWidth (The X-intercept of the graph)
b = LakeShape

The following images demonstrate the variation in the graph appearance upon changing these variables:

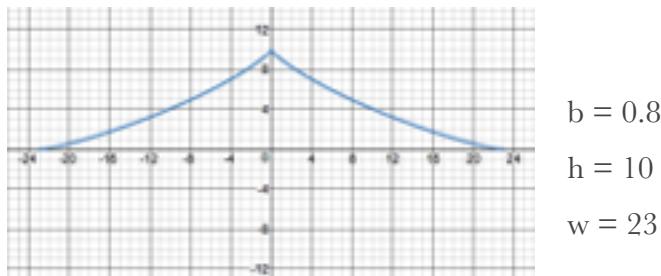


b = 3.2
h = 10
w = 6



b = 2.2
h = 5
w = 15

Note the shape of the left side of the curve due to the value of b (LakeShape). This is why LakeShape has to be calculated differently on either side of the lake to prevent this error. This error occurs only when $(b * 10) / 4 \neq \text{integer}$ e.g. $(2.2 * 10) / 4 = 5.5 \neq \text{integer}$. Consequentially the random number associated with the lake shape of the left side of the lake is multiplied by 4. A similar anomaly occurs in the right side when the result of $(b * 10) / 2 \neq \text{Integer}$; and has a similar solution to the error.



b = 0.8
h = 10
w = 23

(PC2.5) Algorithm to generate a sea.

```
SeaHeight = RandomNum(2, 4) //Defines the minimum point of the graph  
SeaShape = RandomNum(6, 15) / 10 //defines the amplitude and period of the sine/  
cosine function
```

```
Select Case RandomNum(1, 2) //Randomly picks a sine or cosine function
```

```
Case 1
```

```
For GraphX = 1 To 54  
    GraphY = ConvertToInteger(SeaShape * Sin(GraphX / (2 * SeaShape)))  
    X = GraphX  
    Y = GraphY + (30 - SeaHeight)  
    MapArray(Y, X).ptyTileType = "Sea"  
    If GraphY = 1 Then  
        Select Case Temp = RandomNum(1, 3)  
            Case Temp < 3  
                SeaShape = RandomNum(6, 15) / 10  
        End Select  
    End If  
Next GraphX
```

```
Case 2
```

```
For GraphX = 1 To 54  
    GraphY = ConvertToInteger(SeaShape * Cos(GraphX / (2 * SeaShape)))  
    X = GraphX  
    Y = GraphY + (30 - SeaHeight)  
    MapArray(Y, X).ptyTileType = "Sea"  
    If GraphY = -1 Then  
        Select Case Temp = RandomNum(1, 3)  
            Case intTemp < 3  
                SeaShape = RandomNum(6, 15) / 10  
        End Select  
    End If  
Next GraphX
```

```
End Select
```

```
// Now outline is created, fill in remaining sea tiles
```

```
For X = 1 To 54
```

```
    For Y = 1 To 30
```

```
        If MapArray(Y, X).ptyTileType = "Sea" Then
```

```

Count = Y
For GraphY = Count To 30
    MapArray(GraphY, X).ptyTileType = Sea
    Next GraphY
End If
Next Y
Next X

```

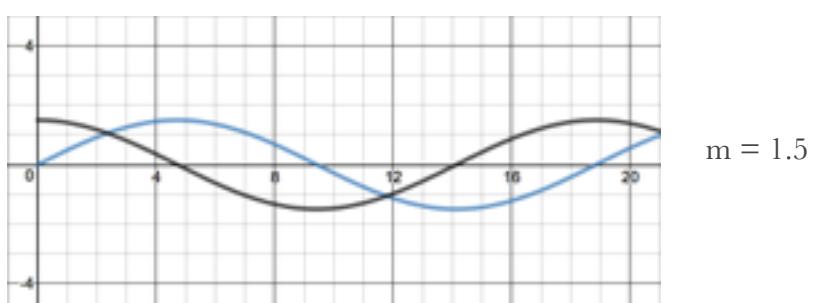
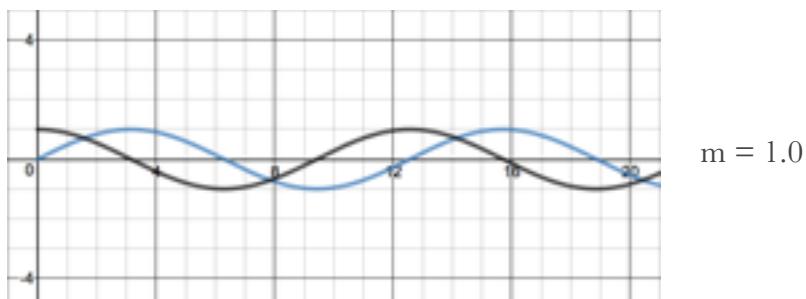
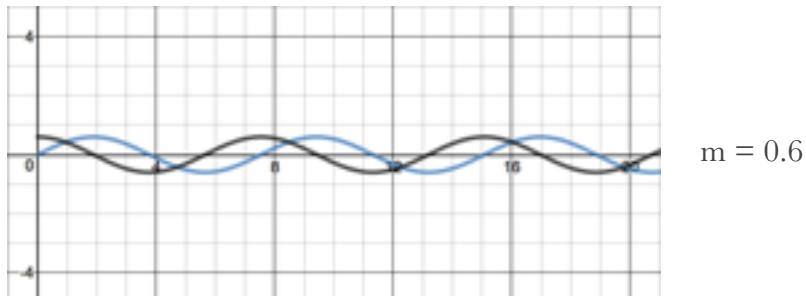
ΔThese functions were also designed in desmos. The equations of the functions are as follows:

$$y = m \sin\left(\frac{x}{2m}\right)$$

m = SeaShape

$$y = m \cos\left(\frac{x}{2m}\right)$$

These following images demonstrate how the graph of these functions change according to the value of m.



(PC2.6) Algorithm to generate a coastline.

```
//First loop down the columns of the array location the first sea tile and adding a coast  
tile above it  
  
For X = 1 To 54  
    For Y = 1 To 30  
        If MapArray(Y, X).ptyTileType = "Sea"  
            If MapArray(Y, X).ptyTileType = "River" Then  
                Exit For  
            End If  
            MapArray(Y - 1, X).ptyTileType = "Coast"  
            Exit For  
        End If  
    Next Y  
  
Next Y  
  
//Now loop through the array rows adding coast tiles to the appropriate location  
For Y = 1 To 30  
    For X = 1 To 54  
        If MapArray(Y, X).ptyTileType = "Sea" And MapArray(Y, X -  
1).ptyTileType = "Empty" Then  
            MapArray(Y, X - 1).ptyTileType = "Coast"  
        End If  
        If MapArray(Y, X).ptyTileType = "Sea" And MapArray(Y, X +  
1).ptyTileType = "Empty" Then  
            MapArray(Y, X + 1).ptyTileType = "Coast"  
        End If  
    Next X  
  
Next Y
```

(PC2.7) Algorithm to generate mountain ranges.

```
//First locate the river source tile and generate mountains on all the surrounding tiles  
except the stream tile beneath it  
  
For Y = 2 To 4  
    For X = 8 To 47  
        If MapArray(Y, X).ptyTileType = "RiverSource" Then  
            Exit For //X co-ordinate of river source tile saved  
        Next X
```

```

If MapArray(Y, X).ptyTileType = "RiverSource" Then
    Exit For //Y co-ordinate of river source tile saved

Next Y

MapArray(Y + 1, X - 1).ptyTileType = "Mountain" //Below and left
MapArray(Y, X - 1).ptyTileType = "Mountain" // Directly left
MapArray(Y - 1, X - 1).ptyTileType = "Mountain" // Above and left
MapArray(Y - 1, X).ptyTileType = "Mountain" //Directly above
MapArray(Y - 1, X + 1).ptyTileType = "Mountain" //Above and right
MapArray(Y, X + 1).ptyTileType = "Mountain" //Directly right
MapArray(Y + 1, X + 1).ptyTileType = "Mountain" //Below and right

//Next loop along the top row of the map adding mountain tiles which will act as
anchors for spreading of mountain ranges. Anchors are randomly added using a
comparison of random numbers and a mountain probability variable which is randomly
generated

For X = 1 To 54

    MountainProbability = RandomNum(1, 3)
    If MountainProbability * RandomNum(1, 8) > RandomNum(12, 16) Then
        MapArray(1, X).ptyTileType = "Mountain"
    End If

Next X

//After designing the pseudo-code for the spread of mountain ranges, it was apparent
that a lot of code was duplicated, so I made a separate subroutine for this code (see
PC2.8)

GenerateMountains(4, 1)
GenerateMountains(4, 2)
GenerateMountains(3, 3)
GenerateMountains(2, 4)

//The parameters passed into the GenerateMountains subroutine will be explained in
PC2.8

```

(PC2.8) Algorithm to spread mountain ranges.

Procedure GenerateMountains(Probability, Y) //Probability is a parameter which is used to decide the likelihood of a mountain range expanding, Y is the current row on which mountains are being generated upon

Temp = Probability //Stores the original value of the Probability variable so that it can be reset after being changed

```

//First expand mountain ranges from the mountains above
For X = 1 To 54
    If MapArray(Y - 1, X).ptyTileType = "Mountain" And MapArray(Y,
X).ptyTileType != "RiverSource" And MapArray(Y, X).ptyTileType != "Stream" Then //Ensures river source or streams tiles are not
overwritten
        Select Case intSelect = RandomNum(1, Probability)
        Case 1
            //Do nothing and reset probability value
            Probability = Temp
        Case 1 < intSelect
            //Generate a mountain
            MapArray(Y, X).ptyTileType = "Mountain"
            Probability = Probability - 1 //Reduces chance of
                mountain range expanding again from the same anchor
        End Select
    End If
Next X
Probability = Temp //Reset value of probability
//Now loop through the current row left to right expanding mountain ranges
For X = 1 To 54
    If MapArray(Y, X - 1).ptyTileType = "Mountain" And MapArray(Y,
X).ptyTileType != "RiverSource" And MapArray(Y, X).ptyTileType != "Stream" Then //Ensures river source or streams tiles are not
overwritten
        Select Case intSelect = RandomNum(1, Probability)
        Case 1
            //Do nothing and reset probability value
            Probability = Temp
        Case 1 < intSelect
            //Generate a mountain
            MapArray(Y, X).ptyTileType = "Mountain"
            Probability = Probability - 1 //Reduces chance of
                mountain range expanding again from the same anchor
        End Select
    End If

```

```

Next X

Probability = Temp //Reset value of probability
//Now loop through the current row right to left expanding mountain ranges
For X = 54 To 1 Step -1 //Loops right to left
    If MapArray(Y, X + 1).ptyTileType = "Mountain" And MapArray(Y,
        X).ptyTileType != "RiverSource" And MapArray(Y, X).ptyTileType !=
        "Stream" Then //Ensures river source or streams tiles are not
        overwritten
            Select Case intSelect = RandomNum(1, Probability)
                Case 1
                    //Do nothing and reset probability value
                    Probability = Temp
                Case 1 < intSelect
                    //Generate a mountain
                    MapArray(Y, X).ptyTileType = "Mountain"
                    Probability = Probability - 1 //Reduces chance of
                    mountain range expanding again from the same anchor
            End Select
        End If
    Next X
End Procedure

```

(PC2.9) Algorithm to create forests.

```

//First convert all empty tiles in the top 3 rows to forest
For Y = 1 To 3
    For X = 1 To 54
        If MapArray(Y, X).ptyTileType = "Empty"
            MapArray(Y, X).ptyTileType = "Forest"
        End If
    Next X
Next Y
//Improve the look of the top forest by converting some of the empty tiles on the fourth
row to forest tiles
For X = 1 To 54
    If MapArray(3, X).ptyTileType = "Forest" Then
        Select Case RandomNum(1, 2) //1/2 chance of forest being generated

```

```

Case 1 //Add forest tile
    MapArray(4, X).ptyTileType = "Forest"
End Select
End If
Next X
//Forests will be generated in a similar fashion to mountain ranges (i.e. creating anchor
points and generating around them
//Forest anchors will be randomly placed on the first two and last two columns and the
bottom two rows where a sea does not exist
//As I don't want the top three rows' forests to spread, the remaining forests will be
generated using a temporary ptyTileType, and converted to forest tiles later on
For X = 1 To 54
    For Y = 4 To 30
        ForestProbability = RandomNum(1, 3)
        If ForestProbability * RandomNum(1, 8) > RandomNum(12, 16) And
            MapArray(Y, X).ptyTileType = "Empty" Then
                MapArray(Y, X).ptyTileType = "Temporary"
            End If
        If X = 2 And Y = 30 Then
            X = 53 //This skips all the middle columns the the penultimate
            column
        End If
    Next Y
Next X
For Y = 29 To 30
    For X = 1 To 54
        If RandomNum(1, 3) > 1 And MapArray(Y, X).ptyTileType =
            "Empty" Then
            MapArray(Y, X).ptyTileType = "Temporary"
        End If
    Next X
Next Y
//Now that the anchors have been generated, the forests must be allowed to spread.
Forests will spread in one of two ways and there is also a chance that they will not spread:
there will be a probability of 1/3 that all 8 tiles around a forest (temporary) tile will be
converted to forest (temporary) tiles; a probability of 1/3 that only the 4 adjacent tiles

```

will be converted to forest (temporary) tiles and finally a probability of 1/3 that nothing will be generated

For Y = 3 To 30

 For X = 1 To 54

```
        If MapArray(Y, X).ptyTileType = "Temporary" Then
            Select Case Probability = RandomNum(1, 3)
                Case 1 //Spread on all 8 tiles around the target tile
                    If MapArray(Y + 1, X - 1).ptyTileType = "Empty" Then
                        MapArray(Y + 1, X - 1).ptyTileType =
                        "Temporary"
                    End If
                    If MapArray(Y, X - 1).ptyTileType = "Empty" Then
                        MapArray(Y, X - 1).ptyTileType = "Temporary"
                    End If
                    If MapArray(Y - 1, X - 1).ptyTileType = "Empty" Then
                        MapArray(Y - 1, X - 1).ptyTileType =
                        "Temporary"
                    End If
                    If MapArray(Y - 1, X).ptyTileType = "Empty" Then
                        MapArray(Y - 1, X).ptyTileType = "Temporary"
                    End If
                    If MapArray(Y - 1, X + 1).ptyTileType = "Empty" Then
                        MapArray(Y - 1, X + 1).ptyTileType =
                        "Temporary"
                    End If
                    If MapArray(Y, X + 1).ptyTileType = "Empty" Then
                        MapArray(Y, X + 1).ptyTileType = "Temporary"
                    End If
                    If MapArray(Y + 1, X + 1).ptyTileType = "Empty" Then
                        Then
                            MapArray(Y + 1, X + 1).ptyTileType =
                            "Temporary"
                        End If
                        If MapArray(Y + 1, X).ptyTileType = "Empty" Then
                            MapArray(Y + 1, X).ptyTileType = "Temporary"
                        End If
                End If
            End If
        End If
    End If
End If
```

```

Case 2 //Spread on 4 adjacent tiles
    If MapArray(Y, X - 1).ptyTileType = "Empty" Then
        MapArray(Y, X - 1).ptyTileType = "Temporary"
    End If
    If MapArray(Y - 1, X).ptyTileType = "Empty" Then
        MapArray(Y - 1, X).ptyTileType = "Temporary"
    End If
    If MapArray(Y, X + 1).ptyTileType = "Empty" Then
        MapArray(Y, X + 1).ptyTileType = "Temporary"
    End If
    If MapArray(Y + 1, X).ptyTileType = "Empty" Then
        MapArray(Y + 1, X).ptyTileType = "Temporary"
    End If
End Select
End If
Next X
Next Y
//Finally change the generated tiles' ptyTileType from "Temporary" to "Forest"
For X = 1 To 54
    For Y = 1 To 30
        If MapArray(Y, X).ptyTileType = "Temporary" Then
            MapArray(Y, X).ptyTileType = "Forest"
        End If
    Next Y
Next X

```

(PC2.10) Algorithm to create plains.

```

//All remaining empty tiles should now be converted to plains tiles
For X = 1 To 54 //Loop through the array identifying empty tiles and making them
plains
    For Y = 1 To 30
        If MapArray(Y, X).ptyTileType = "Empty" Then
            MapArray(Y, X).ptyTileType = "Plains"
        End If
    Next Y
Next X

```

(PC2.11) Algorithm to generate hills.

```
//Hills will be 2x2 in size and will be placed on top of plains tiles
While HillFrequency > 0 //HillFrequency is a variable that can be arbitrarily set a value
//While loop guarantees that the map will contain as many hills as the magnitude of
HillFrequency
For X = 1 To 53 // Cannot check column 54 as the array bounds would be
exceeded
For Y = 1 To 30
    If MapArray(Y, X).ptyTileType = "Plains" And MapArray(Y +
        1, X).ptyTileType = "Plains" And MapArray(Y, X +
        1).ptyTileType = "Plains" And MapArray(Y + 1, X +
        1).ptyTileType = "Plains" Then //Checking that a 2x2 area is
        clear to generate a hill
        Select Case RandomNum(1, 100)
            Case 1 //Generate a hill
                MapArray(Y, X).ptyTileType = "Plains"
                MapArray(Y+ 1, X).ptyTileType =
                    "Plains"
                MapArray(Y, X + 1).ptyTileType =
                    "Plains"
                MapArray(Y + 1, X + 1).ptyTileType =
                    "Plains"
            If HillFrequency = 0 Then //Exits the Y-For loop
                Exit For
            End If
        End Select
    End If
    Next Y
    If HillFrequency = 0 Then //Exits the X-For loop
        Exit For
    End If
    Next X
End While
```

(PC2.12) Algorithm to generate flora.

```
//Flora tiles act identical to plains tiles, except they have different textures
```

For X = 1 To 54

 For Y = 1 To 30

 Select Case RandomNum(1, FloraProbability) //Flora probability can be set an arbitrary value that will increase or decrease the frequency of flora tiles (the smaller the value of FloraProbability the greater the frequency of flora tiles)

 Case 1 //Generate flora

 If MapArray(Y, X).ptyTileType = "Plains" Then

 MapArray(Y, X).ptyTileType = "Flora"

 End If

 End Select

 Next Y

Next X

(PC2.13) Algorithm to print the map.

//Now the map has been generated, it must be printed to the in-game screen

Procedure PrintMap()

 For Y = 1 To 15 //prints the top left sector of the map initially

 For X = 1 To 27

 YCoord = (Y - 1) * 40 //each tile is 40x40 pixels in dimension

 XCoord = (X - 1) * 40

 DestinationRectangle = New Rectangle(XCoord, YCoord, 40, 40)

 Select Case MapArray(Y, X).ptyTileType

 Case "Plains"

 BitmapImage = New Bitmap(Source.PlainsTile.Image)

 InGameGraphics.DrawImage(BitmapImage,

 DestinationRectangle, SourceRectangle,

 GraphicsUnit.Pixel)

 Case "Flora"

 BitmapImage = New Bitmap(Source.FloraTile.Image)

 InGameGraphics.DrawImage(BitmapImage,

 DestinationRectangle, SourceRectangle,

 GraphicsUnit.Pixel)

 Case "Hill"

 BitmapImage = New Bitmap(Source.HillTile.Image)

```
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “Forest”

```
BitmapImage = New Bitmap(Source.ForestTile.Image)  
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “RiverSource”

```
BitmapImage = New  
Bitmap(Source.RiverSourceTile.Image)  
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “Stream”

```
BitmapImage = New Bitmap(Source.StreamTile.Image)  
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “River”

```
BitmapImage = New Bitmap(Source.RiverTile.Image)  
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “Coast”

```
BitmapImage = New Bitmap(Source.CoastTile.Image)  
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “Sea”

```
BitmapImage = New Bitmap(Source.SeaTile.Image)  
InGameGraphics.DrawImage(BitmapImage,  
DestinationRectangle, SourceRectangle,  
GraphicsUnit.Pixel)
```

Case “Mountain”

```

        BitmapImage = New
        Bitmap(Source.MountainTile.Image)
        InGameGraphics.DrawImage(BitmapImage,
        DestinationRectangle, SourceRectangle,
        GraphicsUnit.Pixel)

    End Select

    Next X

    Next Y

End Procedure

//BitmapImage is a bitmap variable that will take on an image from a picture box in the
FormImgRef form

//InGameGraphics is a graphics variable that is created as the graphics of a picture box
in the in-game form called pbMapDisplay

//DestinationRectangle and SourceRectangle are rectangles that copy and paste a
selected bitmap to the graphics variable

//GraphicsUnit.Pixel is a method of printing graphics that prints the graphics pixel-by-
pixel that I will use in the program

//DrawImage is an inbuilt function in Visual studio that takes the arguments:
DrawImage(Bitmap, DestinationRectangle, SourceRectangle, DrawingMethod)

```

Below is a table which describes how each algorithms form part of the solution and which success criteria and requirements they fulfil.

Pseudo-code algorithm	Description of how algorithm forms part of solution	Success criteria/requirements fulfilled	Justification of how algorithms fulfils criteria
PC2.1	If the map is to be randomly generated then numbers related to the map generation must be randomly generated.	R2.1	This allows the generated map to be random.
PC2.2	Creating a border prevents the array bounds from being exceeded, the empty tiles are required in checks throughout the solution.	Does not address the requirements directly, but makes it easier to code the program.	N/A
PC2.3	Randomly generates a river feature.	R2.1 R2.4	Randomly generates a river.
PC2.4	Randomly generates a lake.	R2.1 R2.6	Randomly generates a lake.

Pseudo-code algorithm	Description of how algorithm forms part of solution	Success criteria/ requirements fulfilled	Justification of how algorithms fulfils criteria
PC2.5	Randomly generates a sea.	R2.1 R2.6	Randomly generates a sea.
PC2.6	Generates a coastline around the pre-generated water feature.	R2.1 R2.6	Finishes the water feature generation by outlining it with a coastline.
PC2.7	Randomly generates mountain ranges.	R2.1 R2.5	Randomly creates mountain range anchor points.
PC2.8	Randomly spreads mountain ranges.	R2.1 R2.5	Randomly spreads mountain ranges from the anchor mountains.
PC2.9	Randomly generates forests.	R2.1 R2.3	Arbitrarily creates forest anchors points and randomly spreads the forests from these anchor points.
PC2.10	Fills all remaining empty tiles in with plains tiles, providing an area for the user to build the majority of the buildings.	R2.1 R2.2	Algorithm generates plains tiles for the user to build on on all remaining tiles.
PC2.11	Randomly places hill on top of plains tiles.	R2.1 R2.7	Algorithm generates hills on top of plains tiles.
PC2.12	Randomly replaces plains tiles with functionally identical flora tiles that have different textures.	R2.1 RO2.1	Algorithm changes plains tiles so that they will have different textures.

There will not be enough time in this iteration to implement RO2.2 and RO2.3. These features will have to be considered in future development phases.

Developing the solution (Coding)

The code for this iteration will be implemented in two separate modules:

ModuleMapGeneration and ModulePrintMap. The code will be written in the same order as the pseudocode (although the random number generator function appears towards the end of the ModuleMapGeneration module).

After the code has been programmed, the graphics for the tiles will be added into picture boxes in FormImgRef. These graphics were displayed previously in the usability features section of the description of the solution. Testing will take place throughout implementation of the code and the tests are shown after the coding section.

Code for the Map Generation Module:

Module ModuleMapGeneration

Public arrMapBlueprint(31, 55) **As** MapTile 'Map size 30, 54 and a border to prevent exceeding the array

Sub subGenerateMap()

'Some general purpose variables

Dim intArrYCoord, intArrXCoord, intCount, intTemp,
intGraphX, intGraphY, intProbability **As Integer**

'River related variables

Dim intMagRiverMeandering, intRiverX1, intRiverX2,
intRiverY **As Integer**

'Sea/Lake related variables

Dim intLakeHeight, intLakeWidth, intLakeRight, intLakeLeft,
intLakeLeftMax, intLakeRightWidth, intLakeLeftWidth,
intSeaHeight **As Integer**

Dim dblLakeShape, dblSeaShape **As Double**

'Mountain related variables

Dim intMountainProbability **As Integer**

'Forest related variables

Dim intForestProbability **As Integer**

'Hill related variables

Dim intHillFrequency **As Integer**

'Flora related variables

Dim intFloraProbability **As Integer**

'Fill the map with empty tiles and add a border

For intY = 0 **To** 31

For intX = 0 **To** 55

arrMapBlueprint(intY, intX) = **New** MapTile

If intY = 0 **Or** intY = 31 **Or** intX = 0 **Or** intX = 55 **Then**

arrMapBlueprint(intY, intX).ptyTileType =
"Border"

Else arrMapBlueprint(intY, intX).ptyTileType = "Empty"

```

        End If
    Next
    Next

'=====

'First generate the river source location, should be randomly
placed on the array somewhere in the co-ordinates: 1<=y<=2,
8<=x<=47
intArrYCoord = FuncRandomNumberGen(2, 3)
intArrXCoord = FuncRandomNumberGen(8, 47)
arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"RiverSource"

'=====

'Next generate a stream leading from the source down to the start
of the river (should extend 2-4 tiles and be 1 tile wide)
intCount = intArrYCoord
For intArrYCoord = intCount + 1 To
FuncRandomNumberGen(intArrYCoord + 2, intArrYCoord + 4)
    arrMapBlueprint(intArrYCoord,
    intArrXCoord).ptyTileType = "Stream"
Next

'=====

'Next generate the river beneath the stream leading down to the
bottom of the screen (river should be 2 tiles wide)
'Find starting location of river
intRiverX1 = intArrXCoord
intRiverY = intArrYCoord
'The river should bend. The bendiness of the river depends on
how often the river is given an opportunity to bend. This trait will
be referred to as the magnitude of river meandering
'Decide the magnitude of river meandering - will be saved in the
variable intTemp (should be between 4 and 8)
intMagRiverMeandering = FuncRandomNumberGen(4, 8)

```

```

'Decide which side of the stream to expand (left or right, can be
determined with a random number used in a select case)
Select Case FuncRandomNumberGen(1, 2)
Case 1
    intRiverX2 = intRiverX1 - 1
Case 2
    intRiverX2 = intRiverX1 + 1
End Select
intCount = intRiverY
While intRiverY < 31
    For intRiverY = intCount To intRiverY +
        intMagRiverMeandering
        arrMapBlueprint(intRiverY,
        intRiverX1).ptyTileType = "River"
        arrMapBlueprint(intRiverY,
        intRiverX2).ptyTileType = "River"
    'Ensure that intRiverY does not exceed bounds of
    arrMapBlueprint
    If intRiverY = 30 Then
        Exit While
    End If
Next
'Decide which way to meander
Select Case FuncRandomNumberGen(1, 2)
Case 1
    intRiverX1 = intRiverX1 - 1
    intRiverX2 = intRiverX2 - 1
Case 2
    intRiverX1 = intRiverX1 + 1
    intRiverX2 = intRiverX2 + 1
End Select
intCount = intRiverY
End While
'-----

```

'Now that the river has been generated, the sea must now be
generated (which will always appear on the bottom of the screen)
'First the dimensions of the sea must be decided: width followed
by height (how far inland it reaches)

'Width: the sea can either take up all the columns of the bottom few rows, or several - to appear as a lake: the chances of the sea...
 ...taking up the whole of the bottom of the screen should be lower than the probability of it appearing as a lake
If FuncRandomNumberGen(1, 4) < 4 **Then**
 3/4 of the time it will be a lake
 'the lake must be generated around the bottom of the river,
 thus the X-location of the left and right sides of the river must be determined
 'Determining the location of the left and right side
 If arrMapBlueprint(intRiverY, intRiverX1 + 1).ptyTileType = "Empty" **Then**
 'intRiverX1 is on the right and must be swapped
 intTemp = intRiverX2
 intRiverX2 = intRiverX1
 intRiverX1 = intTemp
 'Now intRiverX1 will always be on the left and intX2 will always be on the right
 End If
 'The bottom of the screen will always be where the lake width is at its maximum and must be decided now (size can vary between 15 and 27)
 'The height of the lake should vary between 4 and 10
 intLakeWidth = FuncRandomNumberGen(15, 27)
 intLakeHeight = FuncRandomNumberGen(4, 10)
 'The lake outline will be drawn using an arc function to find the array co-ordinate values of the lake outline
 'Find out how far the lake will stretch either side of the river, ensure the lake size will not be reduced by edge of form

intLakeLeftMax = intLakeWidth - 2

If intRiverX1 - intLakeWidth < 1 **Then** 'Checking left Bounds
 intLakeLeftMax = intRiverX1 - 1
End If

intLakeLeft = intRiverX1 - FuncRandomNumberGen(2, intLakeLeftMax)

```

intLakeRight = intRiverX2 + (intLakeWidth - (intRiverX1
- intLakeLeft))
intLakeLeftWidth = intRiverX1 - intLakeLeft
intLakeRightWidth = intLakeRight - intRiverX2

```

'The arc equation is as follows: $Y =$
 $(intlakeHeight^{(dblLakeShape)}) - ((X-intlakeRight)/$
 $(intRightWidth/intlakeHeight))^{(dblLakeShape)})^{(1/$
 $(dblLakeShape))}$

'The equation for the left hand side is identical, replace
intlakeRight with intlakeLeft and intRightWidth with -
intLeftWidth

'Where Y-intercept is intlakeHeight and X-intercept is
intlakeHeight * intlakeRightWidth

'Where X = 0 is the river, and Y = 0 is bottom of the
screen i.e X = intRiverX1/intRiverX2, Y = 43

'Create outline for lake left hand side of river:

'First generate a random number between 0.8 and 3.2 for
the intlakeShape variable

'Because of how this function graphs, the left hand side of
the graph only allows decimal values that are divisible by
4 when multiplied by 10 (i.e 1.0 is not allowed as 1.0 * 10
= 10 and 10 / 3 <> integer)

dblLakeShape = (4 * FuncRandomNumberGen(2, 8)) / 10

For intGraphX = -intLakeLeftWidth To 0

```

intGraphY = CInt((intLakeHeight ^ dblLakeShape)
- ((intGraphX / (-intLakeLeftWidth /
intLakeHeight)) ^ dblLakeShape)) ^ (1 /
dblLakeShape)

```

intArrXCoord = intRiverX1 + intGraphX

intArrYCoord = 30 - intGraphY

arrMapBlueprint(intArrYCoord,

intArrXCoord).ptyTileType = "Sea"

[Next](#)

'Create Outline for lake right hand side of the river:

'First generate a random number between 0.8 and 3.2 for
the intlakeShape variable

'Because of how this function graphs, the right hand side

of the graph cannot allow decimal values that are odd when multiplied by 10 (i.e 0.9 is not allowed as $0.9 * 10 = 9$)

```
dblLakeShape = (2 * FuncRandomNumberGen(4, 16)) /  
10
```

```
For intGraphX = 0 To intLakeRightWidth
```

```
    intGraphY = CInt((intLakeHeight ^ dblLakeShape)  
    - ((intGraphX / (intLakeRightWidth /  
    intLakeHeight)) ^ dblLakeShape)) ^ (1 /  
    dblLakeShape)
```

'Moving the co-ordinates into the array bounds at
the correct locations

```
intArrXCoord = intRiverX2 + intGraphX
```

```
intArrYCoord = 30 - intGraphY
```

```
arrMapBlueprint(intArrYCoord,
```

```
intArrXCoord).ptyTileType = "Sea"
```

Next

'Now fill in the lake by locating the top of the lake in each
column and filling down from there

```
For intArrXCoord = intLakeLeft To intLakeRight
```

```
    For intArrYCoord = 30 - intLakeHeight To 30
```

```
        If arrMapBlueprint(intArrYCoord,  
        intArrXCoord).ptyTileType = "Sea" Then
```

```
            intCount = intArrYCoord
```

```
            For intGraphY = intCount To 30
```

```
                arrMapBlueprint(intGraphY,  
                intArrXCoord).ptyTileType =  
                "Sea"
```

Next

End If

Next

Next

'The lake is now generated

Else

'1/4 of the time it will be a sea that takes up the whole of
the bottom of the screen

'The sea shape will be produced by graphing a $Y = \text{Sin}(X)$
or $Y = \text{Cos}(X)$ function. The shape of the $Y = \text{Sin}(X)$ or Y
= $\text{Cos}(X)$ functions can be varied using a variable.

```

'The equation of the sea coast will either be Y =
dblSeaShape*Math.Sin(X/dblSeaShape) or Y =
dblSeaShape*Math.Cos(X/dblSeaShape)
'The shape of the sea should also have a chance to change,
every trig-wavelength there will be a 2/3 chance that
dblSeaShape will change for the next iteration
'First generate intSeaHeight and dblSeaShape
'inSeaHeight is where the middle of the Y = Sin(X) or Y =
Cos(X) functions will be centred upon:
intSeaHeight = FuncRandomNumberGen(2, 4)
'intSeaShape will vary between 0.6 and 1.5:
dblSeaShape = FuncRandomNumberGen(6, 15) / 10

'Decide which trig-function will begin
Select Case FuncRandomNumberGen(1, 2)
Case 1 'Y = dblSeaShape*Math.Sin(X/(2*dblSeaShape))
    For intGraphX = 1 To 54
        intGraphY = CInt(dblSeaShape *
        Math.Sin(intGraphX / (2 * dblSeaShape)))
        intArrXCoord = intGraphX
        intArrYCoord = intGraphY + (30 -
        intSeaHeight)
        arrMapBlueprint(intArrYCoord,
        intArrXCoord).ptyTileType = "Sea"
        If intGraphY = 1 Then
            Select Case intTemp =
            FuncRandomNumberGen(1, 3)
            Case intTemp < 3
                dblSeaShape =
                FuncRandomNumberGen(6, 15) / 10
            End Select
        End If
    Next
Case 2 'Y = dblSeaShape*Math.Cos(X/dblSeaShape)
    For intGraphX = 1 To 54
        intGraphY = CInt(dblSeaShape *
        Math.Cos(intGraphX / (2 * dblSeaShape)))
        intArrXCoord = intGraphX
        intArrYCoord = intGraphY + (30 -
        intSeaHeight)

```

```

arrMapBlueprint(intArrYCoord,
intArrXCoord).ptyTileType = "Sea"
If intGraphY = -1 Then
    Select Case intTemp =
        FuncRandomNumberGen(1, 3)
        Case intTemp < 3
            dblSeaShape =
                FuncRandomNumberGen(6, 15) / 10
        End Select
    End If
    Next
End Select

```

'Now fill in the sea by locating the top of the sea in each column and filling down from there

```

For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        If arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Sea" Then
            intCount = intArrYCoord
            For intGraphY = intCount To 30
                arrMapBlueprint(intGraphY,
                    intArrXCoord).ptyTileType = "Sea"
            Next
        End If
        Next
    Next
End If

```

'The sea is now generated.

'By this point either a sea or a lake has been generated.

```
'=====
```

'Next the sea/lake must be lined with coastline tiles, this will be implemented by first looping down the columns and adding a coast tile above the first sea tile...

```

For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        If arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Sea" Then

```

```

        If arrMapBlueprint(intArrYCoord - 1,
            intArrXCoord).ptyTileType = "River" Then
            Exit For
        End If
        arrMapBlueprint(intArrYCoord - 1,
            intArrXCoord).ptyTileType = "Coast"
        Exit For
    End If
    Next
    Next
    '...and secondly by looping through the rows and adding a coast tile to
    the edge sea tiles
    For intArrYCoord = 1 To 30
        For intArrXCoord = 1 To 54
            If arrMapBlueprint(intArrYCoord,
                intArrXCoord).ptyTileType = "Sea" And
                arrMapBlueprint(intArrYCoord, intArrXCoord -
                1).ptyTileType = "Empty" Then
                    arrMapBlueprint(intArrYCoord, intArrXCoord -
                    1).ptyTileType = "Coast"
            End If
        Next
        For intArrXCoord = 1 To 54
            If arrMapBlueprint(intArrYCoord,
                intArrXCoord).ptyTileType = "Sea" And
                arrMapBlueprint(intArrYCoord, intArrXCoord +
                1).ptyTileType = "Empty" Then
                    arrMapBlueprint(intArrYCoord, intArrXCoord +
                    1).ptyTileType = "Coast"
            End If
        Next
    Next

```

'=====

'Next the mountains will be generated. These will only ever appear along the top of the map, and will never extent down further than 4 tiles from the top. There will always...

'...be mountains around the river source tile, but they will also be generated in clumps. The clump size can be large and small, and the size will be determined by probability.

'First generate mountains around river source tile. Accomplished by locating river tile and filling in mountain tiles around it in preset pattern

'Find river source tile

For intArrYCoord = 2 To 4

 For intArrXCoord = 8 To 48

 If arrMapBlueprint(intArrYCoord,
 intArrXCoord).ptyTileType = "RiverSource" Then

 Exit For

 End If

 Next

 If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
 "RiverSource" Then

 Exit For

 End If

Next

arrMapBlueprint(intArrYCoord + 1, intArrXCoord - 1).ptyTileType =
 "Mountain"

arrMapBlueprint(intArrYCoord, intArrXCoord - 1).ptyTileType =
 "Mountain"

arrMapBlueprint(intArrYCoord - 1, intArrXCoord - 1).ptyTileType =
 "Mountain"

arrMapBlueprint(intArrYCoord - 1, intArrXCoord).ptyTileType =
 "Mountain"

arrMapBlueprint(intArrYCoord - 1, intArrXCoord + 1).ptyTileType =
 "Mountain"

arrMapBlueprint(intArrYCoord, intArrXCoord + 1).ptyTileType =
 "Mountain"

arrMapBlueprint(intArrYCoord + 1, intArrXCoord + 1).ptyTileType =
 "Mountain"

'Now loop along the top of the array and place single mountain tiles randomly to act as anchors for the mountain ranges. Probability of anchor mountains being spawned can be decided...

'by comparing the values of two numbers, the frequency of anchor mountains being spawned can be varied by changing the value of intMountainProbability

For intArrXCoord = 1 To 54

 intMountainProbability = FuncRandomNumberGen(1, 3)

```

If intMountainProbability * FuncRandomNumberGen(1, 8) >
FuncRandomNumberGen(12, 16) Then
    arrMapBlueprint(1, intArrXCoord).ptyTileType =
    "Mountain"
End If

Next
subGenerateMountains(4, 1)
subGenerateMountains(4, 2)
subGenerateMountains(3, 3)
subGenerateMountains(2, 4)
'Mountains now generated

```

'=====

'Now to generate forests:
'All empty tiles in the top three rows should be converted to forest
For intArrYCoord = 1 To 3

```

    For intArrXCoord = 1 To 54
        If arrMapBlueprint(intArrYCoord,
        intArrXCoord).ptyTileType = "Empty" Then
            arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Forest"
        End If

```

Next

Next
'Now to add some detail to the top forest by converting some of the
empty tiles below the 3rd row forest into more forest
For intArrXCoord = 1 To 54

```

    If arrMapBlueprint(3, intArrXCoord).ptyTileType = "Forest"
    Then
        Select Case FuncRandomNumberGen(1, 2)
        Case 1 'Add a forest tile
            arrMapBlueprint(4, intArrXCoord).ptyTileType =
            "Forest"
        End Select

```

End If

Next

'The forests will be generated in a similar fashion to the mountains
(creating anchors and generating around them)

```

'First generate forest anchors on the edge columns of the map on empty
tiles
For intArrXCoord = 1 To 54
    For intArrYCoord = 4 To 30
        intForestProbability = FuncRandomNumberGen(1, 3)
        If intForestProbability * FuncRandomNumberGen(1, 8) >
        FuncRandomNumberGen(12, 16) And
        arrMapBlueprint(intArrYCoord,
        intArrXCoord).ptyTileType = "Empty" Then
            arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Temporary"
        End If
        'Skip middle columns
        If intArrXCoord = 2 And intArrYCoord = 30 Then
            intArrXCoord = 53
        End If
        Next
    Next
    'Now generate forest anchors on the bottom 2 rows of the map on empty
    tiles (won't be generated if a sea is present)
    For intArrYCoord = 29 To 30
        For intArrXCoord = 1 To 54
            If FuncRandomNumberGen(1, 3) > 1 And
            arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord,
                intArrXCoord).ptyTileType = "Temporary"
            End If
        Next
    Next
    'Next allow the forests to spread from all current forest tiles into empty
    tiles in all directions
    For intArrYCoord = 3 To 30
        For intArrXCoord = 1 To 54
            If arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Temporary" Then
                intProbability = FuncRandomNumberGen(1, 3)
                Select Case intProbability
                Case 1 'Spread on all 8 tiles around the anchor

```

```

If arrMapBlueprint(intArrYCoord + 1,
intArrXCoord - 1).ptyTileType = "Empty"
Then
    arrMapBlueprint(intArrYCoord + 1,
intArrXCoord - 1).ptyTileType =
    "Temporary"
End If
If arrMapBlueprint(intArrYCoord,
intArrXCoord - 1).ptyTileType = "Empty"
Then
    arrMapBlueprint(intArrYCoord,
intArrXCoord - 1).ptyTileType =
    "Temporary"
End If
If arrMapBlueprint(intArrYCoord - 1,
intArrXCoord - 1).ptyTileType = "Empty"
Then
    arrMapBlueprint(intArrYCoord - 1,
intArrXCoord - 1).ptyTileType =
    "Temporary"
End If
If arrMapBlueprint(intArrYCoord - 1,
intArrXCoord).ptyTileType = "Empty" Then
    arrMapBlueprint(intArrYCoord - 1,
intArrXCoord).ptyTileType =
    "Temporary"
End If
If arrMapBlueprint(intArrYCoord - 1,
intArrXCoord + 1).ptyTileType = "Empty"
Then
    arrMapBlueprint(intArrYCoord - 1,
intArrXCoord + 1).ptyTileType =
    "Temporary"
End If
If arrMapBlueprint(intArrYCoord,
intArrXCoord + 1).ptyTileType = "Empty"
Then
    arrMapBlueprint(intArrYCoord,
intArrXCoord + 1).ptyTileType =
    "Temporary"
End If

```

```
If arrMapBlueprint(intArrYCoord + 1,  
intArrXCoord + 1).ptyTileType = "Empty"  
Then
```

```
    arrMapBlueprint(intArrYCoord + 1,  
    intArrXCoord + 1).ptyTileType =  
    "Temporary"
```

```
End If
```

```
If arrMapBlueprint(intArrYCoord + 1,  
intArrXCoord).ptyTileType = "Empty" Then  
    arrMapBlueprint(intArrYCoord + 1,  
    intArrXCoord).ptyTileType =  
    "Temporary"
```

```
End If
```

Case 2 'Spread on 4 adjacent tiles

```
If arrMapBlueprint(intArrYCoord,  
intArrXCoord - 1).ptyTileType = "Empty"  
Then
```

```
    arrMapBlueprint(intArrYCoord,  
    intArrXCoord - 1).ptyTileType =  
    "Temporary"
```

```
End If
```

```
If arrMapBlueprint(intArrYCoord - 1,  
intArrXCoord).ptyTileType = "Empty" Then  
    arrMapBlueprint(intArrYCoord - 1,  
    intArrXCoord).ptyTileType =  
    "Temporary"
```

```
End If
```

```
If arrMapBlueprint(intArrYCoord,  
intArrXCoord + 1).ptyTileType = "Empty"  
Then
```

```
    arrMapBlueprint(intArrYCoord,  
    intArrXCoord + 1).ptyTileType =  
    "Temporary"
```

```
End If
```

```
If arrMapBlueprint(intArrYCoord + 1,  
intArrXCoord).ptyTileType = "Empty" Then  
    arrMapBlueprint(intArrYCoord + 1,  
    intArrXCoord).ptyTileType =  
    "Temporary"
```

```
End If
```

```
End Select
```

```
        End If
    Next
    Next
    'Now change the placeholder forest tiles into true forest tiles
    For intArrXCoord = 1 To 54
        For intArrYCoord = 1 To 30
            If arrMapBlueprint(intArrYCoord,
                intArrXCoord).ptyTileType = "Temporary" Then
                arrMapBlueprint(intArrYCoord,
                intArrXCoord).ptyTileType = "Forest"
            End If
        Next
    Next
```

'=====

```
'Now generate plains tiles by reapplying all remaining empty tiles with
plains tiles
```

```
For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        If arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Empty" Then
            arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Plains"
        End If
    Next
Next
```

'=====

```
'Penultimately generate hill tiles in 2x2 clumps
intHillFrequency = 10
```

```
'Loop through array locating plains tiles with space to generate a 2x2
hill
While intHillFrequency > 0
    For intArrXCoord = 1 To 53
        For intArrYCoord = 1 To 30
```

```

If arrMapBlueprint(intArrYCoord,
intArrXCoord).ptyTileType = "Plains" And
arrMapBlueprint(intArrYCoord + 1,
intArrXCoord).ptyTileType = "Plains" And
arrMapBlueprint(intArrYCoord, intArrXCoord +
1).ptyTileType = "Plains" And
arrMapBlueprint(intArrYCoord + 1, intArrXCoord +
1).ptyTileType = "Plains" Then
    Select Case FuncRandomNumberGen(1,
100)
        Case 1 'Generate a hill
            arrMapBlueprint(intArrYCoord,
intArrXCoord).ptyTileType = "Hill"
            arrMapBlueprint(intArrYCoord + 1,
intArrXCoord).ptyTileType = "Hill"
            arrMapBlueprint(intArrYCoord,
intArrXCoord + 1).ptyTileType =
"Hill"
            arrMapBlueprint(intArrYCoord + 1,
intArrXCoord + 1).ptyTileType =
"Hill"
            intHillFrequency = intHillFrequency
- 1
            If intHillFrequency = 0 Then
                Exit For
            End If
        End Select
    End If
    Next
    If intHillFrequency = 0 Then
        Exit For
    End If
    Next
End While

```

'=====

'Finally generate flora tiles which will randomly replace plains tiles to
vary the graphics more
intFloraProbability = 10

```

For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        Select Case FuncRandomNumberGen(1,
            intFloraProbability)
            Case 1
                If arrMapBlueprint(intArrYCoord,
                    intArrXCoord).ptyTileType = "Plains" Then
                    arrMapBlueprint(intArrYCoord,
                        intArrXCoord).ptyTileType = "Flora"
                End If
            End Select
        Next
    Next
End Sub

```

'Random number generator function

```

Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal
intUpperBound As Integer)
    Dim intRandomNumber = 0
    Randomize()
    intRandomNumber = CInt(Math.Floor((intUpperBound -
        intLowerBound + 1) * Rnd())) + intLowerBound 'Upper and lower
    bounds set ranges (inclusively)
    Return intRandomNumber
End Function

```

'Mountain range generator procedure

```

Sub subGenerateMountains(intProbability, intArrYCoord)
    'intTemp used to store input of intProbability
    Dim intTemp, intSelect As Integer
    'Check row above
    intTemp = intProbability
    For intArrXCoord = 1 To 54
        If arrMapBlueprint(intArrYCoord - 1,
            intArrXCoord).ptyTileType = "Mountain" And
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType <>
            "RiverSource" And arrMapBlueprint(intArrYCoord,
                intArrXCoord).ptyTileType <> "Stream" Then
            Select Case intSelect = FuncRandomNumberGen(1,
                intProbability)

```

```

Case 1 'Do nothing
    intProbability = intTemp
Case 1 < intSelect 'Generate a mountain
    arrMapBlueprint(intArrYCoord,
    intArrXCoord).ptyTileType = "Mountain"
    intProbability = intProbability - 1
End Select
End If
Next
'First loop left to right...
intProbability = intTemp
For intArrXCoord = 1 To 54
    If arrMapBlueprint(intArrYCoord, intArrXCoord -
    1).ptyTileType = "Mountain" And
        arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType <>
        "RiverSource" And arrMapBlueprint(intArrYCoord,
        intArrXCoord).ptyTileType <> "Stream" Then
        Select Case intSelect = FuncRandomNumberGen(1,
        intProbability)
        Case 1 'Do nothing
            intProbability = intTemp
        Case 1 < intSelect 'Generate a mountain
            arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType = "Mountain"
            intProbability = intProbability - 1
        End Select
    End If
    Next
    '...second loop right to left:
    intProbability = intTemp
    For intArrXCoord = 54 To 1 Step -1
        If arrMapBlueprint(intArrYCoord, intArrXCoord +
        1).ptyTileType = "Mountain" And
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType <>
            "RiverSource" And arrMapBlueprint(intArrYCoord,
            intArrXCoord).ptyTileType <> "Stream" Then
            Select Case intSelect = FuncRandomNumberGen(1,
            intProbability)
            Case 1 'Do nothing
                intProbability = intTemp
            Case 1 < intSelect 'Generate a mountain

```

```

        arrMapBlueprint(intArrYCoord,
        intArrXCoord).ptyTileType = "Mountain"
        intProbability = intProbability - 1
    End Select
End If
Next
End Sub
End Module

```

Code for the Print Map Module:

```

Imports System.Drawing
Module ModulePrintMap

Public gfxInGameView As System.Drawing.Graphics =
FormInGame.pbMapDisplay.CreateGraphics()
Public rectGraphicsSource As New Rectangle(0, 0, 40, 40)
Public rectGraphicsDestination As Rectangle
Public bmpImage As Bitmap

Sub subPrintMap()
    Dim intYCoord, intXCoord As Integer

    For intY = 1 To 15
        For intX = 1 To 27
            intYCoord = (intY - 16) * 40
            intXCoord = (intX - 1) * 40
            rectGraphicsDestination = New Rectangle(intXCoord,
            intYCoord, 40, 40)
            Select Case arrMapBlueprint(intY, intX).ptyTileType
                Case "Plains" 'Generate a plains graphic at the location
                    bmpImage = New
                    Bitmap(FormImgRef.pbPlainsTile.Image)
                    gfxInGameView.DrawImage(bmpImage,
                    rectGraphicsDestination, rectGraphicsSource,
                    GraphicsUnit.Pixel)
                Case "Flora" 'Generate a flora graphic at the location
                    bmpImage = New
                    Bitmap(FormImgRef.pbFloraTile.Image)
            End Select
        Next
    Next
End Sub

```

```
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "Hill" 'Generate a hill graphic at the location

```
bmpImage = New  
Bitmap(FormImgRef.pbHillTLTile.Image)  
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "Forest" 'Generate a forest graphic at the location

```
bmpImage = New  
Bitmap(FormImgRef.pbForestCentreTile.Image)  
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "RiverSource" 'Generate a riverSource graphic at the location

```
bmpImage = New  
Bitmap(FormImgRef.pbRiverSourceTile.Image)  
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "Stream" 'Generate a stream graphic at the location

```
bmpImage = New  
Bitmap(FormImgRef.pbStreamTile.Image)  
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "River" 'Generate a river graphic at the location

```
bmpImage = New  
Bitmap(FormImgRef.pbRiverTile.Image)  
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "Coast" 'Generate a coast graphic at the location

```
bmpImage = New  
Bitmap(FormImgRef.pbCoastTile.Image)  
gfxInGameView.DrawImage(bmpImage,  
rectGraphicsDestination, rectGraphicsSource,  
GraphicsUnit.Pixel)
```

Case "Sea" 'Generate a sea graphic at the location

```

        bmpImage = New
        Bitmap(FormImgRef.pbSeaTile.Image)
        gfxInGameView.DrawImage(bmpImage,
        rectGraphicsDestination, rectGraphicsSource,
        GraphicsUnit.Pixel)
    Case "Mountain" 'Generate a mountain graphic at the
        location
        bmpImage = New
        Bitmap(FormImgRef.pbMountainTile.Image)
        gfxInGameView.DrawImage(bmpImage,
        rectGraphicsDestination, rectGraphicsSource,
        GraphicsUnit.Pixel)
    End Select
    Next
    Next
    End Sub
End Module

```

Code for the ClassMapTile class:

```

Public Class MapTile
    Private strTileType As String
    Private strBuildingType As String

    Public Property ptyTileType As String
        Get
            Return strTileType
        End Get
        Set(value As String)
            strTileType = value
        End Set
    End Property

    Public Property ptyBuildingType As String
        Get
            Return strBuildingType
        End Get
        Set(value As String)
            strBuildingType = value
        End Set
    End Property

```

End Class

Developing the solution (Testing to inform development):

It has become apparent after testing DD2.1 that most of the during development tests' success depends wholly upon the random number generator. As seen in the table below and in the evidence section for DD2.1, the random number generator has been demonstrated to work correctly; thus the other tests that rely solely upon the validity of the random number generator have been made redundant. These redundancies are shown in the "Test name, reason for test" column and do not need to be tested. Some other tests have been made cancelled as I have deemed them too trivial to be worth testing. These cancellations are also shown in the "Test name, reason for test" column.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
DD2.1	Testing random number generator returns random values within the bounds passed to the function.	Lower Bound = 1 Other value = 6 Upper Bound = 10 All possible invalid values will be tested.	1, 6 and 10 are all returned upon separate runs of the function. All other values will fail.	Yes. All other values outside of bounds returned stack overflow.
DD2.2	Cancelled	—	—	—
DD2.3	Cancelled	—	—	—
DD2.4	Redundant	—	—	—
DD2.5	Cancelled	—	—	—

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
DD2.6	Cancelled	—	—	—
DD2.7	Cancelled	—	—	—
DD2.8	Largely redundant and cancelled	—	—	—
DD2.9	Cancelled	—	—	—
DD2.10	Test that the shape of the lake outline is randomly generated and the shape of the lake is different either side of the river.	No user input. (Repeat 3 times) (dblLakeShape = X) for left hand side of river = $0.8 \leq X \leq 3.2$ step 0.4 (dblLakeShape = X) for right hand side of river = $0.8 \leq X \leq 3.2$ step 0.2	Shape of lake outline corresponds to shape of predicted graph in graphing program desmos.	Yes.
DD2.11	Test that water feature generation does not exceed bounds of array.	No user input. Extreme values for variables and co-ordinates (Y, X) of relevant tile types (left hand side): <ul style="list-style-type: none">• RiverSource tile location: (2, 8)• Bottom Stream tile location: (4, 8)• intMagRiverMeandering = 4• RiverX2 location: (4, 7)• River will always meander left• intLakeWidth = 27 Extreme values for variables and co-ordinates (Y, X) of relevant tile types (left hand side): <ul style="list-style-type: none">• RiverSource tile location: (2, 48).• Bottom Stream tile location: (4, 48)• intMagRiverMeandering = 4• RiverX2 location: (4, 49)• River will always meander right• intLakeWidth = 27	In left hand case: intArrXCoord ≥ 1 at all times In right hand case intArrXCoord ≤ 54 at all times	No. River generated too far left for the left extreme and the array bounds were exceeded for the right hand side. See remedial action for solutions.
DD2.12	Testing that sea shape is correct.	No user input. (Repeat 3 times) $0.6 \leq \text{dblSeaShape} \leq 1.5$ step 0.1	Shape of sea outline corresponds to shape of predicted graph in graphing program desmos.	Yes.
DD2.13	Cancelled	—	—	—
DD2.14	Cancelled	—	—	—
DD2.15	Cancelled	—	—	—
DD2.16	Redundant	—	—	—
DD2.17	Cancelled	—	—	—
DD2.18	Cancelled	—	—	—

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
DD2.19	Cancelled	—	—	—
DD2.20	Cancelled	—	—	—
DD2.21	Redundant	—	—	—
DD2.22	Cancelled	—	—	—

Evidence of testing:

DD2.1

To test the random number generator, I wrote the function as a recursive function so that the stopping condition is when the random number is equal to a specified value (1, 6 or 10). The code was as follows:

Module Module1

```

Sub Main()
    Dim Result As Boolean,
        Result = FuncRandomNumberGen(1, 10)
        Console.WriteLine(Result)
        Console.Read()
End Sub

Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal
intUpperBound As Integer)
    Dim intRandomNumber = 0
    Randomize()
    intRandomNumber = CInt(Math.Floor((intUpperBound -
intLowerBound + 1) * Rnd())) + intLowerBound
    If intRandomNumber <> Value Then
        FuncRandomNumberGen(1, 10)
    End If
    Return True
End Function

```

End Module

Where Value was replaced with 1, 6, 10 and 11 (invalid value) for the tests.

A break point was added by the End Function statement. If this line is executed, it means the recursive function has reached the stopping condition (i.e the specified value was generated) and intRandomNumber = the specified value. If a stack overflow is encountered, the recursive function iterates infinitely as the specified value is never generated (i.e is outside the specified bounds).

When value = 1 the function generated 1 at some point:

```

Module Module1
Sub Main()
    Dim Result As Boolean
    Result = FuncRandomNumberGen(1, 10)
    Console.WriteLine(Result)
    Console.Read()
End Sub

Public Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal intUpperBound As Integer)
    Dim intRandomNumber = 0
    Randomize()
    intRandomNumber = CInt(Math.Floor((intUpperBound - intLowerBound + 1) * Rnd())) + intLowerBound
    If intRandomNumber > 1 Then
        FuncRandomNumberGen(1, 10)
    End If
    Return True
End Function
End Module

```

Name	Value
intRandomNumber	1
intLowerBound	1
intUpperBound	10

When value = 6 the function generated 6 at some point:

```

Module Module1
Sub Main()
    Dim Result As Boolean
    Result = FuncRandomNumberGen(1, 10)
    Console.WriteLine(Result)
    Console.Read()
End Sub

Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal intUpperBound As Integer)
    Dim intRandomNumber = 0
    Randomize()
    intRandomNumber = CInt(Math.Floor((intUpperBound - intLowerBound + 1) * Rnd())) + intLowerBound
    If intRandomNumber > 6 Then
        FuncRandomNumberGen(1, 10)
    End If
    Return True
End Function
End Module

```

Name	Value
intRandomNumber	6
intLowerBound	1
intUpperBound	10

When value = 10 the function generated 10 at some point:

```
Module Module1
    Sub Main()
        Dim Result As Boolean

        Result = FuncRandomNumberGen(1, 10)
        Console.WriteLine(Result)
        Console.Read()
    End Sub

    Public Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal intUpperBound As Integer)
        Dim intRandomNumber = 0
        Randomize()
        intRandomNumber = CInt(Math.Floor((intUpperBound - intLowerBound + 1) * Rnd())) + intLowerBound
        If intRandomNumber > 10 Then
            FuncRandomNumberGen(1, 10)
        End If
        Return True
    End Function
End Module
```

Watch 1

Name	Value
intRandomNumber	10
intLowerBound	1
intUpperBound	10

When the base case of recursion was any value outside of the bounds being generated a stack overflow was returned as expected.

```
Module Module1
    Sub Main()
        Dim Result As Boolean

        Result = FuncRandomNumberGen(1, 10)
        Console.WriteLine(Result)
        Console.Read()
    End Sub

    Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal intUpperBound As Integer)
        Dim intRandomNumber = 0
        Randomize()
        intRandomNumber = CInt(Math.Floor((intUpperBound - intLowerBound + 1) * Rnd())) + intLowerBound
        If 1 < intRandomNumber OR intRandomNumber < 10 Then
            FuncRandomNumberGen(1, 10)
        End If
        Return True
    End Function
End Module
```

Watch 1

Name	Value
intRandomNumber	0
intUpperBound	10
intLowerBound	1

StackOverflowException was unhandled

An unhandled exception of type 'System.StackOverflowException' occurred in mscorelib.dll

Troubleshooting tips:

Make sure you do not have an infinite loop or infinite recursion.
Get general help for this exception.

Search for more Help Online...

Exception settings:

Break when this exception type is thrown

Actions:

[View Detail...](#)
[Copy exception detail to the clipboard](#)
[Open exception settings](#)

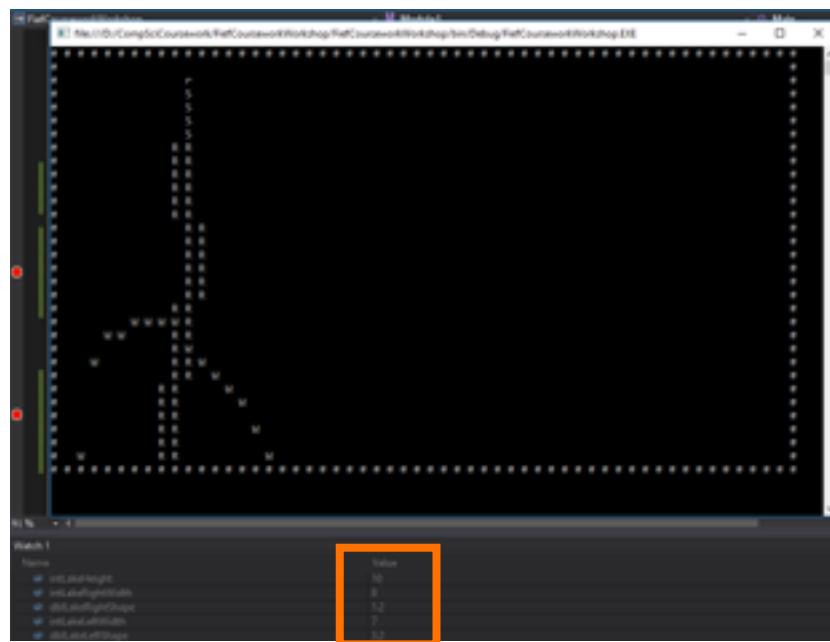
DD2.10

For this test I ran the code and compared the shape of the generated lake to the shape of the corresponding graph on desmos with identical variable values. I repeated 3 times to reduce uncertainty.

In the console '#' represents border tiles; ' ' represents an empty tile 'r' represents the river source; 's' represents stream tiles; 'R' represents river tiles and 'W' represents a sea/lake tile.

First test:

Console result:

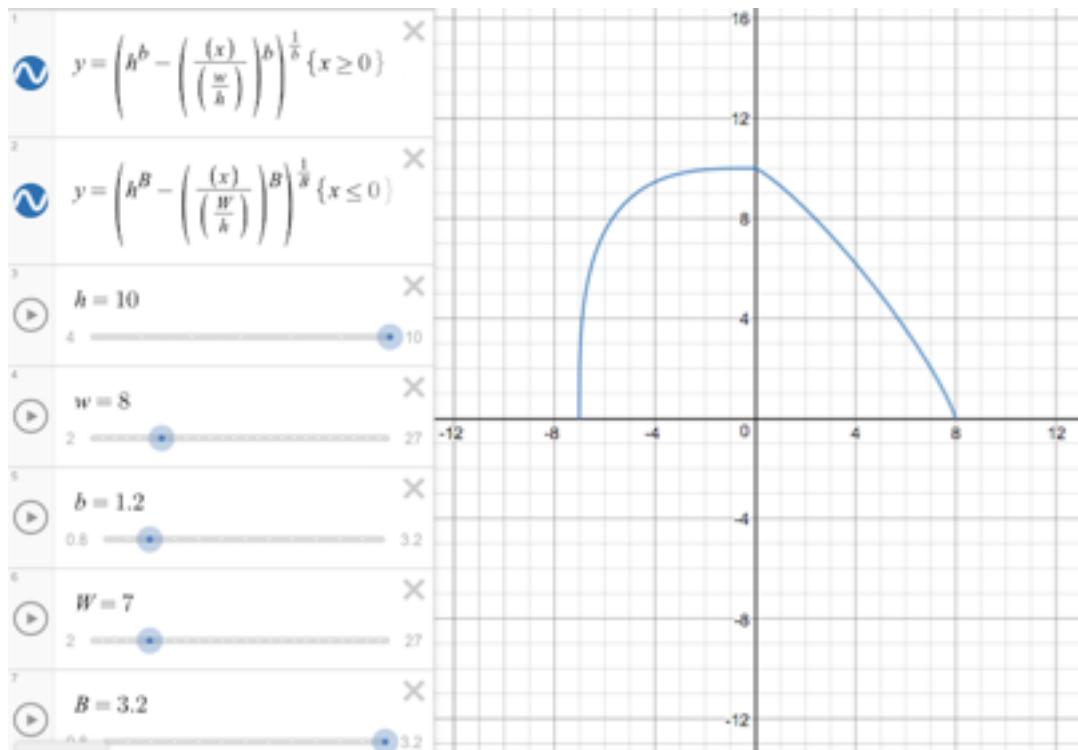


As seen in Watch 1 the values for the variables were as follows:

intLakeHeight	10
intLakeRightWidth	8
intLakeRightShape	1.2
intLakeLeftWidth	7
intLakeLeftShape	3.2

Graph on following page.

The corresponding graph in desmos after entering these values was as follows:



Where $h = \text{intLakeHeight}$; $w = \text{intLakeRightWidth}$; $b = \text{intLakeRightShape}$; $W = \text{intLakeLeftWidth}$ and $B = \text{intLakeLeftShape}$.

This graph corresponds exactly to the shape of the generated lake, therefore this test is a success.

Second test:

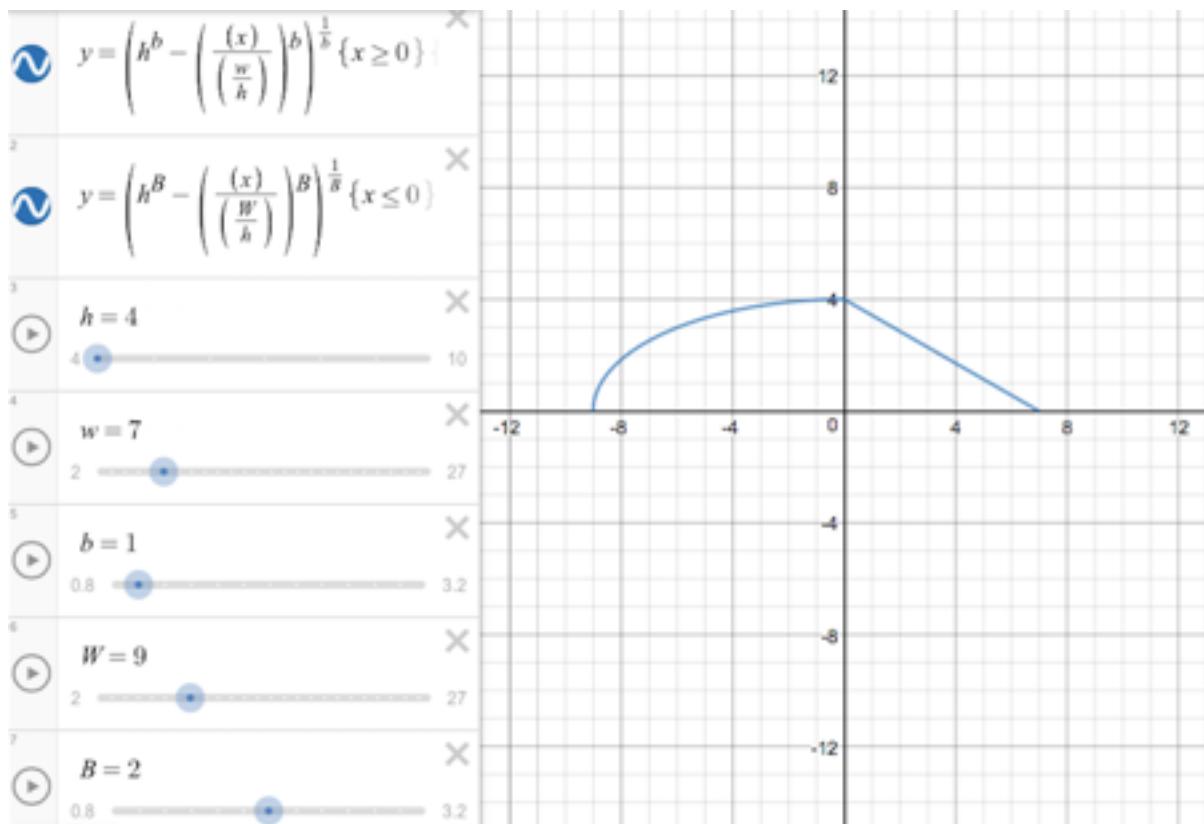
Console result:



As seen in Watch 1 the values for the variables were as follows:

intLakeHeight	4
intLakeRightWidth	7
intLakeRightShape	1
intLakeLeftWidth	9
intLakeLeftShape	2

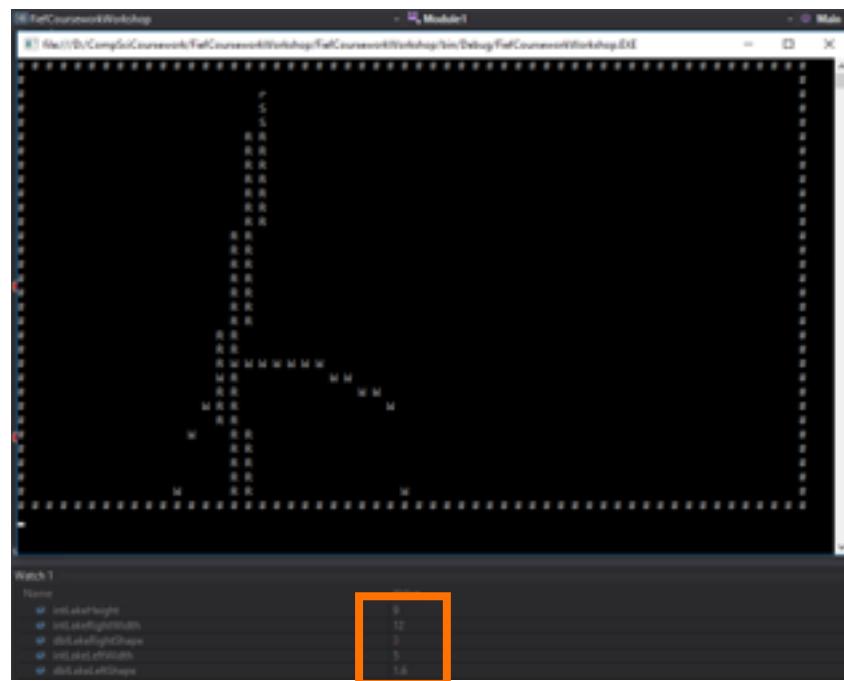
The corresponding graph in desmos after entering these values was as follows:



This graph corresponds exactly to the shape of the generated lake, therefore this test is a success.

Third and final test is shown on the following page.

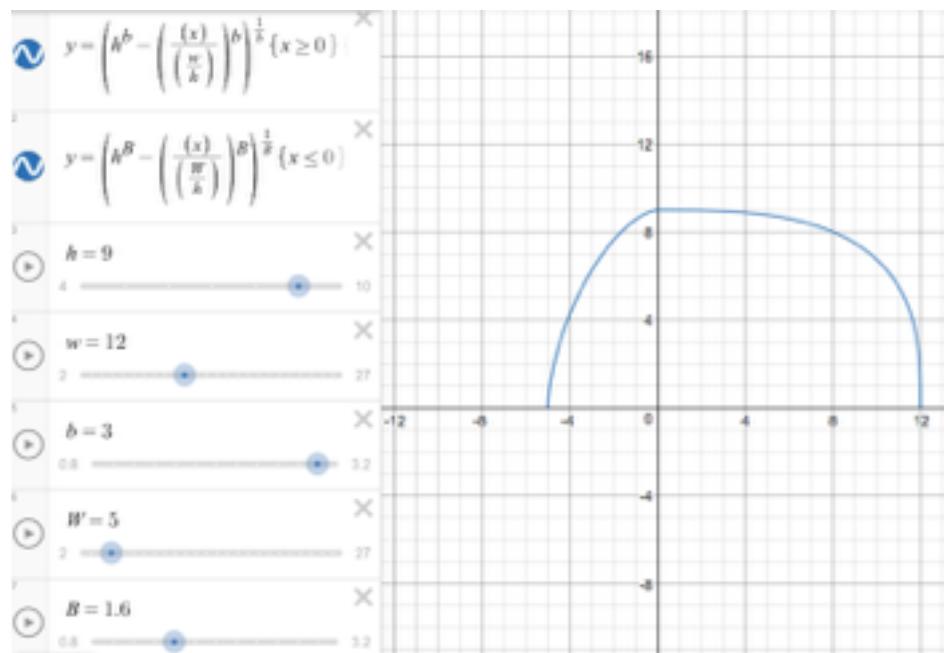
Console result:



As seen in Watch 1 the values for the variables were as follows:

intLakeHeight	9
intLakeRightWidth	12
intLakeRightShape	3
intLakeLeftWidth	5
intLakeLeftShape	1.6

The corresponding graph in desmos after entering these values was as follows:



This graph corresponds exactly to the shape of the generated lake, therefore this test is a success.

DD2.11

As mentioned previously, this test will be conducted by using a trace table of the code, using inputs of the worst case scenarios where the array is most likely to be exceeded. The code to be tested is the following pseudo-code which is based upon the actual program, but only is concerned with statements that edit co-ordinate values.

Line	Code
1	<code>Y = RandomNum(2, 3)</code>
2	<code>X = RandomNum(8, 47)</code>
3	<code>Y = Y + 2</code>
4	<code>RiverX1 = X</code>
5	<code>RiverY = Y + 1</code>
6	<code>RiverX2 = RiverX1 + 1 or RiverX1 - 1 //RiverX2 value with change according to which extreme is being tested</code>
7	<code>While RiverY < 31</code>
8	<code> For RiverY = Y To RiverY + 3 //until RiverY = 30</code>
9	<code> If RiverY = 30 Then Exit While End If</code>
10	<code> Next RiverY</code>
11	<code> RiverX1 = RiverX1 - 1 or RiverX1 + 1 //depends on which extreme is being tested</code>
12	<code> RiverX2 = RiverX2 - 1 or RiverX1 + 1 //depends on which extreme is being tested</code>
13	<code> End While</code>
14	<code> If RiverX1 is on the right swap RiverX1 and RiverX2 around End If</code>
15	<code> LakeWidth = 27</code>
16	<code> If RiverX1 - LakeWidth < 1 Then</code>
17	<code> LakeLeftMax = RiverX1 - 1 End If</code>
18	<code> LakeLeft = RiverX1 - RandomNum(2, LakeLeftMax)</code>
19	<code> LakeRight = RiverX2 + (LakeWidth - (RiverX1 - LakeLeft))</code>

Line	Code
20	LakeLeftWidth = RiverX1 - LakeLeft
21	LakeRightWidth = LakeRight - RiverX2
22	For GraphX = -LakeLeftWidth To 0 or 0 To LakeRightWidth //depending upon
23	X = RiverX1 + GraphX or RiverX2 + GraphX
24	Next GraphX

This pseudocode simulates a situation where the river source generates either at the extreme top-left or top-right; the stream is only 2 tiles long (as seen in line 3), this has the effect of allowing the maximum time for the river to meander; and the river always meanders towards the closest edge of the map. To produce this specific effect when the left extreme is being tested, the values of RiverX1 and RiverX2 will be decreased as seen in lines 11 and 12. These values are increased when testing the right extreme. When doing the dry run, the trace table will only display the final co-ordinates of the while loop of lines 7 to 13 after each meander, to simplify the dry run. The test will be successful if the X value of line 23 is within the bounds of the array. If X ever exceeds the bounds of the array (i.e $X < 1$ or $X > 54$) then the test will fail. The trace table for the left extreme values is shown below:

Line	Variables					
	X	Y	RiverX1	RiverX2	RiverY	Boolean Conditions
1						
2	8					
3		4				
4				8		
5						5
6					7	
7						TRUE
10						9
11			7			
12				6		
10						13
11			6			
12				5		
10						17

Line	X	Variables											
		River X1	River X2	River Y	Lake Width	Lake Left Max	Lake Left	Lake Right	Lake Left Width	Lake Right Width	Graph X	Boolean Conditions	
11		5											
12			4										
10				21									
11		4											
12			3										
10				25									
11		3											
12			2										
10				29									
11		2											
12			1										
10				30									
9												TRUE	
13													
14		1	2										
15					27								
16												TRUE	
17						0							
18							1						
19								28					
20									1				
21										26			

There is no need to continue this dry run of this extreme as it is already clear that it has failed from the values of LakeLeftWidth and LakeRightWidth. The failure was caused by the fact that RiverX1 = 1; which is the edge of the map. Thus in line 18 when the program tries to generate a random number it is presented with a lower bound of 2 and an upper bound of 0, which makes no sense. The program always returns 0 in this case. Line 20 makes the LakeLeftWidth = 1; and as RiverX1 = 1, 1 - 1 = 0, which is a border tile, which is not allowed to be assigned a lake tile type, thus the test fails. This can be fixed by ensuring RiverX1 - LakeLeft > 0. This can be accomplished by

ensuring RiverX1 > 2 by ensuring the river never meanders that far left.

Evidence of remedial action:

Initially the code for the river meandering was as follows:

```
intCount = intRiverY
While intRiverY < 31
    For intRiverY = intCount To intRiverY + intMagRiverMeandering
        arrMapBlueprint(intRiverY, intRiverX1).ptyTileType = "River"
        arrMapBlueprint(intRiverY, intRiverX2).ptyTileType = "River"
        If intRiverY = 30 Then
            Exit While
        End If
    Next
    Select Case FuncRandomNumberGen(1, 2)
        Case 1
            intRiverX1 = intRiverX1 - 1
            intRiverX2 = intRiverX2 - 1
        Case 2
            intRiverX1 = intRiverX1 + 1
            intRiverX2 = intRiverX2 + 1
    End Select
    intCount = intRiverY
End While
```

To ensure the river does not meander too far left, the X-co-ordinate value of the left hand side of the river must never be less than 3. However RiverX1 and RiverX2 could refer to either side of the river. Later on in the program, RiverX1 is set to always be on the left and RiverX2 on the right. This code must now be moved to above this section of code seen above so that the X-co-ordinate of the left hand side of the river is always known. Now in the select case statement it can be guaranteed that the river will always meander left or right if the river is too far right or left respectively by checking ht values of intRiverX2 or intRiverX1 respectively. The new code is shown below:

```
intCount = intRiverY
While intRiverY < 31
    For intRiverY = intCount To intRiverY + intMagRiverMeandering
        arrMapBlueprint(intRiverY, intRiverX1).ptyTileType = "River"
        arrMapBlueprint(intRiverY, intRiverX2).ptyTileType = "River"
```

```

If intRiverY = 30 Then
    Exit While
End If
Next
Select Case intRiverX1
Case 3
    intTemp = 2
Case 51
    intTemp = 1
Case Else
    intTemp = FuncRandomNumberGen(1, 2)
End If

Select Case intTemp
Case 1
    intRiverX1 = intRiverX1 - 1
    intRiverX2 = intRiverX2 - 1
Case 2
    intRiverX1 = intRiverX1 + 1
    intRiverX2 = intRiverX2 + 1
End Select
intCount = intRiverY
End While

```

The new select case checks the value of intRiverX1: if it is equal to 3 then the river is forced to meander right; if it is equal to 51 then the river is forced to meander left. This is in anticipation of a similar issue taking place on the other side. Below are two images of two different maps without water features where the river has generated in each of the extremes. These images demonstrate the river being forced to meander away from the side when they meander too close to the edge.



Now that the bugs associated with the left hand side extreme have been resolved, the right hand side must now be tested. The trace table for the right extreme values is shown below:

Line	Variables					Boolean Conditions
	X	Y	RiverX1	RiverX2	RiverY	
1		2				
2	47					
3		4				
4			47			
5					5	
6				48		
7						TRUE
10					9	
11			48			
12				49		
10					13	
11			49			
12				50		
10					17	
11			50			
12				51		
10					21	
11			51			
12				52		
10					25	
11			52			
12				53		
10					29	
11			53			
12				54		
10					30	
9						TRUE

Line	X	River X1	River X2	Lake Width	Lake Left Max	Lake Left	Lake Right	Lake Left Width	Lake Right Width	Graph X	Boolean Conditions
13											
14											
15				27							
16											FALSE
18						51					
19							79				
20								2			
21									25		
22										0	
23	54										
24										1	
23	55										
24										2	
23	56										

The X value is now such a value that it exceeds not only the acceptable region (i.e. within the border tiles) but also the map array entirely. This test has therefore failed. This happened because all the calculations for the limits of the lake only took the left hand side into account, and didn't check the right hand side at all. This meant that when the LakeLeft value was given a small value whilst the river was on the right hand side; LakeRight was assigned a value that would certainly exceed the boundaries of the array. The remedial action will involve adding a similar bounds check that already exists for the left hand side to the right hand side. This will limit the size of the lake on the right hand side to a minimum of 2 tiles wide if the river is too close to the edge of the map. During the test of the left hand extremes remedial action was taken in advance to ensure the river never meanders too close to the edge of the map on both sides, thus there will also be at least 2 tiles on the right hand side available for the lake to generate. A combination of these solutions will solve the error that caused this test to fail.

Evidence of remedial action:

Initially the code for creating the maximum dimensions of the lake was as follows:

```
intLakeWidth = FuncRandomNumberGen(15, 27)
intLakeHeight = FuncRandomNumberGen(4, 10)
```

```
intLakeLeftMax = intLakeWidth - 2
```

```
If intRiverX1 - intLakeWidth < 0 Then
    intLakeLeftMax = intRiverX1 - 1
End If
```

```
intLakeLeft = intRiverX1 - FuncRandomNumberGen(2, intLakeLeftMax)
intLakeRight = intRiverX2 + (intLakeWidth - (intRiverX1 - intLakeLeft))
intLakeLeftWidth = intRiverX1 - intLakeLeft
intLakeRightWidth = intLakeRight - intRiverX2
```

The new code will mirror this old code but will add an equivalent for the right hand side. If the River is on the right hand side of the map, this new code will run instead of the code above; whilst if the river is on the left on side of the map, the new code will not be run and the old code will be ran instead. The new code is as follows:

```
intLakeWidth = FuncRandomNumberGen(15, 27)
intLakeHeight = FuncRandomNumberGen(4, 10)
```

```
If intRiverX1 <= 27 Then
    intLakeLeftMax = intLakeWidth - 2

    If intRiverX1 - intLakeWidth < 1 Then
        intLakeLeftMax = intRiverX1 - 1
    End If
```

```
    intLakeLeft = intRiverX1 - FuncRandomNumberGen(2, intLakeLeftMax)
    intLakeRight = intRiverX2 + (intLakeWidth - (intRiverX1 - intLakeLeft))
    intLakeLeftWidth = intRiverX1 - intLakeLeft
    intLakeRightWidth = intLakeRight - intRiverX2
```

```
Else
    intLakeRightMax = intLakeWidth - 2
```

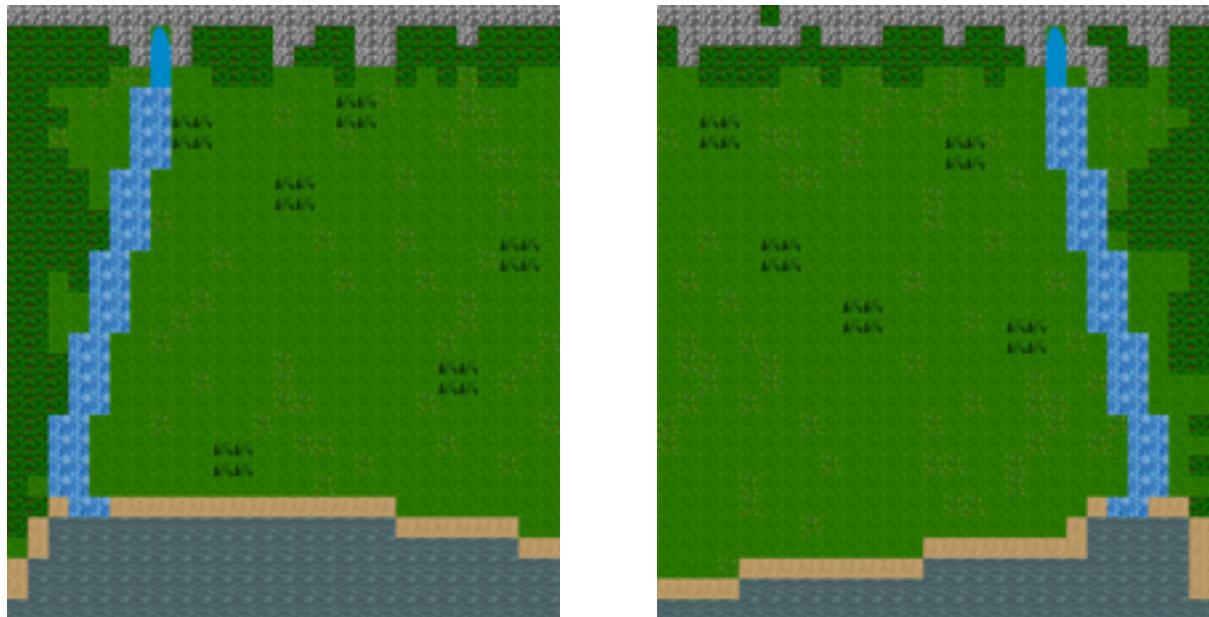
```
    If intRiverX2 - intLakeWidth > 54 Then
```

```

    intLakeRightMax = 54 - intRiverX2
End If
intLakeRight = intRiverX2 - FuncRandomNumberGen(2, intLakeRightMax)
intLakeLeft = intRiverX1 - (intLakeWidth - (intLakeRight - intRiverX2))
intLakeLeftWidth = intRiverX1 - intLakeLeft
intLakeRightWidth = intLakeRight - intRiverX2
End If

```

The new code now ensures that the lake dimensions never exceed the bounds of the array. Two screenshots below demonstrated a map with extreme values for the river and lake dimensions that successfully generates.



Now that the bugs associated with the right hand side are all resolved, this test is complete.

DD2.12

As with DD2.10, I ran the code and compared the shape of the generated sea to the shape of the corresponding graph on desmos with identical variable values. I repeated 3 times to reduce uncertainty.

In the console '#’ represents border tiles; ‘ ’ represents an empty tile ‘r’ represents the river source; ‘s’ represents stream tiles; ‘R’ represents river tiles and ‘W’ represents a sea/lake tile.

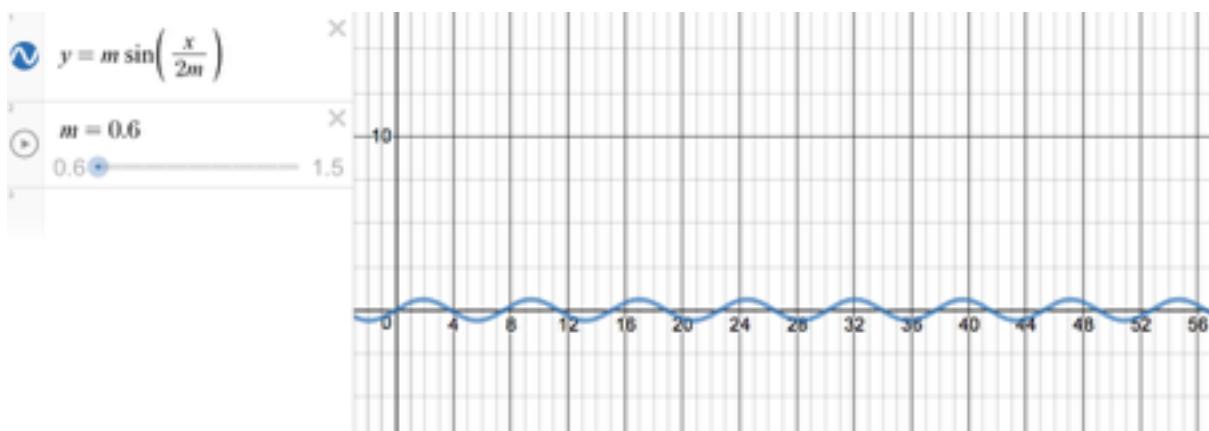
First test:

Console result on following page:



As seen in Watch 1, dblSeaShape = 0.6 and blnSineTrue = True, thus the program utilised a sine function.

The corresponding graph in desmos after entering these values was as follows:



Where m = dblSeaShape.

This graph corresponds approximately to the shape of the generated lake, therefore this test is a success.

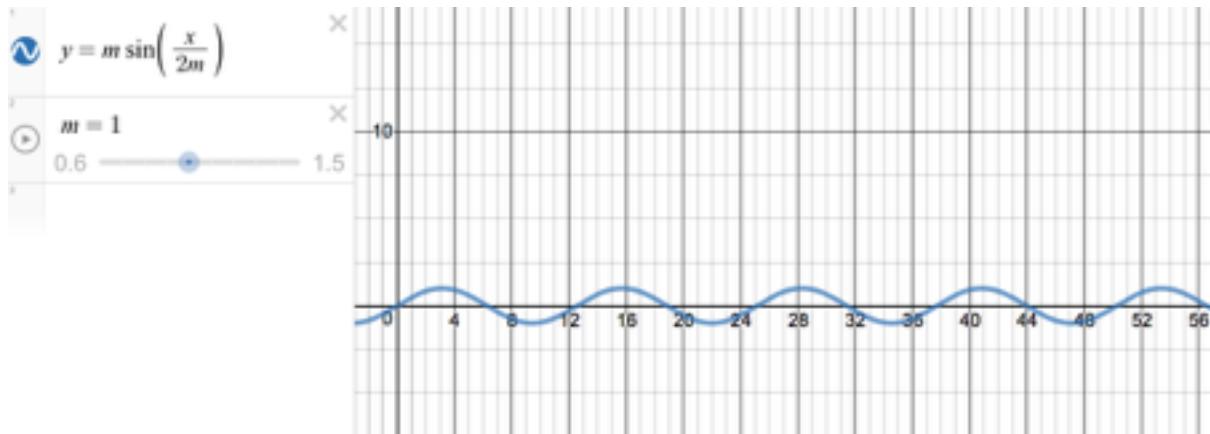
Second test:

Console result:



As seen in Watch 1, dblSeaShape = 1 and blnSineTrue = True, thus the program utilised a sine function.

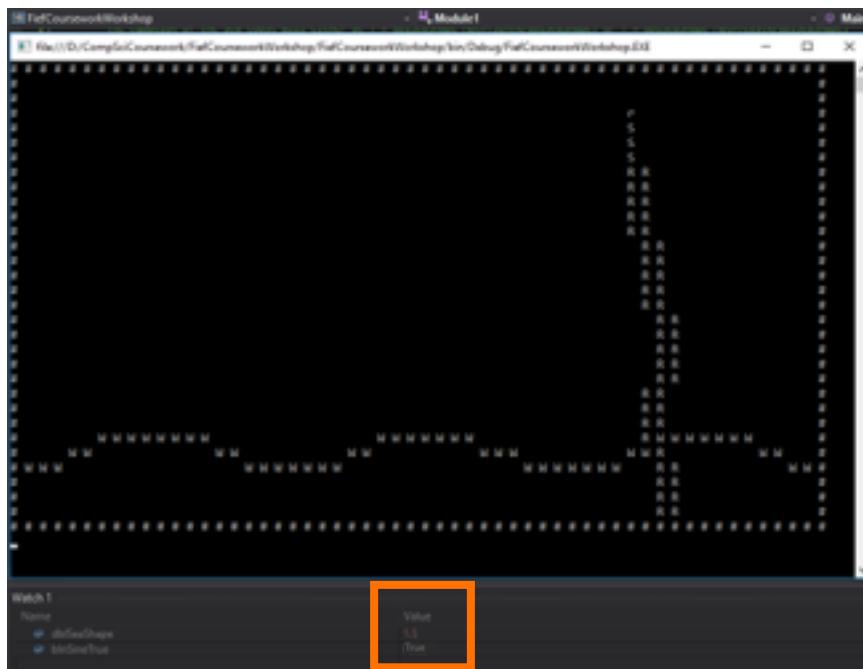
The corresponding graph in desmos after entering these values was as follows:



This graph corresponds approximately to the shape of the generated lake, therefore this test is a success.

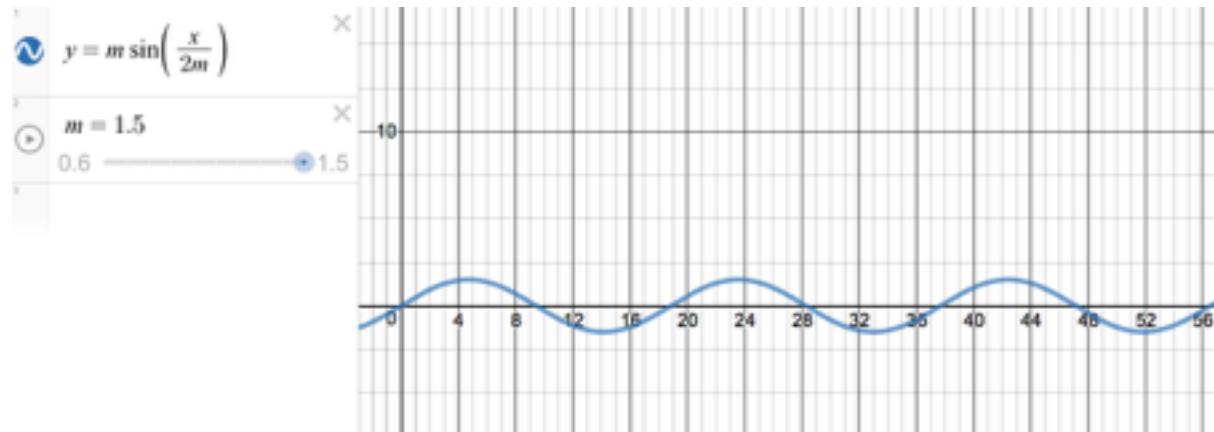
Third and final test:

Console result:



As seen in Watch 1, dblSeaShape = 1.5 and blnSineTrue = True, thus the program utilised a sine function.

The corresponding graph in desmos after entering these values was as follows:



This graph corresponds approximately to the shape of the generated lake, therefore this test is a success. All tests were successful and thus DD2.12 is a success.

Stakeholder review and sign-off of iteration

I held a session with my stakeholders where I demonstrated the final product of this iteration. I asked my stakeholders to consider if the requirements of the iteration were satisfied by the progress made in this iteration. My stakeholders after testing the solution multiple times unanimously agreed that the program in its current form fulfils the requirements of this iteration. Early on in the iteration my stakeholders demanded that the map area be increased by four fold; this meant that only a quarter of the map was available at any one time. Thus my stakeholders have asked if within the next iteration I include map navigation. After testing the program my stakeholders were keen that whilst the map generation should remain random, there should be some features of the map that they have control over. This includes whether a lake or sea is generated; the size of mountain ranges and forest; the frequency of hills and other such issues. They have suggested that a method of controlling these features also be added in the next iteration. I have agreed to include both these in the next iteration. As the requirements of this iteration have been fulfilled, my stakeholders have agreed that I should continue to the third iteration.

END OF SECOND ITERATION

THIRD ITERATION

Discussion with Stakeholders

The core features that this iteration will implement are the in-game timer and the in-game date. As mentioned at the end of the previous iteration, as a result of the end-of-iteration discussion with my stakeholders; this iteration will also implement a method for the user to be able to choose certain features of the map and map navigation. My stakeholders have no preference of which order these features should be implemented so the order will be arbitrary.

My stakeholders said that the in-game date should progress at a rate of one day per second. They also asked that in the interests of realism that leap years be accounted for. The starting date was agreed to be 1st January 1296 which was chosen as it fits in with the civilisations.

A menu will be added to allow the user to choose map features. This will be added in-between the civilisation menu and the town name menu. I asked my stakeholders what features they would like to have control over. They looked through the code of my previous iteration and identified 8 features. These are in no particular order: whether a sea or lake generates; the height of the lake; the width of the lake; the bendiness of the river; the size of mountain ranges; the size of forests; the number of hills; and the density of flora.

I suggested to my stakeholders that I should implement map navigation by means of a mini-map that they could click to move around the map, my stakeholders agreed. This mini-map will have to be visually similar to the actual map so that the users know which area of the map they are currently looking at and which area of the map they would like to view.

Specify the problem: requirements of map settings; navigation and in-game date

The iteration must fulfil the following requirements:

R3.1. The user must be able to control over the following map features from a menu:

- Whether a sea or lake generates;
- The height of the lake;
- The width of the lake;
- The amount of river bendiness;
- The size of mountain ranges;
- The size of forests;
- The number of hills;

- The density of flora.

R3.2. There must be an in-game mini-map which displays a small image of the map.

This mini-map must allow the user to jump to the four different corners of the map so that the user may interact with all parts of the map.

R3.3. There must be an in-game timer that calculates the in-game date starting on the 1st January 1296 and accounts for leap years.

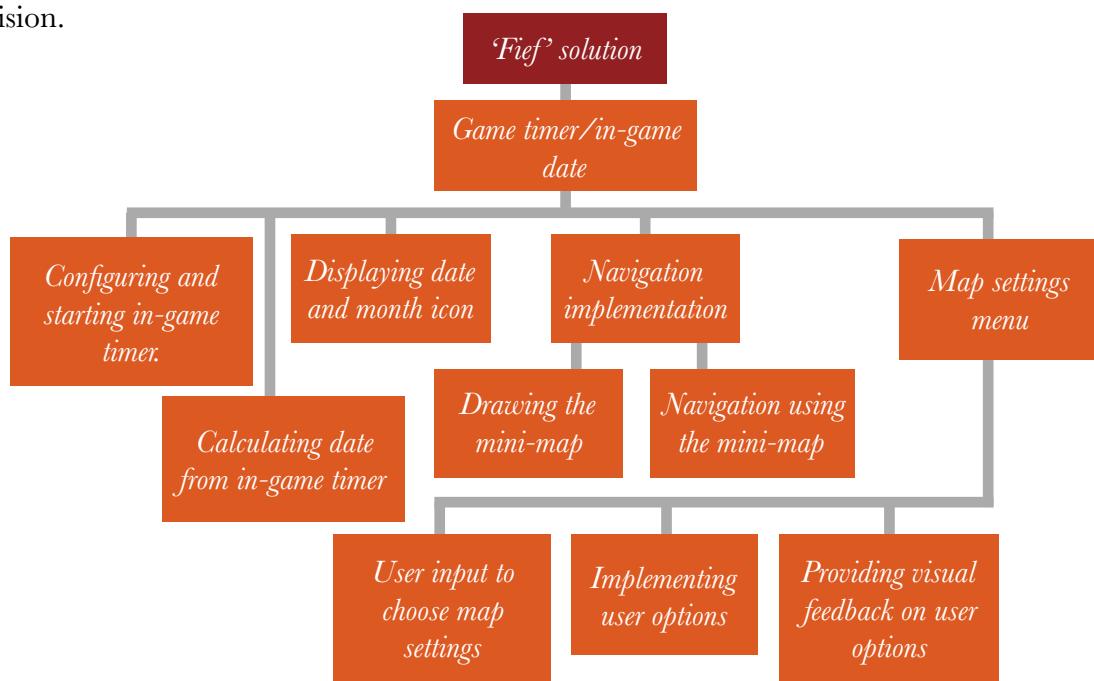
R3.4. The in-game date must be correctly displayed with an icon for each month.

Success criteria and requirements the iteration addresses:

Success Criteria	Requirement	How success criteria requirement is addressed
SC7	SCR7.3	The calculated in-game date will be displayed in the appropriate place and will display an icon illustrating the month.
SC10	SC10.1	The new menu will have a path back to the main menu and the town name menu, and thus a path to all other menus.

Decomposing the Problem

The diagram below illustrates the decomposition of the problem presented in this iteration. Despite the changes to the content of the iteration, the iteration name has remained the same. All code related to the in-game date and timer will be executed once the in-game form is loaded. The mini-map will be printed as the map is being printed to the in-game screen. A separate form will need to be created for the map settings. As the user inputs selections for the various settings they should be given visual feedback illustrating their decision.



Describe the solution (Usability features)

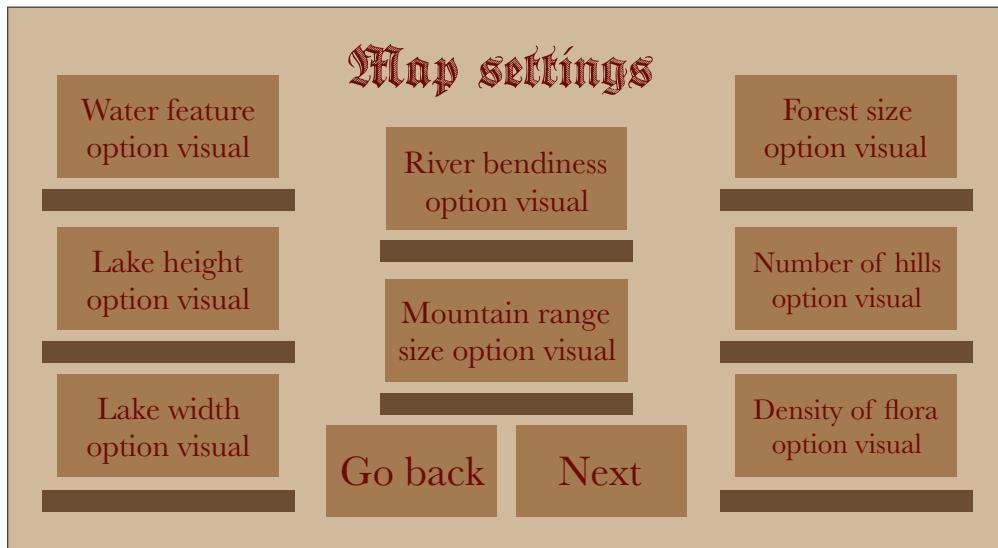
The map settings menu will be accessed by the user after choosing a civilisation and before entering the town name. From the diagram in iteration 1 the new menu will fit in as shown:



As the new menu can access the select civilisation menu, it can indirectly access the main menu; and as it can access the town name menu the new menu can indirectly access the in-game form, thus the new menu will be able to navigate to all menus as required by SCR10.1.

The options for map settings will be implemented using track-bar classes. This class from the Visual studio toolbox allows the user to slide an icon to assign the track-bar a value. This value can be used to change the map settings.

A wireframe design of the new menu is shown below:



Track-bars are shown by dark thin rectangles, each are associated with the option displayed directly above each track-bar.

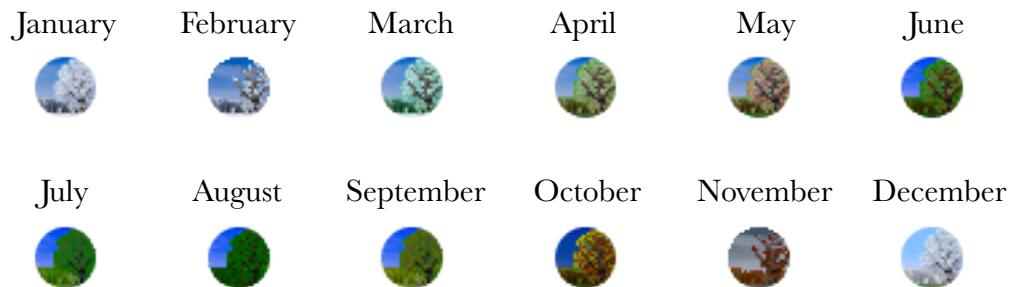
I decided to use track-bars as it simplifies selecting the various options, and when labelled, it is very clear to the user what option they have selected. Upon selecting various values on each track-bar the respective visual display will change image to describe the option to the user.

The mini-map will be drawn in the bottom-right part of the information section of the in-game screen. As illustrated below:



The mini-map will be constructed from four picture boxes which will have printed to them the four respective quadrants of the map. There will be a click event affiliated with each that will print the associated part of the map to the screen. As mentioned previously, the mini-map will be printed at the same time the map is printed to the screen after the map is generated.

The in-game date will be calculated by utilising a timer class from the visual studio toolbox. The value of the timer will increment once every 1 second, every time the timer ticks code that will advance the in-game date will be executed from within the timer tick event. When the in-game form is loaded the timer will be started immediately. When I designed the forms in the first iteration, I also created 12 icons to represent each month. These icons will be printed to a picture box by the in-game date display in the top right corner of the screen. These icons are shown below in order:



I decided to include icons to illustrate the months as it improves the aesthetic of the user interface; and as each icon depicts the same landscape, the user should be given the impression of an animated display.

Describe the solution (Key variable, data structures, classes and validation checks)

In order to implement the map settings, the mini-map and the in-game date, certain key variables, data structures and classes will be required. There is no need for any validation checks this iteration. A table describing the key variables is shown on the following page.

Variable Name	Variable Type	Justification
blnSeaLake	Boolean	Used to allow the user to choose either a sea or lake water feature. If true a sea will generate, else a lake will instead.
intYLower	Integer	Used to define the region of the map that will be printed during navigation.
intYUpper	Integer	"
intXLower	Integer	"
intXUpper	Integer	"
intDateDay	Integer	Used to calculate the in-game date from the in-game timer.
strDateMonth	String	Used to store and display the current in-game month.
intDateYear	Integer	Used to store and display the current in-game year. Also used to calculate leap years.
strDateDay	String	Used to store and display the current in-game day.
gfxMinimapTL	Graphics	Used to print the mini-map.
gfxMinimapTR	Graphics	"
gfxMinimapBL	Graphics	"
gfxMinimapBR	Graphics	"

Some variables from the previous iteration will be reused in a different way in this iteration to implement map settings. These are in no particular order: intMountainFrequency, intForestProbability, intHillFrequency, intFloraProbability, intLakeWidth, intLakeHeight, intRiverBendiness. These will be declared as public variables so that they are accessible in both ModuleMapGeneration and in the map settings form.

Three types of classes will be used to implement the various features of this iteration. Eight track-bar classes will be used to allow the user to customise eight features of the map. These will be named according to the option they are associated with: trbLakeWidth, trbLakeHeight, trbWaterFeature, trbRiverBendiness, trbMountainFrequency, trbForestSize, trbHillFrequency and trbFloraFrequency. These features have been outlined previously.

The second type of class will be the timer class. One of these will be utilised (called tmrGameTimer). The interval between each tick in milliseconds will be 1000. This will be utilised to calculate the in-game date as described previously.

A new form class will be added to accommodate the map settings menu as well. A new form is necessary as there is no room on any other previous menu.

Approach to testing (Usability features)

It will be necessary for my stakeholders to perform usability feature testing on the elements implemented in this iteration to confirm the iteration satisfies the success criteria this iteration addresses and any new requirements resulting from the new additions (mini-map navigation and map settings). A table of these tests is shown below.

Feature number	Usability test description
UF3.1	Is the in-game date displayed correctly and clearly on the in-game form?
UF3.2	Is the method of navigation appropriate, and does it work correctly?
UF3.3	Is the method of selecting map settings appropriate and are the options reflected in the generated map?

Approach to testing (During development)

This iteration implements features that can only be tested by running the program and checking the visual output. Tests will be related to whether the mini-map navigation works and whether the in-game date is calculated correctly. There is no need to compare user inputs in the map settings menu with the generated maps with too much detail as the values are the same as the last iteration and thus it should work in the same way, however this will also be tested. The table below describes the tests to inform development.

Test number	Description of test	Expected Result
DD3.1	Test that the mini-map navigation works correctly.	Clicking the four corners of the map will navigate the user to the respective quadrant of the map.
DD3.2	Test that the mini-map is printed correctly.	The mini-map corresponds exactly to the generated map.
DD3.3	Check the date is calculated and is displayed correctly.	The date is displayed in the top right display of the in-game user interface. The month icon is displayed correctly with the respective image.
DD3.4	Test that the map settings are reflected correctly in-game.	The map settings correctly affect the generated map.

Approach to testing (Post development)

My stakeholders will perform post development tests to test robustness and to ensure that this iteration fulfils the success criteria requirements this iteration addresses. The tests are shown in the table on the following page with descriptions of the tests and expected results.

Test number	Description of test	Expected Result
PD3.1	Is the date formatted correctly, and does the in-game date advance at a reasonable pace?	Date is correctly formatted and the date advances at 1 day per second.
PD3.2	Does the new map settings menu have a route to all other menus/forms?	The new menu has routes to all other menus.

Describe the solution (Pseudo-code algorithms)

(PC3.1) Algorithm for the track-bar scroll event.

Procedure trbMapSetting.Scroll //when the track-bar is scrolled, the code in this procedure will run

Select Case trbMapSetting.value //This will allow me to change the value of the map setting the track-bar is related to.

Case 0

 intMapSettingVariable = Value1 //The relevant variable can be set the appropriate value to change the map generation.

 pbMapSettingDisplay.Image = Image1 //The option display's image can be changed to reflect the user selection

Case 1

 intMapSettingVariable = Value2

 pbMapSettingDisplay.Image = Image2

Case 2

 intMapSettingVariable = Value3

 pbMapSettingDisplay.Image = Image3

//etc

End Procedure

(PC3.2) Algorithm to print the mini-map.

Procedure PrintMinimap(ByVal XLower As Integer, ByVal XUpper As Integer, ByVal YLower As Integer, ByVal YUpper As Integer, ByRef MinimapGraphic As Graphics)

 For Y = YLower To YUpper //prints the region passed to the procedure

 For X = XLower To XUpper

 DestinationRectangle = New Rectangle(X * 2 - Xlower * 2, Y * 2 - YLower * 2, 2, 2) //Each tile will be represented by a 2x2 coloured graphic

```
Select Case MapArray(Y, X).ptyTileType //Selects the tile type  
Case "Plains"
```

```
    SourceRectangle = New Rectangle(0, 0, 2, 2)  
    BitmapImage = New  
    Bitmap(Source.MinimapSource.Image)  
    MinimapGraphic.DrawImage(BitmapImage,  
        DestinationRectangle, SourceRectangle,  
        GraphicsUnit.Pixel)
```

```
Case "Flora"
```

```
    SourceRectangle = New Rectangle(0, 2, 2, 2)  
    BitmapImage = New  
    Bitmap(Source.MinimapSource.Image)  
    MinimapGraphic.DrawImage(BitmapImage,  
        DestinationRectangle, SourceRectangle,  
        GraphicsUnit.Pixel)
```

```
Case "Hill"
```

```
    SourceRectangle = New Rectangle(0, 4, 2, 2)  
    BitmapImage = New  
    Bitmap(Source.MinimapSource.Image)  
    InGameGraphics.DrawImage(BitmapImage,  
        DestinationRectangle, SourceRectangle,  
        GraphicsUnit.Pixel)
```

```
Case "Forest"
```

```
    SourceRectangle = New Rectangle(0, 6, 2, 2)  
    BitmapImage = New  
    Bitmap(Source.MinimapSource.Image)  
    MinimapGraphic.DrawImage(BitmapImage,  
        DestinationRectangle, SourceRectangle,  
        GraphicsUnit.Pixel)
```

```
Case "RiverSource"
```

```
    SourceRectangle = New Rectangle(0, 8, 2, 2)  
    BitmapImage = New  
    Bitmap(Source.MinimapSource.Image)  
    MinimapGraphic.DrawImage(BitmapImage,  
        DestinationRectangle, SourceRectangle,  
        GraphicsUnit.Pixel)
```

Case “Stream”

```
SourceRectangle = New Rectangle(0, 10, 2, 2)
BitmapImage = New
Bitmap(Source.MinimapSource.Image)
MinimapGraphic.DrawImage(BitmapImage,
DestinationRectangle, SourceRectangle,
GraphicsUnit.Pixel)
```

Case “River”

```
SourceRectangle = New Rectangle(0, 12, 2, 2)
BitmapImage = New
Bitmap(Source.MinimapSource.Image)
MinimapGraphic.DrawImage(BitmapImage,
DestinationRectangle, SourceRectangle,
GraphicsUnit.Pixel)
```

Case “Coast”

```
SourceRectangle = New Rectangle(0, 14, 2, 2)
BitmapImage = New
Bitmap(Source.MinimapSource.Image)
MinimapGraphic.DrawImage(BitmapImage,
DestinationRectangle, SourceRectangle,
GraphicsUnit.Pixel)
```

Case “Sea”

```
SourceRectangle = New Rectangle(0, 16, 2, 2)
BitmapImage = New
Bitmap(Source.MinimapSource.Image)
MinimapGraphic.DrawImage(BitmapImage,
DestinationRectangle, SourceRectangle,
GraphicsUnit.Pixel)
```

Case “Mountain”

```
SourceRectangle = New Rectangle(0, 18, 2, 2)
BitmapImage = New
Bitmap(Source.MinimapSource.Image)
MinimapGraphic.DrawImage(BitmapImage,
DestinationRectangle, SourceRectangle,
GraphicsUnit.Pixel)
```

End Select

```
    Next X  
    Next Y  
End Procedure
```

MinimapSource will be a new picture box in FormImgRef that will store the colours used to print the mini-map. When the procedure is called, the relevant graphics variable will be passed to the procedure as a parameter.

(PC3.3) Algorithm to calculate the in-game date. This procedure will be called from the game timer tick event.

```
Procedure CalculateInGameDate  
    intDateDay = DateDay + 1 //This will increment every timer tick  
    //As the processor runs at millions of time per second, the date needs to be  
    checked the in-game day after the turn of the month else the user would never  
    see the last day of the month  
    //Account for non-leap years:  
    If intDateDay = 29 And DateMonth = "February" And DateYear != 0 Then  
        intDateDay = 1 //Reset day counter  
        DateMonth = "March"  
        MonthDisplay.Image = Source.MarchIcon.Image //Update month  
        display  
    End If  
    //Account for leap years  
    If intDateDay = 30 And DateMonth = "February" Then  
        intDateDay = 1 //Reset day counter  
        DateMonth = "March"  
        MonthDisplay.Image = Source.MarchIcon.Image //Update month  
        display  
    End If  
    //Account for other shorter months  
    If intDateDay = 31 And (DateMonth = "April" Or "June" Or "September" Or  
    "November") Then  
        Select Case DateMonth  
        Case "April"  
            DateMonth = "May"
```

```

        MonthDisplay.Image = Source.MayIcon.Image //Update month
        display
    //Repeat cases for remaining months
    End Select
End If
//Account for longer months
If intDateDay = 32 Then
    Select Case DateMonth
    Case "January"
        DateMonth = "February"
        MonthDisplay.Image = Source.FebruaryIcon.Image //Update
        month display
    //Repeat cases for remaining months, incrementing DateYear if the
    month changes from December to January
    End Select
End If
//Format the days correctly
Select Case intDateDay
Case 1 Or 21 Or 31
    strDateDay = (intDateDay & "st")
Case 2 Or 22
    strDateDay = (intDateDay & "nd")
Case 3 Or 23
    strDateDay = (intDateDay & "rd")
Case Else
    strDateDay = (intDateDay & "th")
End Select
//Update the date display
lblDate.Text = (strDateDay & " " & DateMonth & " " & DateYear)
End Procedure

```

The code to navigate upon clicking the mini-map quadrants is almost identical to the code used to print the map in the previous iteration, however some arguments will need to be added so that the procedure will accept regions of the map in order to print the correct quadrant when clicked.

The table below describes how each of these pseudo-code algorithms form part of the solution and which requirements each addresses.

Pseudo-code algorithm	Description of how algorithm forms part of solution	Success criteria/requirements fulfilled	Justification of how algorithms fulfils criteria
PC3.1	This algorithm will be used as a template for all track-bars implemented in the map settings form.	R3.1	This will allow the user to easily select map settings.
PC3.2	This algorithm will print the mini-map so that the user will be able to navigate around the map.	R3.2	By printing the mini-map, the user has a clear view of the generated map, and navigation becomes clearer.
PC3.3	This algorithm will calculate and display the in-game date.	R3.3 R3.4	This algorithm correctly calculates and formats the date (including leap years). The month icon is also updated each month.

Developing the solution (Coding)

The code for this iteration can be divided into three main sections: the map settings form, the in-game date and the mini-map. These will be coded in an arbitrary order. Some other aspects of the solution coded in previous iterations will need to be edited to allow for the new changes (e.g. the method that the water feature was chosen). These edits will be shown after the main coding section.

Code for in-game date and the mini-map click events:

The code for the in-game date all appears within the FormInGame form. The new variables for the in-game time were all declared as global variables, as they needed to be initialised once upon being declared and they needed to be visible from within the UpdateGameDate procedure.

```
Dim intDateDay As Integer = 1
Dim strDateDay As String
Dim strDateMonth As String = "January"
Dim intDateYear As Integer = 1296
```

```
Private Sub FormInGame_Paint(sender As Object, e As PaintEventArgs)
Handles pbMapDisplay.Paint
'Prints the top left quadrant of the map to the screen:
subPrintMap(1, 27, 1, 15)
```

```
'Prints the minimap to four interactive picture boxes:  
subPrintMinimap(1, 27, 1, 15, gfxMinimapTL)  
subPrintMinimap(28, 54, 1, 15, gfxMinimapTR)  
subPrintMinimap(1, 27, 16, 30, gfxMinimapBL)  
subPrintMinimap(28, 54, 16, 30, gfxMinimapBR)  
End Sub
```

```
Private Sub pbMinimapTL_Click(sender As Object, e As EventArgs)  
Handles pbMinimapTL.Click  
    'Print the top left corner of the map to the screen:  
    subPrintMap(1, 27, 1, 15)  
    'Update the cursors of the mini-map buttons:  
    pbMinimapTL.Cursor = Cursors.Default  
    pbMinimapBL.Cursor = Cursors.PanSouth  
    pbMinimapTR.Cursor = Cursors.PanEast  
    pbMinimapBR.Cursor = Cursors.PanSE  
    'Update the relavant variable:  
End Sub
```

```
Private Sub pbMinimapTR_Click(sender As Object, e As EventArgs)  
Handles pbMinimapTR.Click  
    'Print the top right corner of the map to the screen:  
    subPrintMap(28, 54, 1, 15)  
    'Update the cursors of the mini-map buttons:  
    pbMinimapTR.Cursor = Cursors.Default  
    pbMinimapBL.Cursor = Cursors.PanSW  
    pbMinimapTL.Cursor = Cursors.PanWest  
    pbMinimapBR.Cursor = Cursors.PanSouth  
    'Update the relavant variable:  
End Sub
```

```
Private Sub pbMinimapBL_Click(sender As Object, e As EventArgs)  
Handles pbMinimapBL.Click  
    'Print the bottom left corner of the map to the screen:  
    subPrintMap(1, 27, 16, 30)  
    'Update the cursors of the mini-map buttons:  
    pbMinimapBL.Cursor = Cursors.Default  
    pbMinimapTL.Cursor = Cursors.PanNorth  
    pbMinimapTR.Cursor = Cursors.PanNE  
    pbMinimapBR.Cursor = Cursors.PanEast  
    'Update the relavant variable:  
End Sub
```

```

Private Sub pbMinimapBR_Click(sender As Object, e As EventArgs)
Handles pbMinimapBR.Click
    'Print the bottom right corner of the map to the screen:
    ModulePrintMap.subPrintMap(28, 54, 16, 30)
    'Update the cursors of the mini-map buttons:
    pbMinimapBR.Cursor = Cursors.Default
    pbMinimapBL.Cursor = Cursors.PanWest
    pbMinimapTR.Cursor = Cursors.PanNorth
    pbMinimapTL.Cursor = Cursors.PanNW
    'Update the relevant variable:
End Sub

Private Sub tmrGameTimer_Tick(sender As Object, e As EventArgs)
Handles tmrGameTimer.Tick
    'Update the game date:
    subUpdateGameDate()
End Sub

Sub subUpdateGameDate()
    'Increment the day:
    intDateDay = intDateDay + 1
    'Check that the date has reached the end of February (on a normal year)
    If intDateDay = 29 And strDateMonth = "February" And intDateYear Mod
        4 <> 0 Then
            'Reset the day count and increment the month count
            intDateDay = 1
            strDateMonth = "March"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbMarch.Image
        End If
    'Check that the date has reached the end of February (on a leap year)
    If intDateDay = 30 And strDateMonth = "February" Then
        intDateDay = 1
        strDateMonth = "March"
        'Update the month display:
        pbMonthDisplay.Image = FormImgRef.pbMarch.Image
    End If
    'Check that the date has reached the end of a short month (30 days)
    If intDateDay = 31 And (strDateMonth = "April" Or strDateMonth = "June"
        Or strDateMonth = "September" Or strDateMonth = "November") Then
        'Reset the day count and increment the month count according to the
        current month:
        intDateDay = 1

```

```

Select Case strDateMonth
Case "April"
    strDateMonth = "May"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbMay.Image
Case "June"
    strDateMonth = "July"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbJuly.Image
Case "September"
    strDateMonth = "October"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbOctober.Image
Case "November"
    strDateMonth = "December"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbDecember.Image
End Select
End If
'Check that the date has reached the end of a long month (31 days)
If intDateDay = 32 Then
    'Reset the day count and increment the month count according to the
    current month:
    intDateDay = 1
    Select Case strDateMonth
        Case "January"
            strDateMonth = "February"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbFebruary.Image
        Case "March"
            strDateMonth = "April"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbApril.Image
        Case "May"
            strDateMonth = "June"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbJune.Image
        Case "July"
            strDateMonth = "August"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbAugust.Image
        Case "August"
            strDateMonth = "September"

```

```

'Update the month display:
pbMonthDisplay.Image = FormImgRef.pbSeptember.Image
Case "October"
    strDateMonth = "November"
'Update the month display:
pbMonthDisplay.Image = FormImgRef.pbNovember.Image
Case "December"
    strDateMonth = "January"
'Update the month display:
pbMonthDisplay.Image = FormImgRef.pbJanuary.Image
    intDateYear = intDateYear + 1
End Select
End If
'Create the final string for the month display:
Select Case intDateDay
    Case 1
        strDateDay = (intDateDay & "st")
    Case 21
        strDateDay = (intDateDay & "st")
    Case 31
        strDateDay = (intDateDay & "st")
    Case 2
        strDateDay = (intDateDay & "nd")
    Case 22
        strDateDay = (intDateDay & "nd")
    Case 3
        strDateDay = (intDateDay & "rd")
    Case 23
        strDateDay = (intDateDay & "rd")
    Case Else
        strDateDay = (intDateDay & "th")
End Select
lblDate.Text = (strDateDay & " " & strDateMonth & " " & intDateYear)
End Sub

```

Whilst I was implementing the click events for the mini-map, I considered changing the mouse cursor of the picture boxes to indicate navigation direction. This code is shown in the click events of the mini-map click events. The code for the in-game date is in a separate procedure to tmrGameTimer tick event as in later iterations/versions as more features are added, many events will be taking place each game tick and to simplify the solution each

event should be in separate procedures. It also makes it easier to move events around within the tmrGameTimer tick event if required.

Code for printing the mini-map:

This code was added to the ModulePrintMap module as grouping similar code together in the same module appeals to maintainability. The four new graphics variables associated with the mini-map are declared in this module as public graphics variables, as they need to be accessed from within the in-game form.

```
'Subroutine prints a quadrant of the mini-map
Sub subPrintMinimap( ByVal intXLower As Integer, ByVal intXUpper As Integer, ByVal intYLower As Integer, ByVal intYUpper As Integer, ByRef gfxPlaceholder As Graphics)
    For intY = intYLower To intYUpper
        For intX = intXLower To intXUpper
            'Define the destination rectangle:
            rectGraphicsDestination = New Rectangle(intX * 2 - intXLower * 2,
intY * 2 - intYLower * 2, 2, 2)
            Select Case arrMapBlueprint(intY, intX).ptyTileType
                Case "Plains" 'Generate a plains graphic at the minimap location
                    rectGraphicsSource = New Rectangle(0, 0, 2, 2)
                    bmpImage = New
                    Bitmap(FormImgRef.pbMinimapSource.Image)
                    gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
                Case "Flora" 'Generate a flora graphic at the minimap location
                    rectGraphicsSource = New Rectangle(0, 2, 2, 2)
                    bmpImage = New
                    Bitmap(FormImgRef.pbMinimapSource.Image)
                    gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
                Case "Hill" 'Generate a hill graphic at the minimap location
                    rectGraphicsSource = New Rectangle(0, 4, 2, 2)
                    bmpImage = New
                    Bitmap(FormImgRef.pbMinimapSource.Image)
                    gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
                Case "Forest" 'Generate a forest graphic at the minimap location
                    rectGraphicsSource = New Rectangle(0, 6, 2, 2)
```

```

bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "RiverSource" 'Generate a riverSource graphic at the
minimap location
        rectGraphicsSource = New Rectangle(0, 8, 2, 2)
        bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Stream" 'Generate a stream graphic at the minimap
location
        rectGraphicsSource = New Rectangle(0, 10, 2, 2)
        bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "River" 'Generate a river graphic at the minimap location
        rectGraphicsSource = New Rectangle(0, 12, 2, 2)
        bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Coast" 'Generate a coast graphic at the location
        rectGraphicsSource = New Rectangle(0, 14, 2, 2)
        bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Sea" 'Generate a sea graphic at the minimap location
        rectGraphicsSource = New Rectangle(0, 16, 2, 2)
        bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Mountain" 'Generate a mountain graphic at the minimap
location
        rectGraphicsSource = New Rectangle(0, 18, 2, 2)
        bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)

```

```
End Select
```

```
    Next  
    Next  
End Sub
```

Unlike in the print map procedure the arguments of rectGraphicsSource change as the source image is the same, but different sections of the source are selected. For this reason the source rectangle is being assigned a new region for each array co-ordinate.

Code for printing the map settings form:

The majority of the code in this form is related to making the track-bars functional and changing the images of the visual feedback for each setting. The variables related to the map generation that the user can now customise are initialised in this form as each track-bar is set to the lowest value by default and the variables need to reflect that fact. Also if a user decided to start the game without changing any settings then the variables need to be initialised to avoid any bugs.

Public Class FormMapSettings

```
Private Sub btnReturnMapCiv_Click(sender As Object, e As EventArgs)  
Handles btnReturnMapCiv.Click  
    'Closes the map settings menu and opens the select civilisation menu:  
    Me.Close()  
    FormCivilisationMenu.Show()  
End Sub
```

```
Private Sub btnMapSettingsNext_Click(sender As Object, e As EventArgs)  
Handles btnMapSettingsNext.Click  
    'Closes the map settings menu and opens the town name menu:  
    Me.Close()  
    FormTownName.Show()  
End Sub
```

```
Private Sub FormMapSettings_Load(sender As Object, e As EventArgs)  
Handles MyBase.Load  
    'Assigns default map settings:  
    blnSeaLake = True  
    intLakeWidth = 15  
    intLakeHeight = 4
```

```
intRiverBendiness = 3  
intMountainFrequency = 0  
intHillFrequency = 10  
intFloraProbability = 10  
intForestProbability = 5  
End Sub
```

```
Private Sub trbWaterFeature_Scroll(sender As Object, e As EventArgs)  
Handles trbWaterFeature.Scroll  
    'Update the value of the map setting variable as track bar moves and  
    update the map setting feedback display:  
    If trbWaterFeature.Value = 0 Then  
        blnSeaLake = True  
        pbWaterFeature.Image = FormImgRef.pbSeaOption.Image  
    Else  
        blnSeaLake = False  
        pbWaterFeature.Image = FormImgRef.pbLakeOption.Image  
    End If  
End Sub
```

```
Private Sub trbLakeWidth_Scroll(sender As Object, e As EventArgs)  
Handles trbLakeWidth.Scroll  
    'Update the value of the map setting variable as track bar moves and  
    update the map setting feedback display:  
    Select Case trbLakeWidth.Value  
        Case 0  
            intLakeWidth = 15  
            pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image  
        Case 1  
            intLakeWidth = 16  
            pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image  
        Case 2  
            intLakeWidth = 17  
            pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image  
        Case 3  
            intLakeWidth = 18  
            pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image  
        Case 4  
            intLakeWidth = 19  
            pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image  
        Case 5  
            intLakeWidth = 20  
            pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
```

```

Case 6
    intLakeWidth = 21
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 7
    intLakeWidth = 22
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 8
    intLakeWidth = 23
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 9
    intLakeWidth = 24
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
Case 10
    intLakeWidth = 25
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
Case 11
    intLakeWidth = 26
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
Case 12
    intLakeWidth = 27
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
End Select
End Sub

```

```

Private Sub trbLakeHeight_Scroll(sender As Object, e As EventArgs)
Handles trbLakeHeight.Scroll
    'Update the value of the map setting variable as track bar moves and
    update the map setting feedback display:
    Select Case trbLakeHeight.Value
        Case 0
            intLakeHeight = 4
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 1
            intLakeHeight = 5
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 2
            intLakeHeight = 6
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 3
            intLakeHeight = 7
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 4
            intLakeHeight = 8

```

```

pbLakeHeight.Image = FormImgRef.pbLakeTall.Image
Case 5
intLakeHeight = 9
pbLakeHeight.Image = FormImgRef.pbLakeTall.Image
Case 6
intLakeHeight = 10
pbLakeHeight.Image = FormImgRef.pbLakeTall.Image
End Select
End Sub

```

```

Private Sub trbRiverBendiness_Scroll(sender As Object, e As EventArgs)
Handles trbRiverBendiness.Scroll
'Update the value of the map setting variable as track bar moves and
update the map setting feedback display:
Select Case trbRiverBendiness.Value
Case 0
intRiverBendiness = 3
pbRiverBendiness.Image = FormImgRef.pbRiverVBendy.Image
Case 1
intRiverBendiness = 4
pbRiverBendiness.Image = FormImgRef.pbRiverVBendy.Image
Case 2
intRiverBendiness = 5
pbRiverBendiness.Image = FormImgRef.pbRiverLBendy.Image
Case 3
intRiverBendiness = 6
pbRiverBendiness.Image = FormImgRef.pbRiverLBendy.Image
Case 4
intRiverBendiness = 7
pbRiverBendiness.Image = FormImgRef.pbRiverStraight.Image
Case 5
intRiverBendiness = 8
pbRiverBendiness.Image = FormImgRef.pbRiverStraight.Image
End Select
End Sub

```

```

Private Sub trbMountainFrequency_Scroll(sender As Object, e As
EventArgs) Handles trbMountainFrequency.Scroll
'Update the value of the map setting variable as track bar moves and
update the map setting feedback display:
intMountainFrequency = trbMountainFrequency.Value
If intMountainFrequency < 3 Then
pbMountainFrequency.Image = FormImgRef.pbLMountainous.Image

```

```
    Else
        pbMountainFrequency.Image = FormImgRef.pbVMountainous.Image
    End If
End Sub
```

```
Private Sub trbHillFrequency_Scroll(sender As Object, e As EventArgs)
Handles trbHillFrequency.Scroll
```

'Update the value of the map setting variable as track bar moves and update the map setting feedback display:

'Manipulate the value of intHillFrequency to facilitate the map setting in the context of the code of ModuleMapGeneration:

```
    intHillFrequency = trbHillFrequency.Value * 2 + 8
    If trbHillFrequency.Value > 6 Then
        pbHillFrequency.Image = FormImgRef.pbHilly.Image
    Else
        pbHillFrequency.Image = FormImgRef.pbFlat.Image
    End If
End Sub
```

```
Private Sub trbFloraFrequency_Scroll(sender As Object, e As EventArgs)
Handles trbFloraFrequency.Scroll
```

'Update the value of the map setting variable as track bar moves and update the map setting feedback display:

'Manipulate the value of intFloraProbability to facilitate the map setting in the context of the code of ModuleMapGeneration:

```
    intFloraProbability = 11 - trbFloraFrequency.Value
    If intFloraProbability < 4 Then
        pbFloraFrequency.Image = FormImgRef.pbFloraHigh.Image
    ElseIf intFloraProbability > 3 And intFloraProbability < 8 Then
        pbFloraFrequency.Image = FormImgRef.pbFloraMedium.Image
    Else
        pbFloraFrequency.Image = FormImgRef.pbFloraLow.Image
    End If
End Sub
```

```
Private Sub trbForestSize_Scroll(sender As Object, e As EventArgs)
Handles trbForestSize.Scroll
```

'Update the value of the map setting variable as track bar moves and update the map setting feedback display:

'Manipulate the value of intForestProbability to facilitate the map setting in the context of the code of ModuleMapGeneration:

```
    intForestProbability = 6 - trbForestSize.Value
```

```
    If trbForestSize.Value = 1 Then
```

```

pbForestSize.Image = FormImgRef.pbForestSmall.Image
ElseIf trbForestSize.Value = 2 Then
    pbForestSize.Image = FormImgRef.pbForestMedium.Image
Else
    pbForestSize.Image = FormImgRef.pbForestLarge.Image
End If
End Sub
End Class

```

Amended code for map generation:

As previously mentioned, as some of the variables required for map generation are now assigned by the user rather than randomly generated; some of the original code for the map generation module needed to be rewritten. Some of the changes simply required deleting assignment statements where one of the variables was assigned a random number; others required complete rewriting of the code. Such changes are shown below.

Originally the water feature that was to be generate was decided using an assignment statement with the random number generator function, it has now been replaced with:

```
If blnSeaLake = False Then
```

The code for the lake generation follows this selection statement, and the code for the sea generation follows the else section of the if statement.

The statement for randomly generating mountain anchor tiles was originally clumsy and has been rewritten. The new code is as follows:

```

For intArrXCoord = 1 To 54
    If FuncRandomNumberGen(1, 4) > 1 Then
        arrMapBlueprint(1, intArrXCoord).ptyTileType = "Mountain"
    End If
    Next
    'Select the mountain generation sequence that fits the user map setting
choice:
    Select Case intMountainFrequency
        Case 0
            subGenerateMountains(3, 1)
            subGenerateMountains(4, 2)

```

```

    subGenerateMountains(5, 3)
    subGenerateMountains(6, 4)
Case 1
    subGenerateMountains(2, 1)
    subGenerateMountains(3, 2)
    subGenerateMountains(4, 3)
    subGenerateMountains(5, 4)
Case 2
    subGenerateMountains(2, 1)
    subGenerateMountains(2, 2)
    subGenerateMountains(3, 3)
    subGenerateMountains(4, 4)
Case 3
    subGenerateMountains(1, 1)
    subGenerateMountains(2, 2)
    subGenerateMountains(2, 3)
    subGenerateMountains(3, 4)
Case 4
    subGenerateMountains(1, 1)
    subGenerateMountains(1, 2)
    subGenerateMountains(2, 3)
    subGenerateMountains(2, 4)
End Select

```

The lower the value of the first parameter passed to subGenerateMountains, the larger the mountain ranges that are generated. In this way as the value of intMountainFrequency increases, the larger the mountain ranges.

Conversely, the smaller the value of intForestProbability, the larger the size of the forests that are generated. This is because it was much more convenient to implement this way by using a random number generator with the user assigned intForestProbability as a parameter. Again, the code to decide whether forest anchor tiles was originally clumsy, and has also been replaced. This new code is shown below:

```

intProbability = FuncRandomNumberGen(1, intForestProbability)
Select Case intProbability

```

All other amendments to previous code in the map generation module involved deleting redundant code.

The final amendment was related to the print map procedure. Now that navigation has been implemented, the print map procedure needed to be able to take in different regions of the map as parameters and print the relevant part of the map to the screen. The new code is shown below.

'Subroutine that prints an entire map quadrant to the screen

```
Sub subPrintMap(ByVal intXLower As Integer, ByVal intXUpper As Integer,  
ByVal intYLower As Integer, ByVal intYUpper As Integer)
```

'Variables to locate the destination rectangle

```
Dim intYCoord, intXCoord As Integer
```

```
For intY = intYLower To intYUpper
```

```
    For intX = intXLower To intXUpper
```

'Translate the array index values to co-ordinates:

```
    intYCoord = (intY - intYLower) * 40
```

```
    intXCoord = (intX - intXLower) * 40
```

Now that all the amendments to previous code have been made, this section is complete.

Developing the solution (Testing to inform development):

The table below describes the testing process for this iteration.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
DD3.1	Test that the mini-map navigation works correctly.	User input clicking the four quadrants of the mini-map.	Clicking the four corners of the map will navigate the user to the respective quadrant of the map.	Yes
DD3.2	Testing the mini-map prints correctly.	trbWaterFeature set to lake; trbLakeWidth set to minimum value; trbLakeHeight set to maximum value; trbRiverBendiness set to minimum i.e. largest amount of meandering; trbMountainFrequency set to penultimate value; trbForestSize set to medium value; trbHillFrequency set to minimum; trbFloraFrequency set to second lowest;	The mini-map corresponds exactly to the generated map.	Yes
DD3.3	Check the date is calculated and is displayed correctly.	No user input, observe the in-game date for two in-game years: one leap year, one normal year.	The date is displayed in the top right display of the in-game user interface. The month icon is displayed correctly with the respective image.	Yes
DD3.4	Test that the map settings are reflected correctly in-game.	Same user inputs as DD3.1.		Yes

Evidence of testing commences on the following page.

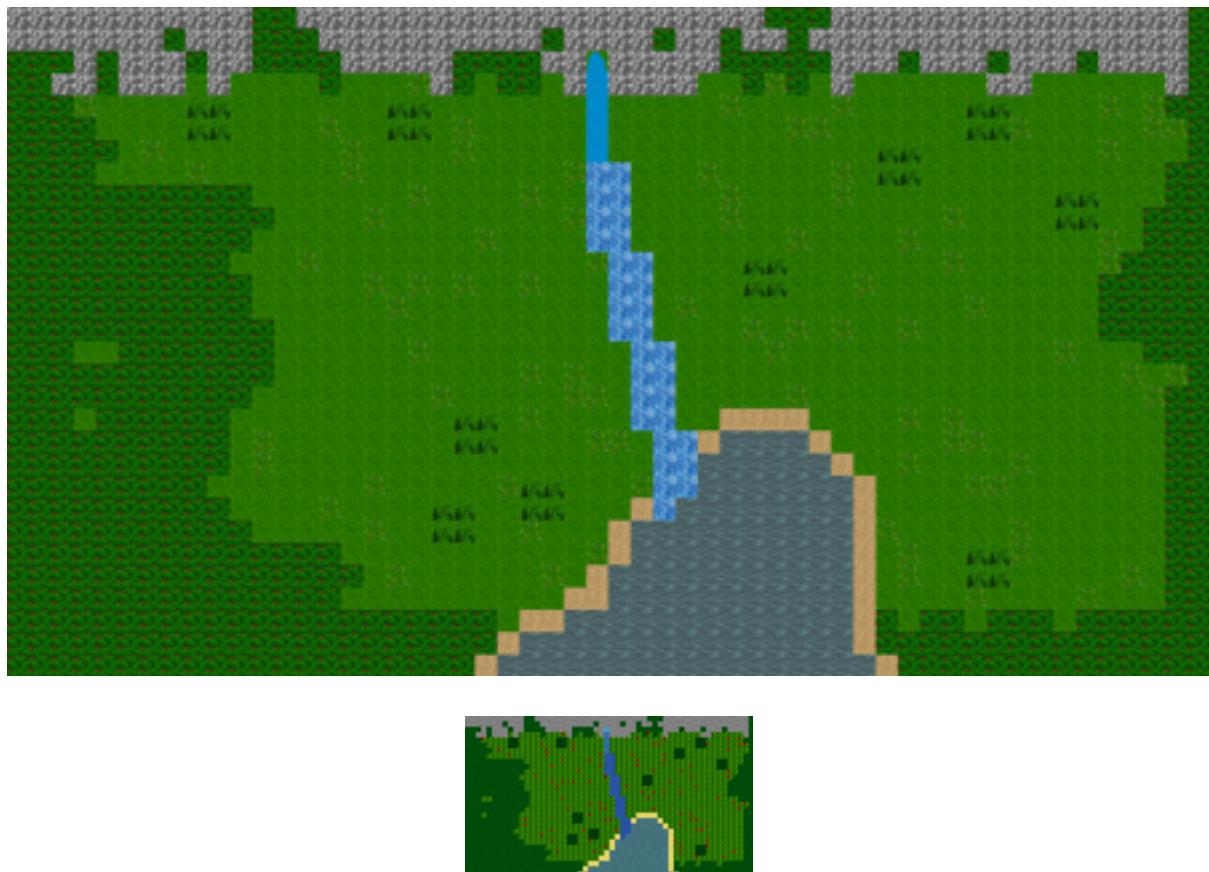
Evidence of testing:

DD3.1

This test was recorded using Grabilla, and the test can be viewed in the file named DD3.1.mp4. This video demonstrates the mini-map navigation working as expected and so this test is a success.

DD3.2

A map was generated according to the settings specified in the test data and the resulting map was compared to the visual provided by the mini-map. Images of both the map that was generated and the mini-map that was created are both shown below. The two are identical in terms of terrain appearance and so this test is a success.



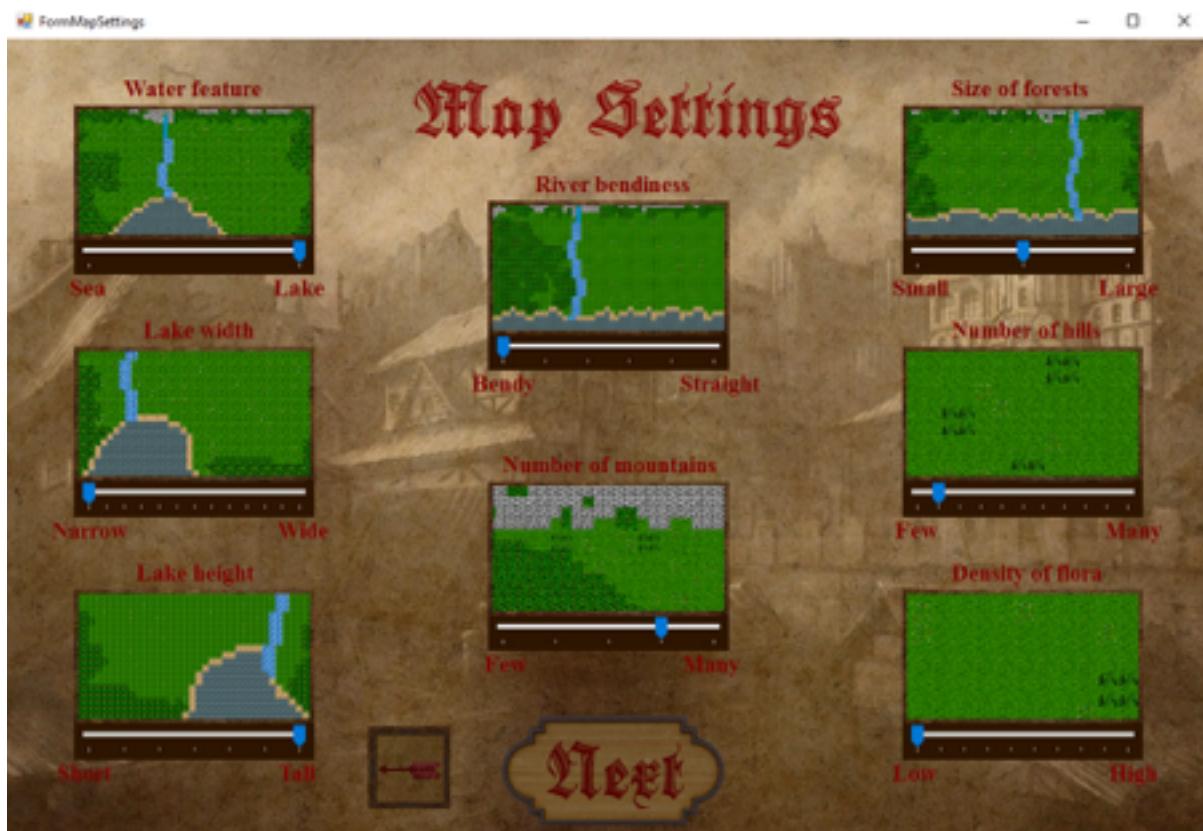
DD3.3

This test was recorded using Grabilla, and the test can be viewed in the file named DD3.3.mp4. This video demonstrates the date being calculated correctly. The year 1296 was a leap year and this is correctly shown. The year 1297 was not a leap year and this is also

correctly shown. The in-game timer tick delay was reduced from 1000 milliseconds to 50 milliseconds to reduce the length of the video. As all the months are correctly calculated, this test is a success.

DD3.4

I decided to perform this test with my stakeholders to see if they believed that this test is success (whether or not the map settings are reflected in the map generation). The proof of the settings used in this test is shown below



The map that was generated is shown in an image on the following page. My stakeholders unanimously agreed that trivial settings such as the lake being generated with the expected dimensions; the river bendiness; the density of flora and the frequency of hills were all correctly reflected in the map generation. They also unanimously agreed that the forests are of a sizes that they would expect from selecting the medium forest size. However one of my stakeholders disagreed with the other two when they agreed that the size of the mountain ranges are of the expected size. The stakeholder said she expected larger mountain ranges from the chosen setting, however as the map generation is ultimately random, she agreed with the majority and allowed the test to pass. Based on the consensus of my stakeholders opinion,



this test is a success.

Stakeholder review and sign-off of iteration

Directly after the final test was completed, I interviewed my stakeholder to determine whether they agreed that the iteration requirements had been met. All my stakeholders thought that the program in its current form fulfils all the requirements of this iteration and have allowed me to continue to the next iteration.

END OF THIRD ITERATION

FOURTH ITERATION

Discussion with Stakeholders

As time is limited, I have agreed with my stakeholders to conglomerate the contents of iterations four and five together, as such this iteration will implement both treasury/income information and building placement.

I presented my stakeholders with certain types of information that could appear on the treasury menu and asked them what they think the treasury menu should contain. We decided that detailed information about gold income and resource stockpiles should be displayed here. This menu will purely be a read only menu for information about the town. From the analysis of the problem my stakeholders said they would prefer over 10 unique resources including gold. I consulted my stakeholders on which resources they believed should be included. As time is limited I will implement 11 of the resources we discussed including gold. The other 10 resources will be: wood, stone, iron, barley, wheat, oats, wool, beef, fish and ale. These resources will all be displayed in the treasury menu.

Building placement will involve calculating if the user can afford the building, allowing the user to click on the map to place the building and finally deducting the resources once the building is placed. In order to fulfil the success criteria of the project I will implement 11 of the buildings. More may be implemented in later versions. I referred to my analysis of the problem and discussed with my stakeholders which 11 buildings should be implemented in this iteration and the following were chosen: cruck hut, farm, lumberjack hut, pasture, fishery, brewery, reeve house, water mill (all serf buildings); cottage, quarry (both freemen buildings) and a mine (yeomen building). We agreed on these buildings as they will allow the implementation of resource gathering next iteration. The unique buildings of each civilisation will also be displayed upon choosing the civilisation, but in this version they will not be accessible.

Specify the problem: requirements of resources and building placement

The iteration must fulfil the following requirements:

- R4.1.** The treasury menu must include detailed information about gold income and resource stockpiles.
- R4.2.** The following resources must be implemented: wood, stone, iron, barley, wheat, oats, wool, beef, fish and ale.
- R4.3.** The user must be able to place the following 10 building types: cruck hut, farm, lumberjack hut, pasture, fishery, brewery, reeve house, cottage, quarry and mines.

- R4.4.** Hovering over the placable buildings must display the resource and cost requirements of the building.
- R4.5.** Clicking a building button must check whether the user has the requisite resources to construct the building and return an error message if the user does not meet the requirements.
- R4.6.** If the user has the correct amount of resources when clicking a building button, the program must provide guidance to help the user place the building, e.g. the tile being hovered over with the mouse cursor would change colour.
- R4.7.** Placing a building should deduct the correct amount of resources from the treasury and print the building to the screen.

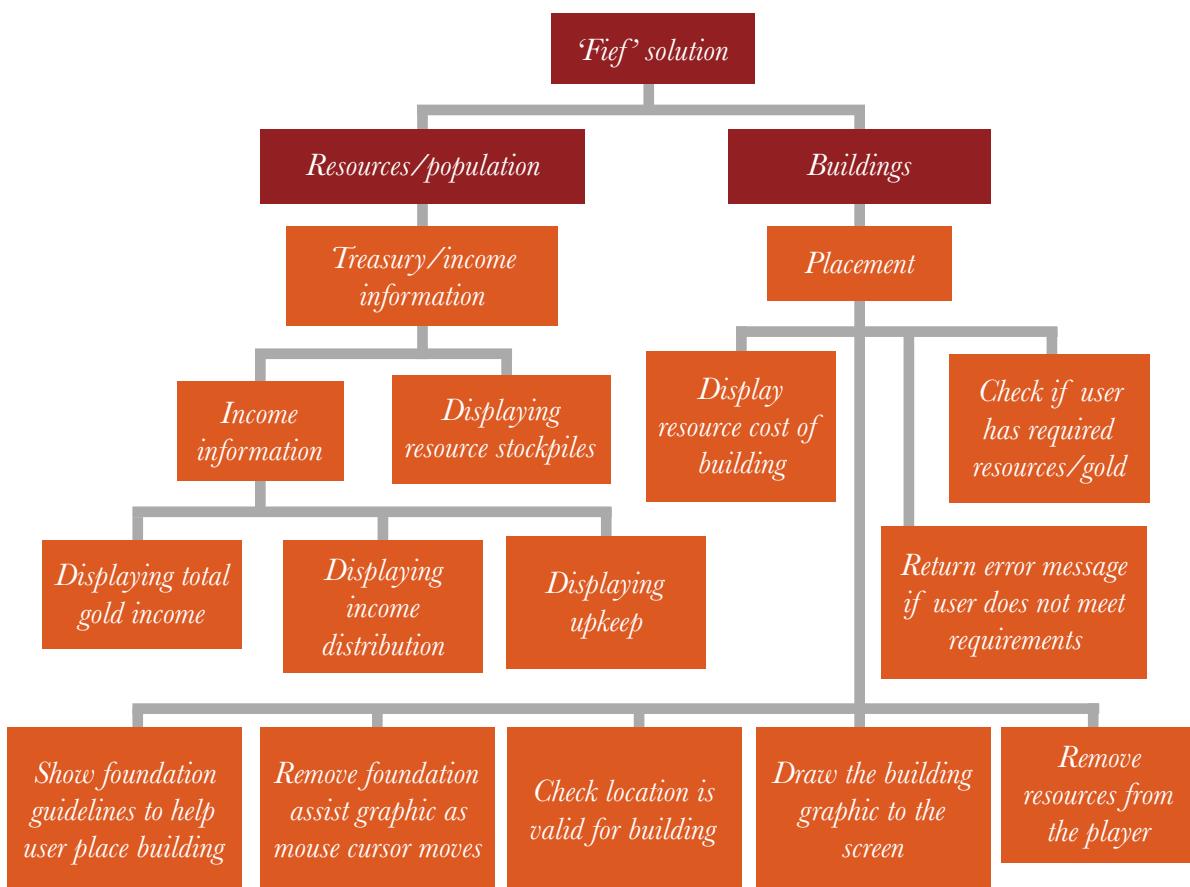
Success criteria and requirements the iteration addresses:

Success Criteria	Requirement	How success criteria requirement is addressed
SC2	SCR2.1	A unique building will appear on the building menu depending upon the civilisation.
SC11	SC11.1	10 unique buildings other than the civilisation buildings will be implemented.
SC11	SC11.3	Certain buildings will be prevented from being placed in certain locations.
SC11	SC11.4	Inverse of SC11.3: Certain buildings must be allowed to be placed in certain locations. See note 1.
SC12	SC12.1	There must be 3 levels of civilisation: peasant, freemen and yeomen.

Decomposing the Problem

On the following page is a diagram illustrating the decomposition of the problems presented in this iteration. As this iteration implements the features of two iterations, these are broken down separately. The treasury/income menu is broken down into two main parts: the income information and the resource stockpile display. The resource stockpile display will simply display how much of each resource the user has; whilst the income display is broken down into three main displays: the total gold income, the income distribution and the upkeep information. These displays will be tackled individually. The building placement is much more complicated. The cost of the building needs to be displayed before the user builds the building; the cost of the building needs to be compared to the amount of resources/gold the user has and the user must be prevented from constructing a building if the prerequisite

resources are not stockpiled. The program needs to show to the user the size of the building when placing to aid the user in placing the building, any graphic used to help the user in this way needs to be cleared as the user moves the mouse cursor. Once the building is placed, the location needs to be checked to ensure it is suitable for the building, if the location is suitable, the building needs to be drawn to the map and the map array needs to save the building to the map-tile building type property and finally remove the resources from the user required to build the building. These elements of placing the building will all be tackled individually as shown in the diagram below.

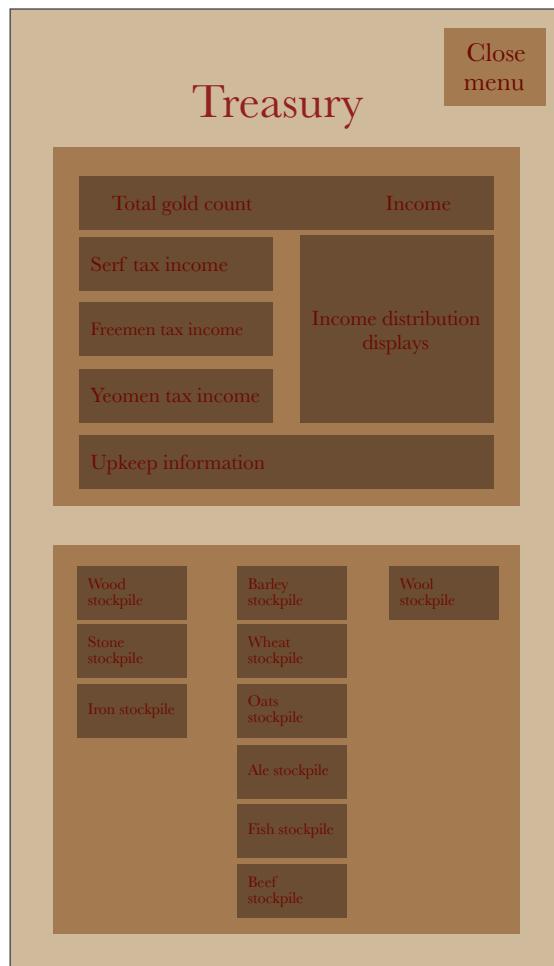


Describe the solution (Usability features)

The treasury menu design will need to be overhauled to accommodate the information that is to be displayed on it. As the treasury menu purely displays information, the treasury menu will be composed of mostly labels. A wireframe design of the new treasury menu is shown on the following page. The treasury is split into the two main areas as described in the decomposition of the problem section: the income information and the resource stockpiles section. The stockpiles of the 10 non-gold resources are shown with labels next to icons

illustrating the resource stockpile the label represents.

The total gold count is shown at the top of the top half with a label representing income beside it. Directly below the incomes from each civilisation level are shown with a diagram illustrating the distribution of revenue from the population. Below that are three labels displaying the three types of upkeep that deduct from tax profits: cost of running resource collecting buildings; cost of running public and administrative buildings and military upkeep costs. These are illustrated with icons.



There has been space left in the resource stockpile section for any future resources that may be added in later versions of the game.

The icons I have designed for each of the resources are designed to be unique and recognisable so that the user can easily understand which resource they illustrate. These icons are shown in a table on the following page describing what each resource is intended to be used for.

Icon	Resource	Use of resource:
	Wood	Fundamental construction material. Required to build all buildings.
	Stone	Important construction material. Used to build mid-late game buildings.
	Iron	Used to make weapons and tools. Also used in late game construction.
	Barley	Crop used to produce ale.
	Wheat	Crop used in food production.
	Oats	Crop used in food production.
	Ale	Consumed by population.
	Fish	Consumed by mid-game populations.
	Beef	Consumed by late-game populations.
	Wool	Used in production of clothing for population.

As described previously during the discussion with stakeholders, this version of the game will only implement fundamental resources as time is limited. In later versions more sophisticated resources will be added.

Upkeep is a game mechanic that will be designed to increase game difficulty and force the user to expand the town in a way that the population buildings will be proportional to the number of resource collecting/public buildings. Without such a mechanic the user could build many such buildings without any repercussions and each game would be very short.

Each building will require different amounts of gold and resources to build, with cost increasing as the value of the building increases. The image of the button from the building menu is shown below with the costs of each building. These may need to be changed in later versions to balance the game correctly.

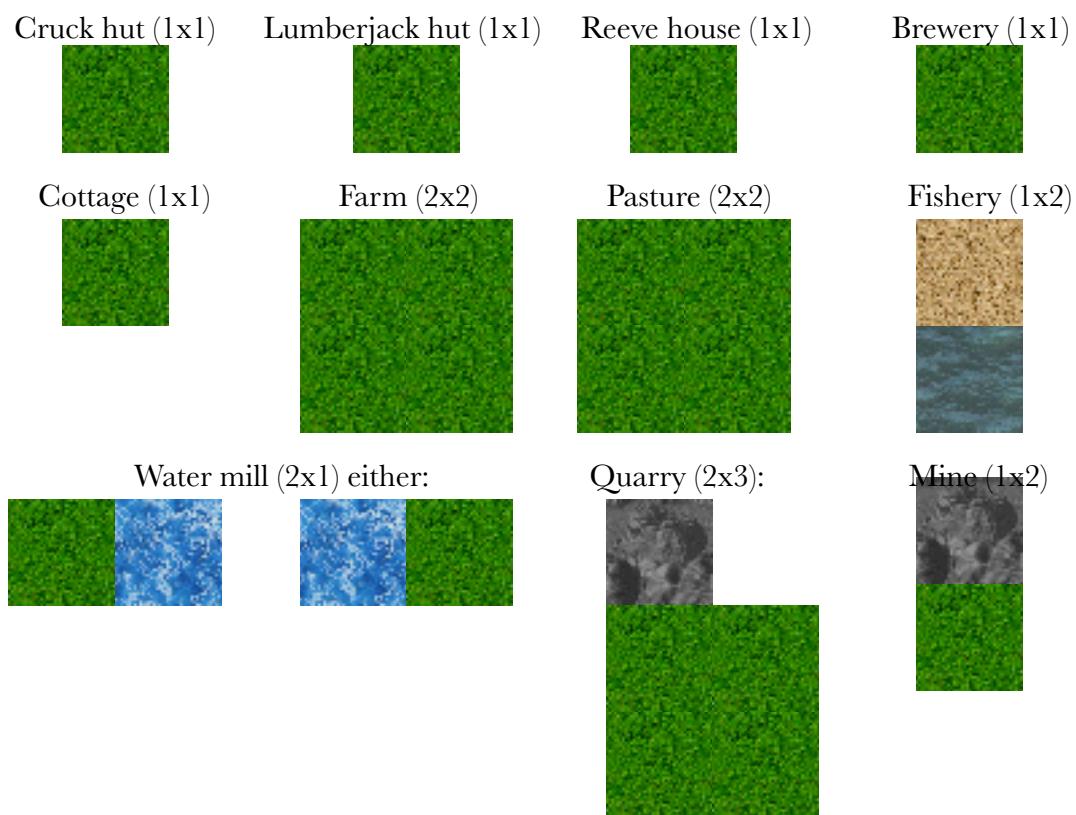
Icon	Building	Wood cost	Stone cost	Gold cost
	Cruck hut		5	10
	Farm		15	100
	Water-mill		50	150
	Lumberjack hut	10		50
	Reeve house		50	20
	Pasture		25	250
	Fishery		20	150
	Brewery		20	250
	Cottage	2	3	200
	Quarry		100	500
	Mine	200	100	750

Iron is described on the previous page as a construction material in late-game buildings, no such building will be implemented in this version, thus it does not appear in this table.

The function of each building is as follows:

- Cruck hut: Houses serfs;
- Farm: Grow barley, wheat or oats;
- Water-mill: Used to process grain crops;
- Lumberjack hut: Collects wood from adjacent forest tiles;
- Reeve house: Unlocks the management menu and allows the periodic removal of forest tiles;
- Pasture: Used to graze livestock (sheep or cattle);
- Fishery: Collects fish from the adjacent sea tiles;
- Brewery: Produces ale from barley;
- Cottage: Houses freemen;
- Quarry: Mines stone;
- Mine: Mines iron.

Each building can only be placed on certain types of terrain. The diagrams below use terrain tiles to illustrate the configuration of terrain that each building is allowed to be placed upon. The size of each building is also shown.



The tile type of the top right tile of the quarry does not matter for its placement.

The flora tile type should be identical in functionality to the plains tile type, but this element of flora tiles will be added in a future version.

The graphics of the building are shown below. These may be subject to change in later versions, but for this version these graphics will suffice.

Cruck hut:



Lumberjack hut:



Reeve house:



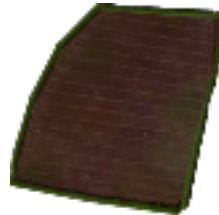
Brewery:



Cottage:



Farm:



Pasture:



Fishery:



Water mill either of:



Quarry:



Mine:



When choosing a location for the building, the cursor will be accompanied by a maroon rectangle that indicates the size of the building to be placed. The resources required to place a building will be removed from the user upon placement of a building, however the resources and gold count will be updated every game tick (once per second). The user will only be able to place one building at a time as a validation check needs to be done every time a building button is clicked. This validation is described in greater detail in the following section. If the user does not have the required materials, text will appear in a pop-up message box indicating this.

Describe the solution (Key variable, data structures, classes and validation checks)

Some new variables are required to be used to keep track of the various resource stockpiles, gold count/income and to place the buildings. These key variables are shown in the table below with a justification for each.

Variable Name	Variable Type	Justification
strCurrentBuilding	String	Required so that the solution knows which building is to be placed when the user performs a mouse click.
blnBuildingSelected	Boolean	Required so that the solution knows whether a building is to be placed upon a mouse click.
strCurrentQuadrant	String	Required so that the program knows which quadrant of the map is being interacted with at any point.
intMouseX	Integer	Stores the X-co-ordinate of the mouse location so that the program can calculate which tile is being interacted with.
intMouseY	Integer	Stores the Y-co-ordinate of the mouse location so that the program can calculate which tile is being interacted with.
intPrevMouseX	Integer	Stores the X-co-ordinate of the last mouse location so that the program can remove any redundant textures from the most recent tile.
intPrevMouseY	Integer	Stores the Y-co-ordinate of the last mouse location so that the program can remove any redundant textures from the most recent tile.
intGoldCount	Integer	Required to store how much gold the user has.
intWoodStockpile	Integer	Required to store how much wood the user has.
intStoneStockpile	Integer	Required to store how much stone the user has.
intIronStockpile	Integer	Required to store how much iron the user has.
intBarleyStockpile	Integer	Required to store how much barley the user has.
intWheatStockpile	Integer	Required to store how much wheat the user has.
intOatStockpile	Integer	Required to store how many oats the user has.
intWoolStockpile	Integer	Required to store how much wool the user has.
intBeefStockpile	Integer	Required to store how much beef the user has.
intFishStockpile	Integer	Required to store how much fish the user has.
intAleStockpile	Integer	Required to store how much ale the user has.
intXLower	Integer	Used with intMouseX to calculate the array value of a tile location according to the current quadrant.
intYLower	Integer	Used with intMouseY to calculate the array value of a tile location according to the current quadrant.

No new data structures or classes are introduced in this iteration, however there are two significant validation checks that will be implemented in this iteration.

The first is a check that the user has the required resources when attempting to click a building button, this is important to ensure that the user can afford the building. This validation check will compare the values of the cost of the building, to the amount of the relevant resources the user has.

The second is a check when the building is placed to ensure that the location is suitable for the building. This validation check is required to satisfy the success criteria requirements 11.3 and 11.4. This will occur when the user performs a mouse click whilst `blnBuildingSelected = true`, and will compare the tile being clicked to the acceptable tiles.

Approach to testing (Usability features)

In order to ensure the project meets the success criteria, my stakeholders will perform usability tests comparing the solution to the success criteria this iteration addresses and determine whether the success criteria have been correctly addressed.

Feature number	Usability test description
UF4.1	Testing that each civilisation has access to a unique building.
UF4.2	Test that the user has access to, and can build at least 10 different types of building.
UF4.3	Testing that each level of civilisation has a set of buildings that can be placed.

Approach to testing (During development)

This iteration will be testing two implements: the treasury menu and the building placement. The testing related to the treasury menu is trivial: testing that the resources are correctly updated every game tick; however many aspects of building placement need to be tested: whether the validation checks are successful and whether the graphics are correctly drawn to the map. A table below describes these test that will be performed to help inform development.

Test number	Description of test	Expected Result
DD4.1	Test that the resource stockpiles and gold count in the treasury menu are updated every game tick.	Each resource stockpile value has the correct value assigned to respective labels each game tick.
DD4.2	Test that the user is prevented from clicking a building button if the user lacks sufficient resources.	The user is prevented from building the building and a message box describes what the user is missing.
DD4.3	Test that clicking a building button with the correct amount of resources creates a rectangle anchored to the cursor that shows the building size.	A rectangle with the dimensions of the building to be placed is drawn around the mouse cursor and is removed as the mouse cursor moves.
DD4.4	Test that clicking the map while a building is selected in the incorrect location does not place a building.	No building is placed.
DD4.5	Test that clicking the map while a building is selected in the correct location places the building and removes the resources required to build the building from the user.	A building is drawn to the map and the correct amount of resources are removed from the user.

Approach to testing (Post development)

My stakeholders will perform post development tests to test robustness and to ensure that this iteration fulfils the success criteria requirements this iteration addresses. The tests are shown in the table on the following page with descriptions of the tests and expected results.

Test number	Description of test	Expected Result
PD4.1	Test that each civilisation has the correct unique building available.	The Yuan Dynasty civilisation has access to terraced farms in the serfs section; the English have access to fletchers in the freemen section and the Seljuk Turks have access to spice markets in the yeomen section.
PD4.2	Test that at least 10 buildings are available to the user that can be placed.	The user can place at least 10 individual buildings.
PD4.3	Test that certain buildings are prevented from being placed in certain locations.	Attempting to place certain buildings in certain locations fails.
PD4.4	Test that certain buildings are allowed to be placed in certain locations.	Attempting to place certain buildings in certain locations succeeds.

Test number	Description of test	Expected Result
PD4.5	Test that there are buildings unique to the three levels of civilisation.	There is at least one building available to each level of civilisation.

Describe the solution (Pseudo-code algorithms)

(PC4.1) Pseudo-code to move the foundation guide rectangle with the mouse cursor.

Procedure MouseMove //MouseMove is an event that runs every time the mouse cursor is moved

```

Dim XLower As Integer
Dim YLower As Integer
MouseX = RoundDown(MouseXCoordinate / 40)
MouseY = RoundDown(MouseYCoordinate / 40)
//creating mouse co-ordinates that are based upon tiles, rather than pixels
If MouseX != PrevMouseX Or MouseY != PrevMouseY Then
    //If statement checks if the mouse cursor has moved to a different tile
    Select Case CurrentQuadrant
        Case "Top Left"
            XLower = 1
            YLower = 1
        Case "Top Right"
            XLower = 28
            YLower = 1
        Case "Bottom Left"
            XLower = 1
            YLower = 16
        Case "Bottom Right"
            XLower = 28
            YLower = 16
    End Select Case
    //Select case assigns values to the XLower, YLower variables in
    depending upon the current quadrant being viewed so that the array co-
    ordinates can be correctly referenced
    Print tiles in a 2x3 rectangle centred above and right of the mouse cursor

```

```

//This will be accomplished using a call to a new procedure which is
shown in PC4.6. This has the effect of clearing any previous foundation
guide graphics.

End if

PrevMouseX = MouseX

PrevMouseY = MouseY

//Updates the values of the previous mouse co-ordinate variables in preparation
for the next time the mouse cursor moves to another tile

If BuildingSelected = True Then

    //Will only draw a foundation rectangle if a building is selected.

    Select Case CurrentBuilding

        Case "Cruck Hut"

            MapGraphics.DrawRectangle(Colour.Maroon, MouseX * 40,
                                      MouseY * 40, CruckHutWidth, CruckHutHeight)

            //This draws a maroon rectangle on this current tile the mouse
            cursor is hovered over with dimensions corresponding to the size
            of the building that is to be placed, in this case, a cruck hut

        Case ... //Individual cases with corresponding dimensions for each
                  building

    End Select Case

End If

End Procedure

```

(PC4.2) Pseudo-code to place a selected building.

Procedure MouseClick //MouseMove is an event that runs every time the user performs
a mouse click

```

Dim XLower As Integer
Dim YLower As Integer
Select Case CurrentQuadrant
    Case "Top Left"
        XLower = 1
        YLower = 1
    Case "Top Right"
        XLower = 28
        YLower = 1
    Case "Bottom Left"

```

```

    XLower = 1
    YLower = 16
Case "Bottom Right"
    XLower = 28
    YLower = 16
End Select Case
//Select case performs identical task to the CurrentQuadrant select case in
PC4.2
If BuildingSelected And MapBlueprint(MouseY + YLower, MouseX +
XLower).BuildingType = "Empty" Then
//Checks that a building is selected and that the target location is empty before
attempting to place a building
    Select Case CurrentBuilding
        Case "Cruck Hut"
            If MapBlueprint(MouseY + YLower, MouseX +
XLower).TileType = "Plains" Then
                MapBlueprint(MouseY + YLower, MouseX +
XLower).BuildingType = "Cruck Hut"
                Print the building to the map
                //Using the procedure described by PC4.6.
                WoodStockpile = WoodStockpile - WoodCost
                GoldCount = GoldCount - GoldCost
                //Removes the resources required to build the building
                from the user
                BuildingSelected = False
                //Forces the user to make another validation check before
                placing another building so that the user cannot place any
                buildings they cannot afford.
            End If
            Case ... //Similar cases for each building checking the appropriate
            region for the correct tile types; drawing the building and removing the
            correct amount of resources from the user
        End Select Case
    End If
End Procedure

```

(PC4.3) Pseudo-code to select a building.

```
Procedure BuildingButton.Click
    If CheckCost(WoodCost, StoneCost, IronCost, GoldCost) = True Then
        //CheckCost is a function described in PC4.4.
        CurrentBuilding = CorrespondingBuilding
        //Assign the CurrentBuilding variable the value associated with the
        building related to the button that has been clicked
        BuildingSelected = True //So that the building will be placed when the
        user performs a mouse click in an acceptable region
    End If
End Procedure
```

(PC4.4) Pseudo-code to check the cost of a building compared to the resources the user has.

```
Function CheckCost(Val WoodCost As Integer, Val StoneCost As Integer, Val
IronCost As Integer, Val GoldCost As Integer)
    If WoodCost > WoodStockpile Then
        MessageBox.Show("You require more wood")
        Return False
    End If
    //Prevents the user from clicking a building button if the resource requirements
    are not met
    //Similar if statements will be used for the three other resources
    If all resource stockpiles > resource costs Then
        Return True
        //Allows the user to place a building if all costs are met
    End If
End Procedure
```

(PC4.5) Algorithm to display the cost of a building when the building's button is hovered over with the mouse cursor.

```
Procedure BuildingButton.MouseHover
    FormInGame.Resource1.Image = Wood icon
    FormInGameResource1.Labal = Wood cost of building
    FormInGame.Resource2.Image = Stone icon
    FormInGameResource2.Labal = Stone cost of building
    FormInGame.Resource3.Image = Gold icon
```

```

FormInGameResource2.Labal = Gold cost of building
//This will utilise the picture boxes in the information section of the in-game UI
to display the cost of a resource next to the corresponding icon so that the user
knows which resource is being referred to

```

(PC4.6) Algorithm to print a single map tile to the screen

```

Procedure PrintTile(ByVal XCoord As Integer, ByVal YCoord As Integer, ByVal XLower
As Integer, ByVal YLower As Integer)
//Uses Coord and Lower parameters to calculate the correct location in the
MapBlueprint array
Dim X, Y As Integer
Y = YCoord * 40
X = XCoord * 40
//Converts the tile co-ordinates to pixel co-ordinates
DestinationRectangle = New Rectangle(X, Y, 40, 40)
SourceRectangle = New Rectangle(0, 0, 40, 40)
//First print the tile type graphic to the screen
Select Case MapBlueprint(MouseY + YLower, MouseX + XLower).TileType
Case "Plains"
    Bitmap = New Bitmap(FormImgRef.PlainsGraphic.Image)
    MapGraphics.DrawImage(Bitmap, DestinationRectangle, SourceRectangle,
    Graphics.Pixel)
    //GraphicsUnit.Pixel is a method of printing graphics that prints the graphics
    pixel-by-pixel that I will use in the program, this has been mentioned previously
    in PC2.13
Case ... //Similar cases for each tile type
End Select
//Now print the building graphic over the terrain graphic
Select Case MapBlueprint(MouseY + YLower, MouseX +
    XLower).BuildingType
Case "Cruck Hut"
    Bitmap = New Bitmap(FormImgRef.CruckHutGraphic.Image)
    MapGraphics.DrawImage(Bitmap, DestinationRectangle, SourceRectangle,
    Graphics.Pixel)

```

```

Case ... //Similar cases for each building type
End Select
End Procedure

```

The table below describes how each of these pseudo-code algorithms form part of the solution and which requirements each addresses.

Pseudo-code algorithm	Description of how algorithm forms part of solution	Success criteria/requirements fulfilled	Justification of how algorithms fulfils criteria
PC4.1	Will help guide the user when placing a building. Provides information on the size of the building to the user.	R4.6	Algorithm provides building guidance to the user as described in R4.6
PC4.2	Allows the user to place buildings.	R4.3 R4.7	Algorithm allows for the placement of the 10 buildings listed in R4.3. Algorithm removes the resources required to build a building.
PC4.3	Allows the user to choose a building to be placed.	R4.3 R4.5	Algorithm allows the user to choose any of the 10 buildings implemented in this version. Uses validation check to prevent a building being built without the required resources.
PC4.4	Prevents the user from placing a building if they cannot afford it and allows them to if they can.	R4.5	Performs a validation check to ensure the user has the required resources to build the selected building.
PC4.5	Informs the user how much a building costs so that they know whether or not they can afford it.	R4.4	Algorithm displays the cost of the target building.
PC4.6	Draws the building to the map.	R4.3	By drawing the building to the map, the user gets a sense that they are building a town from the 10 buildings in this version.

Developing the solution (Coding)

The code to implement the new treasury menu and the resources is very trivial, and was coded first as the buildings require resources to be built, so it seemed only logical to code this first.

Code related to implementing the resources:

The resource variables are defined as public variables in the FormInGame class. They are given values once the settle town button is clicked in the town name form. These values will need to be balanced later with how many buildings the user should initially be allowed to build. For now they have been very large numbers so that I can test the building placement. The code defining these variables is as follows:

```
Public intGoldCount, intWoodStockpile, intStoneStockpile, intIronStockpile,  
intBarleyStockpile, intWheatStockpile, intOatStockpile, intWoolStockpile,  
intBeefStockpile, intFishStockpile, intAleStockpile As Integer
```

The code assigning these variables values is as follows:

'Sets the resource stockpiles to the default levels:

```
FormInGame.intGoldCount = 1000  
FormInGame.intWoodStockpile = 1000  
FormInGame.intStoneStockpile = 1000  
FormInGame.intIronStockpile = 1000  
FormInGame.intBarleyStockpile = 1000  
FormInGame.intWheatStockpile = 1000  
FormInGame.intOatStockpile = 1000  
FormInGame.intWoolStockpile = 1000  
FormInGame.intBeefStockpile = 1000  
FormInGame.intFishStockpile = 1000  
FormInGame.intAleStockpile = 1000
```

The code updating the labels for these resources in the treasury menu every game tick is as follows:

```
lblWealth.Text = intGoldCount  
FormTreasuryMenu.lblTreasuryGold.Text = intGoldCount  
FormTreasuryMenu.lblWoodStockpile.Text = intWoodStockpile  
FormTreasuryMenu.lblStoneStockpile.Text = intStoneStockpile  
FormTreasuryMenu.lblIronStockpile.Text = intIronStockpile  
FormTreasuryMenu.lblBarleyStockpile.Text = intBarleyStockpile  
FormTreasuryMenu.lblWheatStockpile.Text = intWheatStockpile  
FormTreasuryMenu.lblOatStockpile.Text = intOatStockpile  
FormTreasuryMenu.lblAleStockpile.Text = intAleStockpile
```

```
FormTreasuryMenu.lblFishStockpile.Text = intFishStockpile  
FormTreasuryMenu.lblBeefStockpile.Text = intBeefStockpile
```

Code related to buildings buttons in the building menu:

This code is all within the FormBuildingMenu Class. This code is all related to the click events on the button. A validation check on the resources is performed.

```
Private Sub btnCruckHut_Click(sender As Object, e As EventArgs) Handles  
btnCruckHut.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(5, 0, 0, 10) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Cruck Hut"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnFarm_Click(sender As Object, e As EventArgs) Handles  
btnFarm.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(15, 0, 0, 100) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Farm"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnFishery_Click(sender As Object, e As EventArgs) Handles  
btnFishery.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(20, 0, 0, 150) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Fishery"
```

```
    FormInGame.blnBuildingSelected = True  
End If  
End Sub
```

```
Private Sub btnWaterMill_Click(sender As Object, e As EventArgs) Handles  
btnWaterMill.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(50, 0, 0, 150) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
    FormInGame.strCurrentBuilding = "Water Mill"
```

```
    FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnQuarry_Click(sender As Object, e As EventArgs) Handles  
btnQuarry.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(100, 0, 0, 500) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
    FormInGame.strCurrentBuilding = "Quarry"
```

```
    FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnPasture_Click(sender As Object, e As EventArgs) Handles  
btnPasture.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(25, 0, 0, 250) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
    FormInGame.strCurrentBuilding = "Pasture"
```

```
    FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnLumberjackHut_Click(sender As Object, e As EventArgs) Handles btnLumberjackHut.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(10, 0, 0, 50) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Lumberjack Hut"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnCottage_Click(sender As Object, e As EventArgs) Handles btnCottage.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(2, 3, 0, 200) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Cottage"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnReeve_Click(sender As Object, e As EventArgs) Handles btnReeve.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(50, 20, 0, 300) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Reeve House"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnBrewery_Click(sender As Object, e As EventArgs) Handles btnBrewery.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(20, 0, 0, 250) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Brewery"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnMine_Click(sender As Object, e As EventArgs) Handles btnMine.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(200, 100, 0, 750) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Mine"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

The following function performs the validation check required to ensure the user can afford the building they have selected.

```
Private Function SubCheckBuildingCost(ByVal intWoodCost As Integer, ByVal intStoneCost As Integer, ByVal intIronCost As Integer, ByVal intGoldCost As Integer)
```

'Check wood cost against wood stockpile:

```
If FormInGame.intWoodStockpile < intWoodCost Then
```

```
    MessageBox.Show("You require more wood")
```

```
    Return False
```

```
End If
```

'Check stone cost against stone stockpile:

```
If FormInGame.intStoneStockpile < intStoneCost Then
```

```
    MessageBox.Show("You require more stone")
```

```
    Return False
```

```
End If
```

'Check iron cost against iron stockpile:

```
If FormInGame.intIronStockpile < intIronCost Then
```

```

    MessageBox.Show("You require more iron")
    Return False
End If
'Check gold cost against gold count:
If FormInGame.intGoldCount < intGoldCost Then
    MessageBox.Show("You require more funds")
    Return False
End If
'Check that the user has the required resources:
If FormInGame.intWoodStockpile >= intWoodCost And
FormInGame.intStoneStockpile >= intStoneCost And
FormInGame.intIronStockpile >= intIronCost And FormInGame.intGoldCount
>= intGoldCost Then
    Return True
End If
End Function

```

The next section of code displays the cost of each building when a mouse hover event is triggered on the respective buttons of each building. This section also includes the code to remove these displays when the mouse cursor leaves the button.

```

Private Sub btnCruckHut_MouseHover(sender As Object, e As EventArgs)
Handles btnCruckHut.MouseHover
    'Displays the cost of the building in the information tab on the in-game
    screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "5"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "10"
End Sub
Private Sub btnCruckHut_MouseLeave(sender As Object, e As EventArgs)
Handles btnCruckHut.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
    screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

```

```

Private Sub btnFarm_MouseHover(sender As Object, e As EventArgs)
Handles btnFarm.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "15"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "100"
End Sub
Private Sub btnFarm_MouseLeave(sender As Object, e As EventArgs)
Handles btnFarm.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

Private Sub btnLumberjackHut_MouseHover(sender As Object, e As EventArgs)
Handles btnLumberjackHut.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "10"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "50"
End Sub
Private Sub btnLumberjackHut_MouseLeave(sender As Object, e As EventArgs)
Handles btnLumberjackHut.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

Private Sub btnCottage_MouseHover(sender As Object, e As EventArgs)
Handles btnCottage.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image

```

```
FormInGame.lblResource1.Text = "2"
```

```
FormInGame.pbResource2.Image = FormImgRef.pbStoneIcon.Image
```

```
FormInGame.lblResource2.Text = "3"
```

```
FormInGame.pbResource3.Image = FormImgRef.pbGoldIcon.Image
```

```
FormInGame.lblResource3.Text = "200"
```

```
End Sub
```

```
Private Sub btnCottage_MouseLeave(sender As Object, e As EventArgs)  
Handles btnCottage.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
```

```
FormInGame.lblResource1.Text = " "
```

```
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
```

```
FormInGame.lblResource2.Text = " "
```

```
FormInGame.pbResource3.Image = FormImgRef.pbEmpty.Image
```

```
FormInGame.lblResource3.Text = " "
```

```
End Sub
```

```
Private Sub btnBrewery_MouseHover(sender As Object, e As EventArgs)  
Handles btnBrewery.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
```

```
FormInGame.lblResource1.Text = "20"
```

```
FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
```

```
FormInGame.lblResource2.Text = "250"
```

```
End Sub
```

```
Private Sub btnBrewery_MouseLeave(sender As Object, e As EventArgs)  
Handles btnBrewery.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
```

```
FormInGame.lblResource1.Text = " "
```

```
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
```

```
FormInGame.lblResource2.Text = " "
```

```
End Sub
```

```
Private Sub btnQuarry_MouseHover(sender As Object, e As EventArgs)  
Handles btnQuarry.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
```

```
FormInGame.lblResource1.Text = "100"
```

```

FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
FormInGame.lblResource2.Text = "500"
End Sub
Private Sub btnQuarry_MouseLeave(sender As Object, e As EventArgs)
Handles btnQuarry.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

Private Sub btnReeve_MouseHover(sender As Object, e As EventArgs)
Handles btnReeve.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "50"
    FormInGame.pbResource2.Image = FormImgRef.pbStoneIcon.Image
    FormInGame.lblResource2.Text = "20"
    FormInGame.pbResource3.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource3.Text = "300"
End Sub
Private Sub btnReeve_MouseLeave(sender As Object, e As EventArgs)
Handles btnReeve.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
    FormInGame.pbResource3.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource3.Text = " "
End Sub

Private Sub btnFishery_MouseHover(sender As Object, e As EventArgs)
Handles btnFishery.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "20"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image

```

```
    FormInGame.lblResource2.Text = "150"
End Sub
Private Sub btnFishery_MouseLeave(sender As Object, e As EventArgs)
Handles btnFishery.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
```

```
End Sub
```

```
Private Sub btnMine_MouseHover(sender As Object, e As EventArgs)
Handles btnMine.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "200"
    FormInGame.pbResource2.Image = FormImgRef.pbStoneIcon.Image
    FormInGame.lblResource2.Text = "100"
    FormInGame.pbResource3.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource3.Text = "750"
```

```
End Sub
```

```
Private Sub btnMine_MouseLeave(sender As Object, e As EventArgs)
Handles btnMine.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
    FormInGame.pbResource3.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource3.Text = " "
```

```
End Sub
```

```
Private Sub btnPasture_MouseHover(sender As Object, e As EventArgs)
Handles btnPasture.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "25"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "250"
```

```

End Sub
Private Sub btnPasture_MouseLeave(sender As Object, e As EventArgs)
Handles btnPasture.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

Private Sub btnWaterMill_MouseHover(sender As Object, e As EventArgs)
Handles btnWaterMill.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "50"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "150"
End Sub
Private Sub btnWaterMill_MouseLeave(sender As Object, e As EventArgs)
Handles btnWaterMill.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

```

The next section of code displays the mouse move event in the FormInGame class. This will draw the guidance graphic to the screen when the user is choosing a location for the building.

```

Private Sub pbMapDisplay_MouseMove(sender As Object, e As
MouseEventArgs) Handles pbMapDisplay.MouseMove
    'This subroutine checks that when the mouse cursor is moved to a
different tile the previous tile's graphic is replaced to facilitate the building
placement guide
    Dim intXLower As Integer
    Dim intYLower As Integer

```

'Stores the location of the mouse mouse position in terms of tile coordinates

```
intMouseX = Math.Floor(e.X / 40)  
intMouseY = Math.Floor(e.Y / 40)
```

'Checks whether the current tile is different to the last tile:

```
If intMouseY <> intPrevMouseY Or intMouseX <> intPrevMouseX Then  
    Select Case strCurrentQuadrant
```

'Updates the values of intXLower and intYLower to ensure the solution prints the tiles of the correct quadrant

Case "Top Left"

```
    intXLower = 1  
    intYLower = 1
```

Case "Top Right"

```
    intXLower = 28  
    intYLower = 1
```

Case "Bottom Left"

```
    intXLower = 1  
    intYLower = 16
```

Case "Bottom Right"

```
    intXLower = 28  
    intYLower = 16
```

End Select

'Prints the map tiles in a 2x3 region around the mouse cursor (the largest size the building placement guide can be)

```
    subPrintTile(intPrevMouseX, intPrevMouseY, intXLower, intYLower)  
    subPrintTile(intPrevMouseX, intPrevMouseY - 1, intXLower,  
    intYLower)
```

```
    subPrintTile(intPrevMouseX + 1, intPrevMouseY, intXLower,  
    intYLower)
```

```
    subPrintTile(intPrevMouseX + 1, intPrevMouseY - 1, intXLower,  
    intYLower)
```

```
    gfxGameMap.FillRectangle(Brushes.Transparent, intMouseX * 40,  
    intMouseY * 40, 120, 120)
```

If intPrevMouseY > 0 Then

'Prevents the indices of the array being exceeded

```
    subPrintTile(intPrevMouseX, intPrevMouseY - 2, intXLower,  
    intYLower)
```

```
    subPrintTile(intPrevMouseX + 1, intPrevMouseY - 2, intXLower,  
    intYLower)
```

End If

End If

'Updates the previous location of the mouse cursor

```
intPrevMouseX = intMouseX
```

```

intPrevMouseY = intMouseY

'Prints the building placement guide around the mouse cursor of the
appropriate size according to the building selected
If bInBuildingSelected = True Then
    Select Case strCurrentBuilding
        Case "Cruck Hut"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Farm"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 79, 79)
        Case "Lumberjack Hut"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Water Mill"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 79, 39)
        Case "Cottage"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Brewery"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Quarry"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
(intMouseY - 2) * 40, 79, 119)
        Case "Reeve House"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Fishery"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 39, 79)
        Case "Mine"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 39, 79)
        Case "Pasture"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 79, 79)
    End Select
End If
End Sub

```

The next section of code displays the mouse click event in the FormInGame class. This will attempt to place a building if blnBuildingSelected = True. Validation checks take place in each of the cases of the ptyBuildingType select case to ensure that the target location is suitable for the selected building.

```
Private Sub pbMapDisplay_MouseClick(sender As Object, e As MouseEventArgs) Handles pbMapDisplay.MouseClick
    'This subroutine checks whether a building is selected when the user
    clicks and places the building if possible
    Dim intXLower As Integer
    Dim intYLower As Integer

    Select Case strCurrentQuadrant
        'Updates the values of intXLower and intYLower to ensure the solution
        prints the tiles of the correct quadrant
        Case "Top Left"
            intXLower = 1
            intYLower = 1
        Case "Top Right"
            intXLower = 28
            intYLower = 1
        Case "Bottom Left"
            intXLower = 1
            intYLower = 16
        Case "Bottom Right"
            intXLower = 28
            intYLower = 16
    End Select

    'Checks that a building is selected to prevent this code being run
    unnecessarily:
    If blnBuildingSelected = True And arrMapBlueprint(intMouseY +
    intYLower, intMouseX + intXLower).ptyBuildingType = "Empty" Then
        'Checks which building is selected:
        Select Case strCurrentBuilding
            Case "Cruck Hut"
                'Checks the selected location is valid:
                If arrMapBlueprint(intMouseY + intYLower, intMouseX +
                intXLower).ptyTileType = "Plains" Then
                    'Updates the building property of the tile/region:
```

```

        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Cruck Hut"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 5
        intGoldCount = intGoldCount - 10
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Farm"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower + 1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower + 1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower + 1).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower + 1).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower).ptyBuildingType = "FarmBL"
        arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower + 1).ptyBuildingType = "FarmBR"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower).ptyBuildingType = "FarmTL"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower + 1).ptyBuildingType = "FarmTR"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY - 1, intXLower,
intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 15
        intGoldCount = intGoldCount - 100
        'Deselect the building:
        blnBuildingSelected = False

```

```

    End If
Case "Fishery"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Sea" And arrMapBlueprint(intMouseY + intYLower -
1, intMouseX + intXLower).ptyTileType = "Coast" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "FisheryB"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "FisheryT"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 20
        intGoldCount = intGoldCount - 150
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Water Mill"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower, intMouseX + intXLower + 1).ptyTileType = "River" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Water MillL"
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "Water MillR"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
        'Checks the other possible region of this building:
        ElseIf arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "River" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then

```

```

'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Water MillL"
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "Water MillR"
'Print the building:
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 50
intGoldCount = intGoldCount - 150
'Deselect the building:
bInBuildingSelected = False
End If
Case "Quarry"
'Checks the selected location is valid:
If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 2, intMouseX +
intXLower).ptyTileType = "Mountain" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower - 2, intMouseX +
intXLower).ptyBuildingType = "Empty" Then
'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "QuarryM"
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "QuarryBL"
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "QuarryBR"
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "QuarryTL"
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower + 1).ptyBuildingType = "QuarryTR"
'Print the building:
subPrintTile(intMouseX, intMouseY - 2, intXLower, intYLower)

```

```

        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY - 1, intXLower,
intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 100
        intGoldCount = intGoldCount - 500
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Mine"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Mountain" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "MineB"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "MineT"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 200
        intStoneStockpile = intStoneStockpile - 100
        intGoldCount = intGoldCount - 750
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Pasture"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY +

```

```

intYLower - 1, intMouseX + intXLower + 1).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then
    'Updates the building property of the tile/region:
    arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "PastureBL"
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "PastureBR"
            arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "PastureTL"
                arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower + 1).ptyBuildingType = "PastureTR"
                    'Print the building:
                    subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
                    subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
                    subPrintTile(intMouseX + 1, intMouseY - 1, intXLower,
intYLower)
                    subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
                    'Remove the resources from the user:
                    intWoodStockpile = intWoodStockpile - 25
                    intGoldCount = intGoldCount - 250
                    'Deselect the building:
                    blnBuildingSelected = False
    End If
Case "Lumberjack Hut"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Lumberjack Hut"
            'Print the building:
            subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
            'Remove the resources from the user:
            intWoodStockpile = intWoodStockpile - 10
            intGoldCount = intGoldCount - 50
            'Deselect the building:
            blnBuildingSelected = False
    End If
Case "Cottage"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then

```

```

'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Cottage"
'Print the building:
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 2
intStoneStockpile = intStoneStockpile - 3
intGoldCount = intGoldCount - 200
'Deselect the building:
bInBuildingSelected = False
End If
Case "Reeve House"
'Checks the selected location is valid:
If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Reeve House"
'Print the building:
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 200
intStoneStockpile = intStoneStockpile - 20
intGoldCount = intGoldCount - 750
'Deselect the building:
bInBuildingSelected = False
End If
Case "Brewery"
'Checks the selected location is valid:
If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Brewery"
'Print the building:
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 20
intGoldCount = intGoldCount - 250
'Deselect the building:
bInBuildingSelected = False
End If

```

```

    End Select
End If
End Sub
End Class

```

The following lines of code appear in the mini-map click events. The code assigns a string to the strCurrentQuadrant variable which has been used to calculate the array location of a map tile. Each line appears in the respective mini-map click event.

```

strCurrentQuadrant = "Top Left"
strCurrentQuadrant = "Top Right"
strCurrentQuadrant = "Bottom Left"
strCurrentQuadrant = "Bottom Right"

```

The final section of code prints individual map tiles to the screen. This code appears in the PrintMapModule. Two select cases check the tile type and the building type of the target array location and print the appropriate graphics to the tile.

```

'Subroutine that prints a single maptile to the screen
Sub subPrintTile(ByVal intXCoord As Integer, ByVal intYCoord As Integer,
ByVal intXLower As Integer, ByVal intYLower As Integer)
    Dim intX, intY As Integer
    intY = intYCoord * 40
    intX = intXCoord * 40
    rectGraphicsDestination = New Rectangle(intX, intY, 40, 40)
    rectGraphicsSource = New Rectangle(0, 0, 40, 40)
    Select Case arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyTileType
        Case "Plains" 'Generate a plains graphic at the location
            bmpImage = New Bitmap(FormImgRef.pbPlainsTile.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "Flora" 'Generate a flora graphic at the location
            bmpImage = New Bitmap(FormImgRef.pbFloraTile.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    End Select
End Sub

```

```

Case "Hill" 'Generate a hill graphic at the location
bmplImage = New Bitmap(FormImgRef.pbHillTLTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Forest" 'Generate a forest graphic at the location
bmplImage = New Bitmap(FormImgRef.pbForestCentreTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "RiverSource" 'Generate a riverSource graphic at the location
bmplImage = New Bitmap(FormImgRef.pbRiverSourceTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Stream" 'Generate a stream graphic at the location
bmplImage = New Bitmap(FormImgRef.pbStreamTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "River" 'Generate a river graphic at the location
bmplImage = New Bitmap(FormImgRef.pbRiverTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Coast" 'Generate a coast graphic at the location
bmplImage = New Bitmap(FormImgRef.pbCoastTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Sea" 'Generate a sea graphic at the location
bmplImage = New Bitmap(FormImgRef.pbSeaTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Mountain" 'Generate a mountain graphic at the location
bmplImage = New Bitmap(FormImgRef.pbMountainTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

End Select

Select Case arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyBuildingType
Case "Cruck Hut" 'Print a cruck hut
bmplImage = New Bitmap(FormImgRef.pbBuildingCruckHut.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Lumberjack Hut" 'Print a lumberjack hut

```

```

bmplImage = New
Bitmap(FormImgRef.pbBuildingLumberjackHut.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Cottage" 'Print a cottage
bmplImage = New Bitmap(FormImgRef.pbBuildingCottage.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Reeve House" 'Print a reeve house
bmplImage = New
Bitmap(FormImgRef.pbBuildingReeveHouse.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Brewery" 'Print a brewery
bmplImage = New Bitmap(FormImgRef.pbBuildingBrewery.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmTL" 'Print the top left of a farm
bmplImage = New Bitmap(FormImgRef.pbBuildingFarmTL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmTR" 'Print the top right of a farm
bmplImage = New Bitmap(FormImgRef.pbBuildingFarmTR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmBL" 'Print the bottom left of a farm
bmplImage = New Bitmap(FormImgRef.pbBuildingFarmBL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmBR" 'Print the bottom right of a farm
bmplImage = New Bitmap(FormImgRef.pbBuildingFarmBR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FisheryT" 'Print the top of a fishery
bmplImage = New Bitmap(FormImgRef.pbBuildingFisheryT.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FisheryB" 'Print the bottom of a fishery

```

```
bmpImage = New Bitmap(FormImgRef.pbBuildingFisheryB.Image)
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "Water MillL" 'Print the left of a water mill
    If arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyTileType = "Plains" Then
        bmpImage = New
Bitmap(FormImgRef.pbBuildingWaterMillLP.Image)
    Else
        bmpImage = New
Bitmap(FormImgRef.pbBuildingWaterMillLR.Image)
    End If
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "Water MillR" 'Print the right of a water mill
    If arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyTileType = "Plains" Then
        bmpImage = New
Bitmap(FormImgRef.pbBuildingWaterMillRP.Image)
    Else
        bmpImage = New
Bitmap(FormImgRef.pbBuildingWaterMillRR.Image)
    End If
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "QuarryM" 'Print the mountain section of a quarry
    bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryM.Image)
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "QuarryTL" 'Print the top left of a quarry
    bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryTL.Image)
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "QuarryTR" 'Print the top right of a quarry
    bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryTR.Image)
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "QuarryBL" 'Print the bottom left of a quarry
    bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryBL.Image)
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
```

```

Case "QuarryBR" 'Print the bottom right of a quarry
bmplImage = New Bitmap(FormImgRef.pbBuildingQuarryBR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "MineT" 'Print the top of a mine
bmplImage = New Bitmap(FormImgRef.pbBuildingMineT.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "MineB" 'Print the bottom of a mine
bmplImage = New Bitmap(FormImgRef.pbBuildingMineB.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "PastureTL" 'Print the top left of a pasture
bmplImage = New Bitmap(FormImgRef.pbBuildingPastureTL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "PastureTR" 'Print the top right of a pasture
bmplImage = New
Bitmap(FormImgRef.pbBuildingPastureTR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "PastureBL" 'Print the bottom left of a pasture
bmplImage = New Bitmap(FormImgRef.pbBuildingPastureBL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "PastureBR" 'Print the bottom right of a pasture
bmplImage = New
Bitmap(FormImgRef.pbBuildingPastureBR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

End Select
End Sub

```

Finally code was added in the subPrintMap procedure to print buildings when a quadrant is printed to the screen. The code is almost identical to the code of the previous section:

```

Select Case arrMapBlueprint(intY, intX).ptyBuildingType
Case "Cruck Hut" 'Print a cruck hut
bmplImage = New
Bitmap(FormImgRef.pbBuildingCruckHut.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

```

```

    Case "Lumberjack Hut" 'Print a lumberjack hut
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingLumberjackHut.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "Cottage" 'Print a cottage
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingCottage.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "Reeve House" 'Print a reeve house
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingReeveHouse.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "Brewery" 'Print a brewery
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingBrewery.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "FarmTL" 'Print the top left of a farm
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingFarmTL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "FarmTR" 'Print the top right of a farm
            bmpImage = New
    Bitmap(FormImgRef.pbBuildingFarmTR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "FarmBL" 'Print the bottom left of a farm
            bmpImage = New
    Bitmap(FormImgRef.pbBuildingFarmBL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "FarmBR" 'Print the bottom right of a farm
            bmpImage = New
    Bitmap(FormImgRef.pbBuildingFarmBR.Image)

```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "FisheryT" 'Print the top of a fishery
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingFisheryT.Image)
```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "FisheryB" 'Print the bottom of a fishery
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingFisheryB.Image)
```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "Water MillL" 'Print the left of a water mill
```

```
If arrMapBlueprint(intY, intX).ptyTileType = "Plains" Then
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingWaterMillLP.Image)
```

```
Else
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingWaterMillLR.Image)
```

```
End If
```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "Water MillR" 'Print the right of a water mill
```

```
If arrMapBlueprint(intY, intX).ptyTileType = "Plains" Then
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingWaterMillRP.Image)
```

```
Else
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingWaterMillRR.Image)
```

```
End If
```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "QuarryM" 'Print the mountain section of a quarry
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryM.Image)
```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

```
Case "QuarryTL" 'Print the top left of a quarry
```

```
bmpImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryTL.Image)
```

```

gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "QuarryTR" 'Print the top right of a quarry
bmplImage = New
Bitmap(FormImgRef.pbBuildingQuarryTR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "QuarryBL" 'Print the bottom left of a quarry
bmplImage = New
Bitmap(FormImgRef.pbBuildingQuarryBL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "QuarryBR" 'Print the bottom right of a quarry
bmplImage = New
Bitmap(FormImgRef.pbBuildingQuarryBR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "MineT" 'Print the top of a mine
bmplImage = New
Bitmap(FormImgRef.pbBuildingMineT.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "MineB" 'Print the bottom of a mine
bmplImage = New
Bitmap(FormImgRef.pbBuildingMineB.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "PastureTL" 'Print the top left of a pasture
bmplImage = New
Bitmap(FormImgRef.pbBuildingPastureTL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "PastureTR" 'Print the top right of a pasture
bmplImage = New
Bitmap(FormImgRef.pbBuildingPastureTR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "PastureBL" 'Print the bottom left of a pasture
bmplImage = New
Bitmap(FormImgRef.pbBuildingPastureBL.Image)

```

```
gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "PastureBR" 'Print the bottom right of a pasture
    bmpImage = New
    Bitmap(FormImgRef.pbBuildingPastureBR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
End Select
Next
Next
End Sub
```

This concludes the coding section of this iteration.

Developing the solution (Testing to inform development):

All tests for this iteration were recorded using a screen recording software called Grabilla.

Evidence of testing:

The table below describes the testing process for this iteration.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
DD4.1	Test that the resource stockpiles and gold count in the treasury menu are updated every game tick.	A cruck hut is placed	5 wood and 10 gold is removed from the user.	Yes.
DD4.2	Test that the user is prevented from clicking a building button if the user lacks sufficient resources.	No test data. The user attempts to click a building button.	The user is prevented from building the building and a message box describes what the user is missing.	Yes.
DD4.3	Test that clicking a building button with the correct amount of resources creates a rectangle anchored to the cursor that shows the building size.	Each building will be tested.	A rectangle with the dimensions of the building to be placed is drawn around the mouse cursor and is removed as the mouse cursor moves.	Yes.
DD4.4	Test that clicking the map while a building is selected in the incorrect location does not place a building.	Attempt to place all buildings in an invalid location (in the sea).	No building is placed.	Yes.
DD4.5	Test that clicking the map while a building is selected in the correct location places the building and removes the resources required to build the building from the user.	Attempt to place all buildings in their respective valid locations.	A building is drawn to the map and the correct amount of resources are removed from the user.	Yes.

DD4.1

This test can be viewed in the file called DD4.1 DD4.5.mp4. 6 seconds into the video the resources can be seen to be updated after a cruck hut is placed. Other examples of this

can be seen throughout the video as more buildings are placed. Other resources This test is therefore a success.

DD4.2

This test can be viewed in a file called DD4.2.mp4.

The video shows that when the user attempts to click a building button without the prerequisite resources, a message box appears describing what the user is missing. The user is prevented from placing the building. This test is therefore a success.

DD4.3

This test can be viewed in a file called DD4.3.mp4.

The video demonstrates that each building has a foundation guideline centred on the mouse of equal dimensions to the building that is to be placed. This video demonstrates that this test is a success.

DD4.4

This test can be viewed in a file called DD4.4.mp4.

The video demonstrates that when an attempt is made to place a building in an invalid location, it fails. This test is therefore a success.

DD4.5

This test can be viewed in a file called DD4.1 DD4.5.mp4.

The video demonstrates that after a building is placed the correct amount of resources are removed from the user, as seen in the treasury menu. Therefore this test is a success.

Stakeholder review and sign-off of iteration

I interviewed my stakeholders after they tried the final product of the iteration and asked them whether they thought that the iteration's content met all the requirements of the iteration. My stakeholders unanimously agreed that the content that this iteration set out to implement has been successfully implemented into the solution.

END OF FOURTH ITERATION

END OF ITERATIVE DEVELOPMENT

EVALUATION

Testing to inform evaluation

In each iteration of the development process, I planned usability feature tests to test the solution in relation to the success criteria of the solution that each iteration addressed and post development tests that test the solution for robustness and test the solution against the success criteria requirements that each iteration addressed. These tests will help inform the evaluation process, and identify errors in the program and areas that can be improved on in later versions of the solution.

Post development testing for functionality and robustness:

Throughout the iterative development process I designed post development tests to test the functionality and robustness of the solution. These will be performed now to test these aspects of the solution and to discern whether the success criteria requirements were addressed by the implementation of the solution. The table below describes this process.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
PD1.1	Testing that clicking the select Seljuk Turks button on the select civilisation menu correctly updates the intCivilisation variable.	Mouse input, intCivilisation	intCivilisation is assigned the value 2	Yes.
PD1.2	Testing that clicking the select Kingdom of England button on the select civilisation menu correctly updates the intCivilisation variable.	Mouse input, intCivilisation	intCivilisation is assigned the value 1	Yes.
PD1.3	Testing that clicking the select Yuan Dynasty button on the select civilisation menu correctly updates the intCivilisation variable.	Mouse input, intCivilisation	intCivilisation is assigned the value 3	Yes.
PD1.4	Testing that upon entering the town name into the town name menu, the value of strTownName is updated.	Text input of tbTownName (Westminster), strTownName	strTownName is assigned the string “Westminster”	Yes.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
PD1.5	Testing that the text of the in-game label displaying the town name takes on the value of strTownName.	The text of the label displaying the town name is equal to strTownName (test name is Westminster)	In-game town name label displays “Westminster”	Yes.
PD1.6	Testing that the button to open to the help menu from the main menu functions correctly.	Mouse input clicking the help button on the main menu.	The help menu is opened without closing any other form.	Yes.
PD1.7	Testing that the button to open to the help menu from the in-game screen functions correctly.	Mouse input clicking the help button on the in-game screen.	The help menu is opened without closing any other form.	Yes.
PD1.8	Testing that the label displaying the population level is displayed correctly.	None.	lblPopulationLevel displays the initial population (0).	Yes.
PD1.9	Testing that the label displaying wealth and income is displayed correctly.	None.	lblWealth displays in the initial value of gold (1000); lblIncome displays the initial value of income (0).	Yes.
PD1.10	Testing that the label displaying the in-game date is displayed correctly.	None.	lblDate displays the date correctly (with correct number of days in each month) starting on 1st January 1296; and pbMonth's image is updated to each month.	Yes.
PD1.11	Testing that the building menu button opens the construction menu correctly.	Mouse input clicking the building menu button.	The building menu opens upon this button being clicked.	Yes.
PD1.12	Testing that the management menu button opens the management menu correctly.	Mouse input clicking the management menu button.	The management menu opens upon this button being clicked.	Yes.
PD1.13	Testing that the treasury button opens the treasury menu correctly.	Mouse input clicking the treasury menu button.	The treasury menu opens upon this button being clicked.	Yes.
PD1.14	Testing that the start button opens the start menu correctly.	Mouse input clicking the start menu button.	The start menu opens upon this button being clicked.	Yes.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
PD1.15	Testing that the save game button is displayed correctly in the start menu.	None.	The save game button appears in the start menu.	Yes.
PD1.16	Testing that the return to main menu functions correctly.	Mouse input clicking the main menu button.	The main menu is opened and the in-game screen is closed.	Yes.
PD1.17	Testing that the navigation buttons on each menu work correctly.	Mouse input clicking navigation buttons on each menu.	Upon clicking the navigation buttons the respective menu is opened.	Yes.
PD2.1.1	Does the solution generate a river source tile?	None.	River source tile generates in the region ($8 \leq X \leq 47$, $2 \leq Y \leq 3$).	Yes.
PD2.1.2	Does the solution generate a stream leading from the river source tile?	None.	Stream of length 2 to 4 tiles generated beneath river source tile.	Yes.
PD2.1.3	Is a river generated beneath the last stream tile?	<code>intRiverBendiness = 3</code> (minimum value, maximum bendiness).	River with a width of 2 tiles generates, meandering every 4th tile (after every 3 straight tiles) to the bottom of the map.	Yes.
PD2.2.1	If a lake is generated, is it shaped randomly with varying curvature?	<code>blnSeaLake = False</code> <code>intLakeHeight = 15</code> <code>intLakeWidth = 27</code> (maximum values).	Lake is generated around bottom of river with varying shape either side of river.	Yes.
PD2.2.2	If a sea is generated, does the shape of the sea appear curved?	<code>blnSeaLake = True</code> .	Shape of sea appears modelled upon sine or cosine function, with changing amplitude.	Yes.
PD2.2.3	Are the tiles surrounding the edge of the water feature replaced with coast tiles?	None.	All tiles (except river tiles) adjacent to the edge sea tiles are replaced with coast tiles.	Yes.
PD2.3	Is the top of the map populated by mountain ranges of random shape?	<code>intMountainFrequency = 4</code> (maximum value).	Mountain ranges are generated within the co-ordinate bounds of ($1 \leq X \leq 54$, $1 \leq Y \leq 4$)	Yes.
PD2.4	Are the remaining edges of the map filled with randomly shaped and sized forests?	<code>intForestProbability = 3</code> (maximum value).	Forests of varying shape and size are generated around the remaining edges of the map.	Yes.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
PD2.5	Does the map feature hills of size 2x2 tiles?	intHillFrequency = 28 (maximum value).	A large number of hills sized 2x2 populate the map with a random distribution	Yes.
PD2.6	Does the map include randomly placed flora tiles?	intFloraFrequency = 11 (minimum value).	Plains tiles are replaced randomly with flora tiles.	Yes.
PD2.7	Are all remaining empty tiles filled with plains tiles?	None.	No empty tiles remain after plains tiles are generated.	Yes.
PD3.1	Is the date formatted correctly, and does the in-game date advance at a reasonable pace?	None.	Date is correctly formatted and the date advances at 1 day per second. Stakeholders were satisfied with pace of in-game time.	Yes. Stakeholders were satisfied with pace of in-game time.
PD3.2	Does the map settings menu have a route to all other menus/forms?	-	-	Redundant.
PD4.1	Test that each civilisation has the correct unique building available.	Testing English unique building: intCivilisation = 1 Testing Seljuk Turk unique building: intCivilisation = 2 Testing Yuan Dynasty unique building: intCivilisation = 3	The Yuan Dynasty civilisation has access to terraced farms in the serfs section; the English have access to fletchers in the freemen section and the Seljuk Turks have access to spice markets in the yeomen section.	Yes.
PD4.2	Test that at least 10 buildings are available to the user that can be placed.	Mouse input clicking buttons to place 10 different buildings.	The user can place at least 10 individual buildings.	Yes.
PD4.3	Test that certain buildings are prevented from being placed in certain locations.	Mouse input attempting to place buildings in invalid locations (in the sea, in forests and in mountains).	Attempting to place certain buildings in certain locations fails.	Yes.
PD4.4	Test that certain buildings are allowed to be placed in certain locations.	Mouse input attempting to place buildings in their respective valid locations.	Attempting to place certain buildings in certain locations succeeds.	Yes.

I considered adding another test called PD3.3 which would test the map generation with all the most extreme values, to ensure that the map generation could cope with any input. However this test would have been made redundant by tests PD2.3, PD2.4, PD2.5, PD2.6 and PD2.7. This is shown in the evidence of testing section.

Evidence of testing:

PD1.1, PD1.2 and PD1.3

For this test I added a line of temporary code to each select civilisation button that would output the value of intCivilisation to the MessageBox:

`MessageBox.Show(FormInGame.intCivilisation)`

This test can be viewed in a file called PD1.1 PD1.2 PD1.3.mp4.

The video demonstrates the output of intCivilisation to the MessageBox upon clicking the civilisation buttons. The values of intCivilisation were equal to the expected values, thus the tests are successful. The code that was added to perform this test was removed after the tests passed.

PD1.4 and PD1.5

For this test I added a line of temporary code to the settle town click event which displays the value of strTownName:

`MessageBox.Show(FormInGame.strTownName)`

This test can be viewed in a file called PD1.4 PD1.5.mp4.

The video demonstrates the output of strTownName to the MessageBox upon clicking the settle town button. The value of strTownName was equal to the expected value, thus PD1.4 is a success; the string displayed in lblTownName was equal to strTownName, hence PD1.5 is also a success. The code that was added to perform this test was removed after the tests passed.

PD1.6 and PD1.7

These tests were captured using Grabilla and can be viewed in a file called PD1.6 PD1.7.mp4. The help menu was successfully opened from both the main menu and the in-game screen correctly, thus these tests are successful.

PD1.8, PD1.9 and PD1.10

These tests were captured using Grabilla and can be viewed in a file called PD1.8 PD1.9 PD1.10.mp4. lblPopulationLevel, lblWealth and lblIncome all displayed the expected values. Thus PD1.8 and PD1.9 are successful. For PD1.10 I reduced the interval property of tmrGameTimer from 1000 to 50 to decrease the length of the test. The date was displayed correctly and the date was formatted correctly with the right number of days in each month; the video demonstrates all the months of a leap year and January and February of a non-leap year to demonstrate this. pbMonth displayed the correct image at all times. Thus this test is a success. The interval property of tmrGameTimer was returned to 1000 after the tests.

PD1.11, PD1.12, PD1.13, PD1.14, PD1.15, PD1.16

These tests were captured using Grabilla and can be viewed in a file called PD1.11 PD1.12 PD1.13 PD1.14 PD1.15 PD1.16.mp4. As seen in video, the building menu; the management menu; the treasury menu and the start menu were all opened correctly, thus PD1.11, PD1.12, PD1.13 and PD1.14 are all successful. The save game button appeared correctly in the start menu hence PD1.15 is also a success. Finally clicking the main menu button in the start menu closed the in-game screen and opened the main menu as expected, thus PD1.16 is also successful.

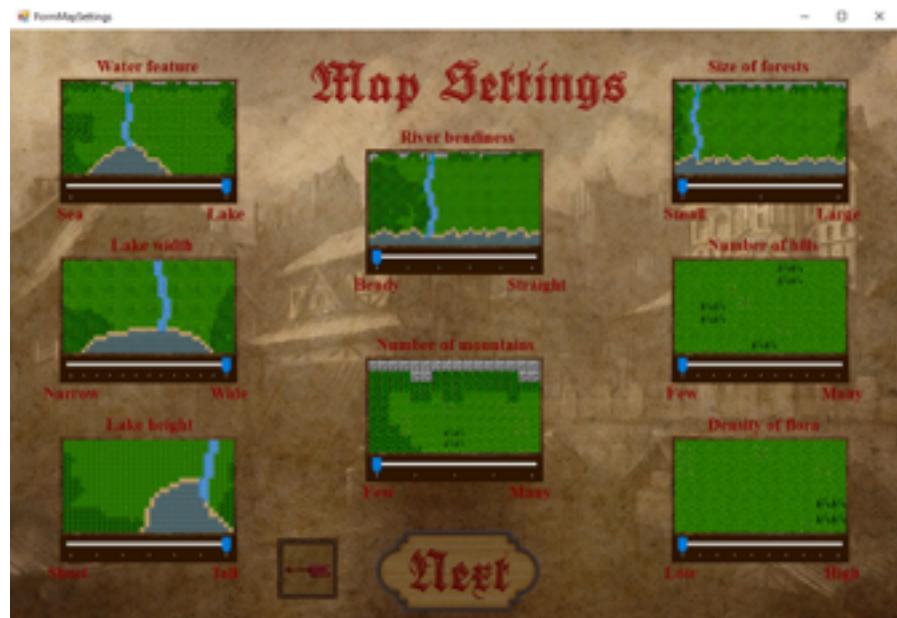
PD1.17

This test was captured using Grabilla and can be viewed in a file called PD1.17.mp4. The video demonstrates the use of the navigation buttons to move between menus. All buttons worked as expected and thus this test is a success.

PD2.1.1, PD2.1.2, PD2.1.3 and PD2.2.1

To perform these tests I inputed the values for intRiverBendiness, blnSeaLake, intLakeHeight and intLakeWidth using the map settings menu. The test data for these values were chosen because these are the maximum values and hence the most extreme values. Evidence of these values being input can be seen on the following page.

Image of test data being input on the map settings menu.



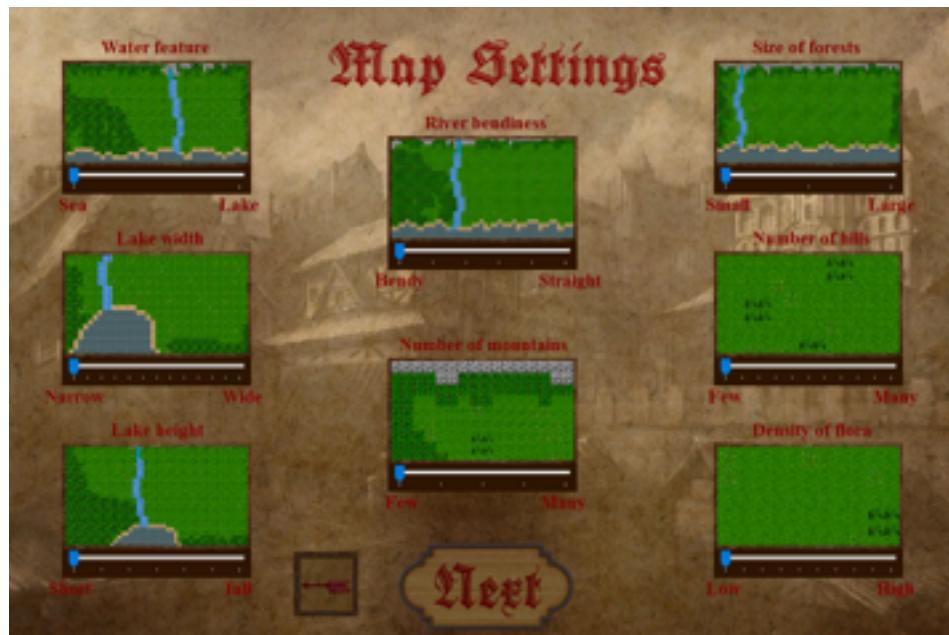
After these had been input, starting the game would allow me to simultaneously test these four tests. The generated map is shown below:



It is clear that the river source tile, the stream tiles and the river tiles generated in the expected locations and format. Thus PD2.1.1, PD2.1.2 and PD2.1.3 are all successful. I thought that the lake was generated in a way that fits the description of the expected result, but I decided to consult my stakeholders. They all agreed with me and so PD2.2.1 is also successful. The result of this test will also be used to partially test PD2.2.3 as the coast tiles generated around the lake as expected. In order to pass PD2.2.3 however, the same result must be seen with a map that includes a sea.

PD2.2.2 and PD2.2.3

To perform PD2.2.2, and to complete PD2.2.3 I inputed the value of blnSeaLake to match the test data and started the game. The image demonstrating this value being input is shown below.



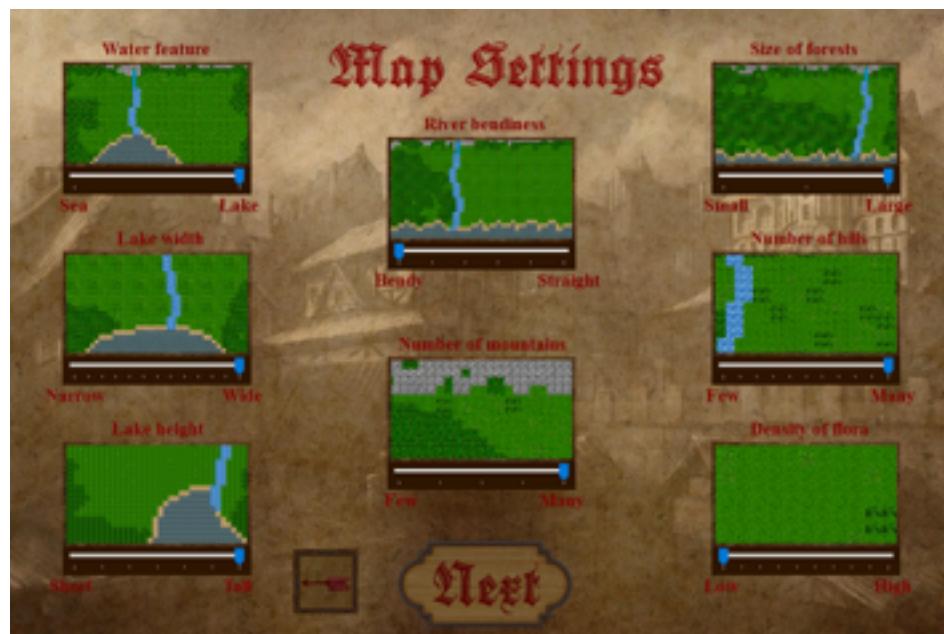
After this had been input, I started the game, generating the map. The generated map is shown below:



Once again I consulted my stakeholders to gain their opinion as to whether the sea that had been generated met the description of the expected result. They unanimously agreed that the it did. Thus PD2.2.2 is a success. As the coast generated correctly around the lake (seen in PD2.1.1, PD2.1.2, PD2.1.3 and PD2.2.1) and the coast also generated correctly around the sea in this test, PD2.2.3 also is a success.

PD2.3, PD2.4, PD2.5, PD2.6 and PD2.7

To perform these tests I inputed the test data using the map settings menu. Evidence of this is shown below. I decided to test these all together as all the specified data are the extreme values of each variable. Thus if the map successfully generates with all these extreme values, then the map generation will be able to cope with any input. The inputs of the lake/sea related variables is arbitrary.



The map that was generated from these settings is shown below:



The mountains generated within the correct region and thus PD2.3 is a success. My stakeholders agreed that the shape and size of forests met their expectations and so PD2.4 is also a success. My stakeholders also agreed that the number of hills generated met their expectations and all the hills are of the correct size hence PD2.5 passes. The flora frequency

also was of the expected magnitude and so PD2.6 also passes. The remaining tiles were converted to plains tiles (as seen), thus PD2.7 passes.

PD3.1

This test has been made partially redundant by PD1.10 which demonstrated that the date is formatted correctly. In order for this test to pass, my stakeholders need to be satisfied with the game pace. My stakeholders were all satisfied with the game pace upon consultation and so this test passes.

PD3.2

This test has been made entirely redundant by PD1.17 which demonstrated the map settings menu navigation working correctly.

PD4.1

This test was captured using Grabilla and can be viewed in the file PD4.1.mp4. As seen in the video the English have access to the fletchers, the Seljuk Turks have access to spice markets and the Yuan Dynasty have access to terraced farms as expected. Hence this test is a success.

PD4.2 and PD4.4

These tests were captured using Grabilla and can be viewed in the file called PD4.2 PD4.4.mp4. These tests both demonstrate very similar things and can be performed simultaneously. By demonstrating that 10 buildings can be placed (PD4.2), it would also be shown that the buildings are allowed to be placed in specific locations (PD4.4). All buildings were allowed to be placed in the correct place and 10 buildings could be placed. Thus both PD4.2 and PD4.4 pass.

PD4.3

This test was captured using Grabilla and can be viewed in the file called PD4.3.mp4. The video demonstrates attempting to place each building in the sea, in forests and in mountains (all invalid locations). As no building is placed, as expected, this test is a success.

All post-development tests for functionality and robustness were successful. This concludes the post-development section.

Usability features testing:

Throughout iterative development I designed usability tests to test usability features of the solution. These will be performed now to discern whether the success criteria were addressed by the implementation of the solution. The table below describes the testing process.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
UF1.1	Does each civilisation have unique theme for the user interfaces?	None.	Town name menu loads interface specific to each civilisation. In-game screen has unique user interface for each civilisation.	Yes.
UF1.2	Is the town name fully customisable within reason (of reasonable length)	Using 'WWWWWWWWWWWWWWWW' (14 Ws) as a name to see whether the town name label is exceeded. (W is the largest character that can be entered). 'Constantinople' (14 letters of varying size).	Label is not exceeded. Only 14 characters may be entered.	Partial success. Extreme case (14 Ws failed, however the normal 14 character name succeeded). This feature must be reworked in a future version.
UF1.3	Is there a help menu available to the user from important locations? (i.e. the main menu and from in-game)	Mouse input clicking the help menu buttons from in-game and main menu.	Help menu opens successfully after clicking the relevant buttons on the main menu and in-game screen.	Yes.
UF1.4	Does the initial in game screen have a user interface that provides the user with the most fundamental information about the town?	intGoldCount = 1000 intPopulationLevel = 0 intGoldIncome = 0 strTownName = Westminster strDate = 1st January 1296	lblPopulationLevel = 0 lblGoldCount = 1000 lblGoldIncome = 0 lblTownName = Westminster lblDate = 1st January 1296 pbMonth = January icon	Yes.
UF2.1	Does the map generate randomly?	intSeaLake = False intLakeHeight = 15 intLakeWidth = 27 intRiverBendiness = 3 intMountainFrequency = 4 intForestProbability = 3 intHillFrequency = 28 intFloraFrequency = 6	Map looks different each time.	Yes.
UF3.1	Is the in-game date displayed correctly and clearly on the in-game form?	None.	lblDate contains date starting from 1st January 1296	Yes.

Test number	Test name, reason for test	Test data	Expected Result	Test successful? Comments
UF3.2	Is the method of navigation appropriate, and does it work correctly?	Mouse input clicking navigation buttons on mini-map.	Clicking the individual buttons on the mini-map draws the relevant part of the map to the screen.	Yes. See stakeholder feedback.
UF3.3	Is the method of selecting map settings appropriate and are the options reflected in the generated map?	First test: Water features set to sea; River bendiness set to minimum; Number of mountains set to few; Size of forests set to small; Number of hills set to few; Density of flora set to low. Second test: Water features set to lake; Lake width set to Narrow; Lake height set to tall; River bendiness set to maximum; Number of mountains set to many; Size of forests set to large; Number of hills set to many; Density of flora set to high.	Stakeholders are satisfied that the map settings are reflected correctly by the map generation.	Yes. See stakeholder feedback.
UF4.1	Testing that each civilisation has access to a unique building.	intCivilisation = 1, 2 or 3 depending upon the civilisation.	Each civilisation has access to a unique building	Yes.
UF4.2	Test that the user has access to, and can build at least 10 different types of building.	Mouse input placing 10 buildings.	10 buildings can be placed.	Yes.
UF4.3	Testing that each level of civilisation has a set of buildings that can be placed.	None.	Each level of civilisation has at least 1 building available.	Yes.

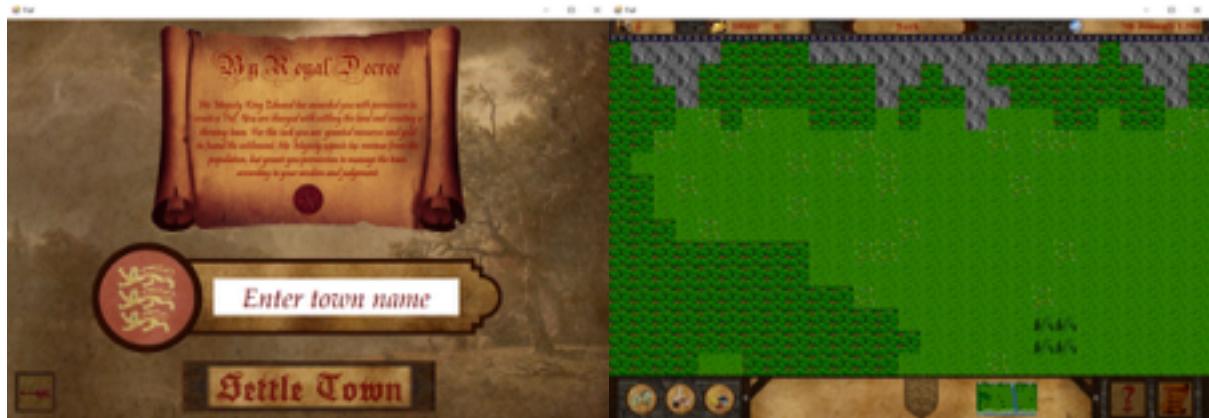
Evidence of testing:

UF1.1

For this test I selected each civilisation using the select civilisation menu and took pictures of the user interfaces for the town name menu and the in-game screen for each civilisation. These are shown on the following page. As each civilisation had unique user interfaces for both the town name menu and the in-game screen as expected, this test passes. I consulted my stakeholders to get their opinion on the matter and whether they thought the user interfaces were of a reasonable standard. They all agreed that the user interfaces met their expectations.

UI for Kingdom of England civilisation:

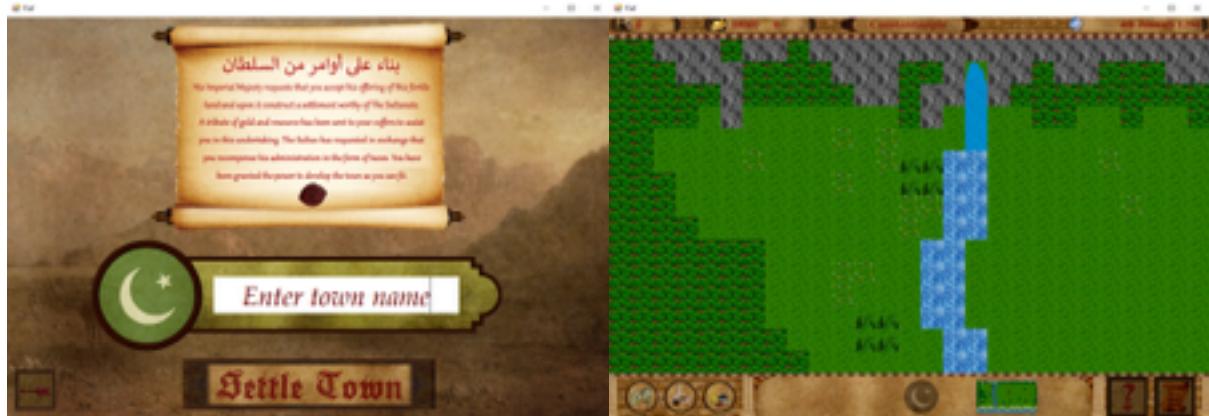
UI for town name menu:



UI for in-game screen:

UI for Seljuk Turk civilisation:

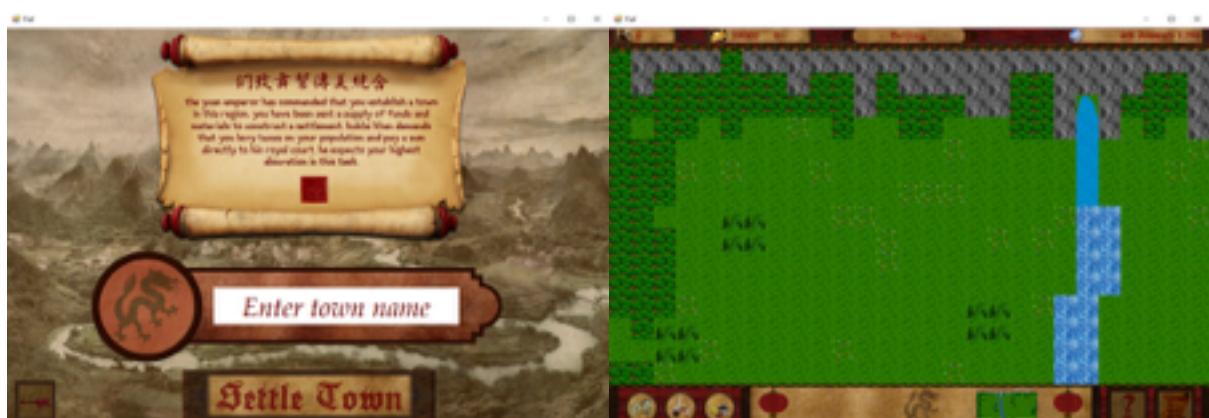
UI for town name menu:



UI for in-game screen:

UI for Yuan Dynasty civilisation:

UI for town name menu:



UI for in-game screen:

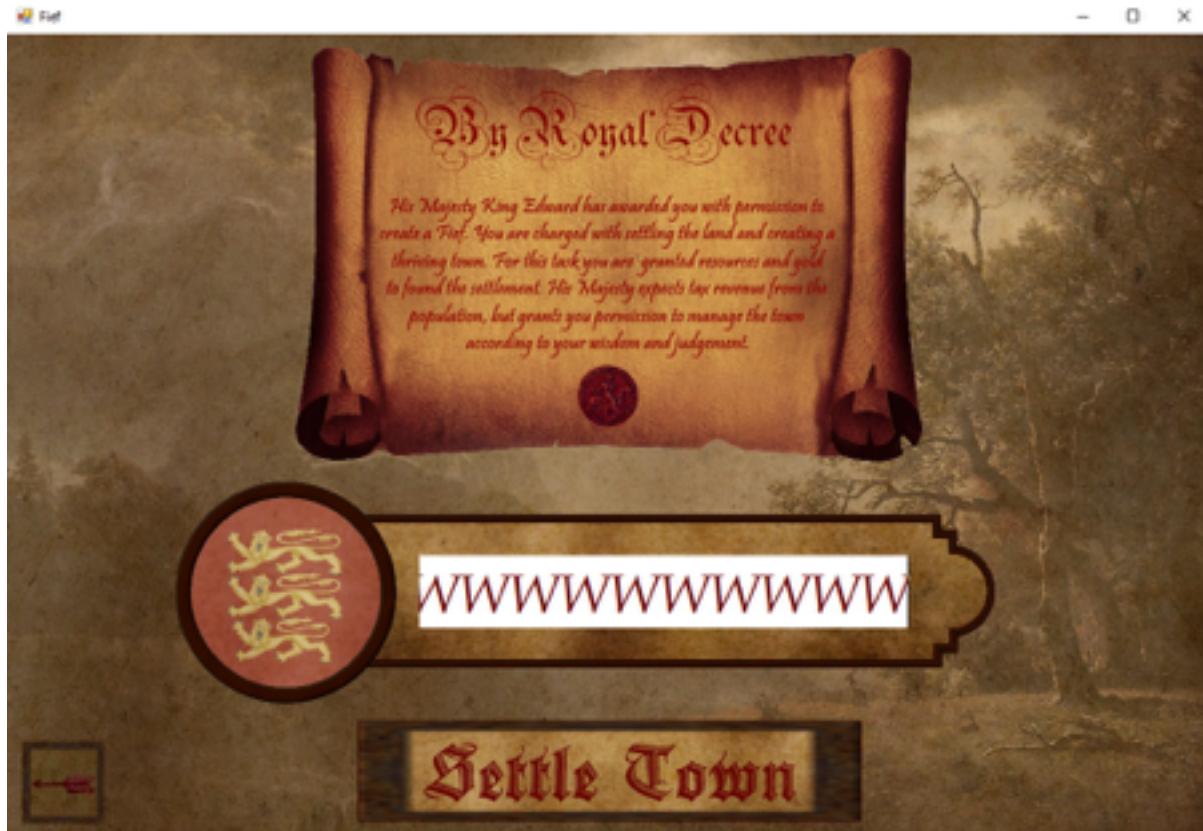
UF1.2

I performed this test by typing in the test town names into the text box on the town

name menu and tested that the input would fit in the text box and also checked that the inputted text fitted into lblTownName on the in-game screen. The results of this are shown below:

Testing: 'WWWWWWWWWWWWWWWW' (longest possible input)

Input of test string into town name menu text box was successful:



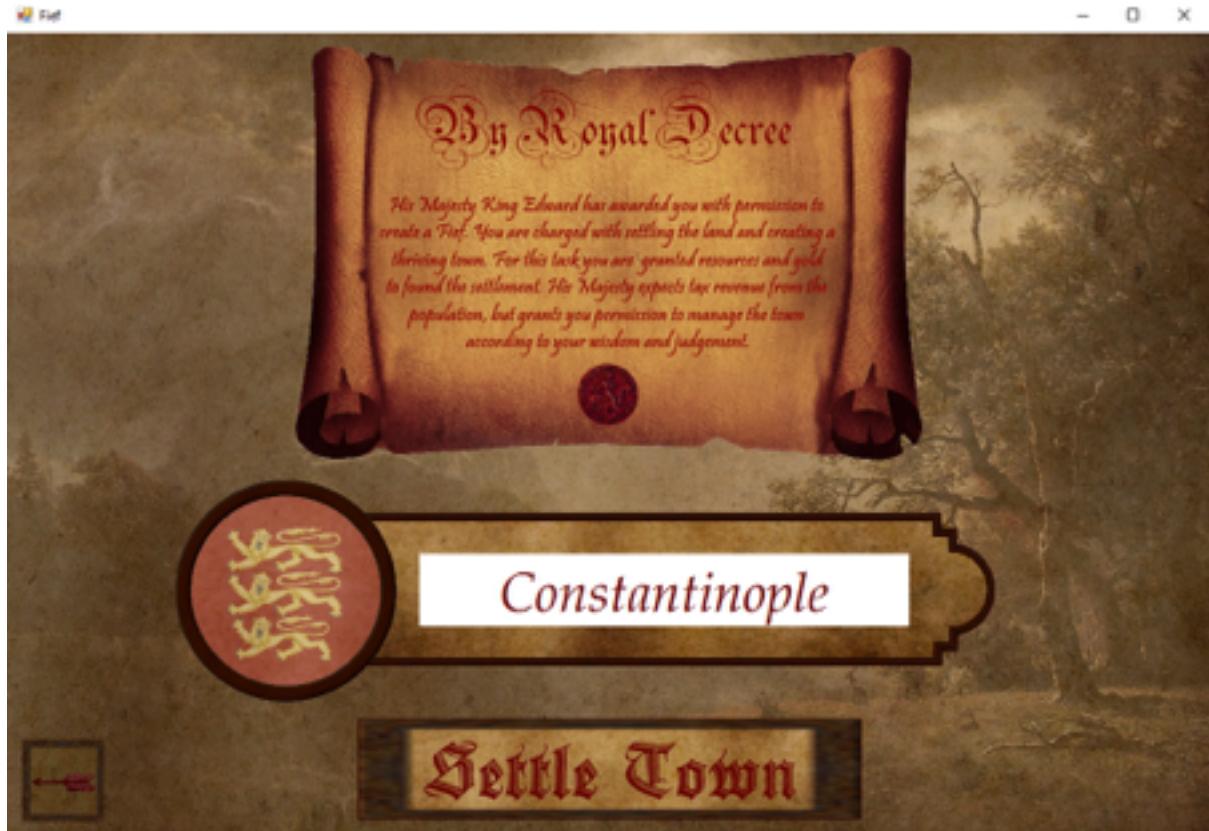
However lblTownName only displayed 9 of the Ws. This is because of the size of the text box:



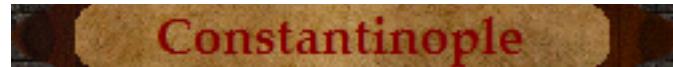
This demonstrates that the town name isn't necessarily fully customisable, however this town name is ridiculous. I will consult my stakeholders if this is acceptable after testing the string 'Constantinople'.

Testing: 'Constantinople' (string of maximum characters, but using smaller characters).

Input of test string into town name text box was successful (on following page):



This string was correctly displayed in lblTownName:



This demonstrates that town names of up to 14 characters can be used normally. I consulted my stakeholders after testing this text and they agreed that they were satisfied that the fact that the 14 Ws were not displayed properly is not a concern of theirs as it is an extreme case. They suggested that in a later version I ensure that such an extreme case is catered for by changing the text size depending upon the size of the string. Based upon the response of my stakeholders, this test is a partial success, albeit not a total success as this feature must be reworked in a future version.

UF1.3

This test was recorded using Grabilla and can be viewed in a file called UF1.3.mp4. The help menu was successfully opened from both the in-game screen and the main menu hence this test is a success.

UF1.4

I tested this by entering 'Westminster' into the town name menu. The image below shows the initial screen.



All the most fundamental information about the town is displayed correctly hence this test is a success.

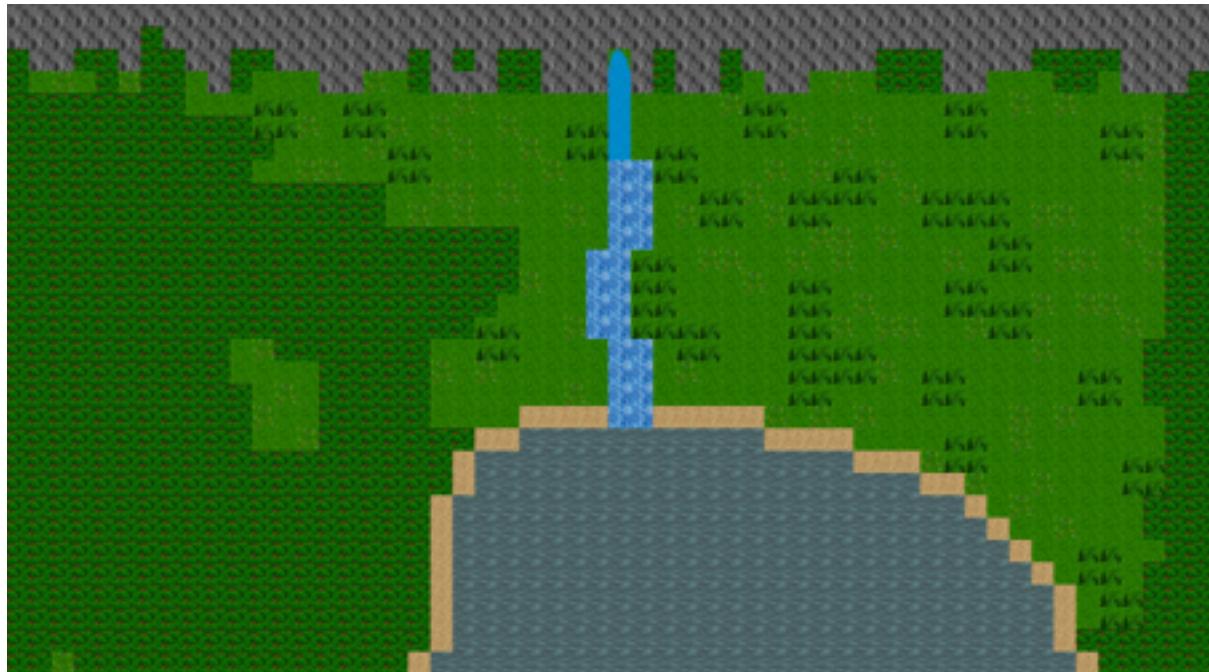
UF2.1

I generated two maps using the same test data by inputting it into the map settings menu. Evidence of the test data being input is shown below:

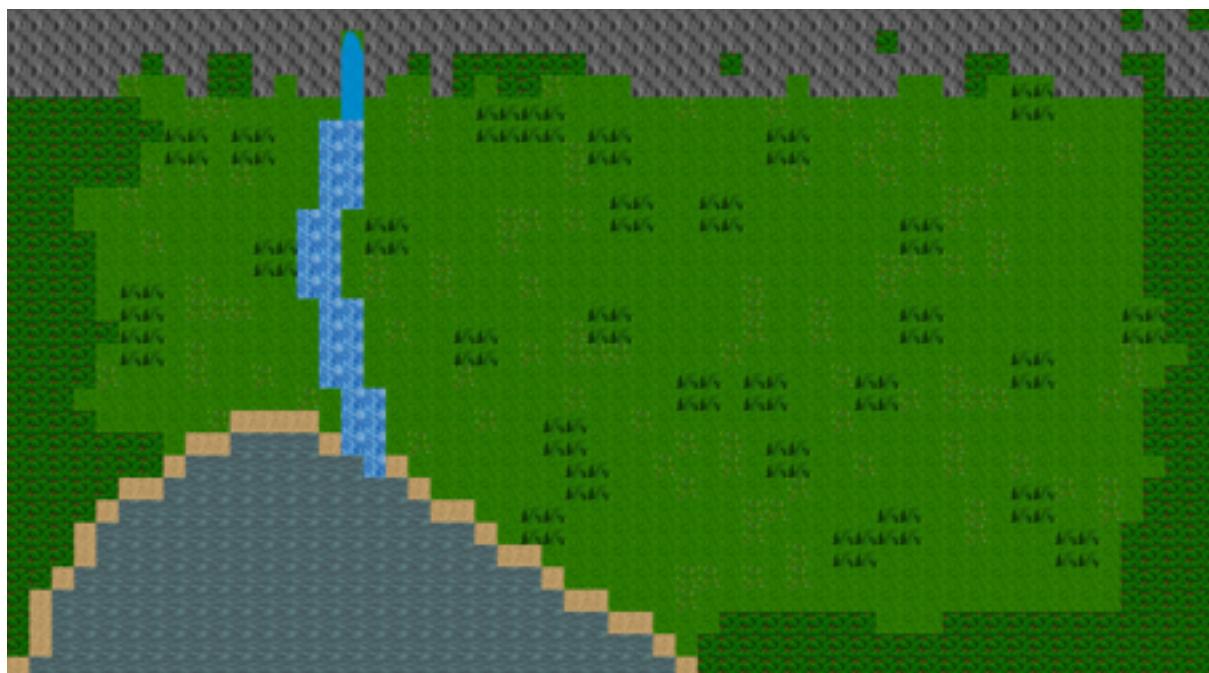


I consulted my stakeholders for their opinion as to whether they thought the two maps with the same test data were different enough to say that the maps are randomly generated. The two maps that were generated are shown below:

First map:



Second map:



My stakeholders agreed that the two maps were significantly different and so they agreed that the map generates in a random fashion. Hence this test passes.

UF3.1

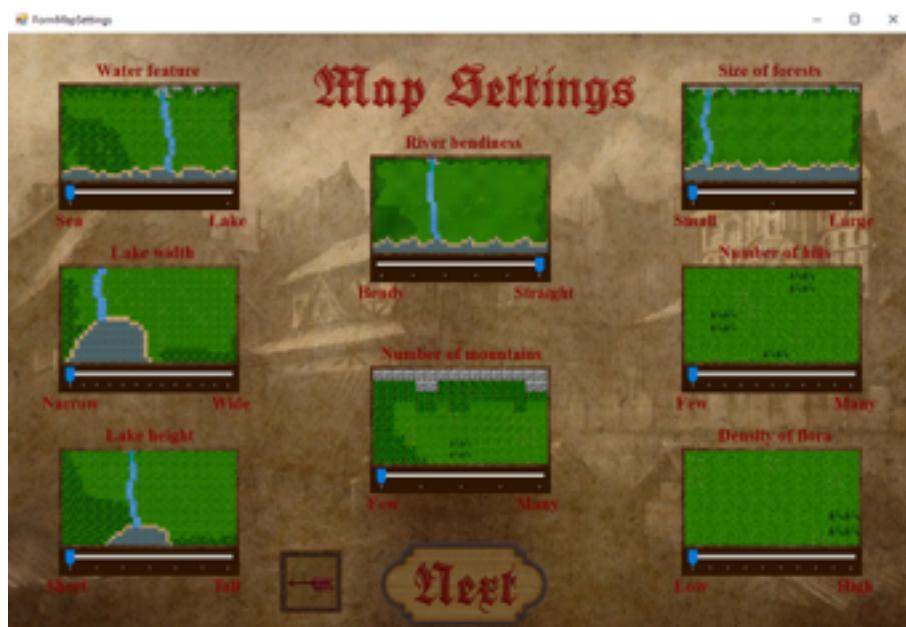
This test was captured using Grabilla. This test can be viewed in the file called UF3.1.mp4. For this test the interval of tmrGameTimer was reduced to 50 to speed up the test. The date is correctly formatted throughout the test and all possible months (including February on a leap year and a non leap year) were demonstrated to have the correct number of days. Hence this test passes. The value for the interval of tmrGameTimer was returned to 1000 after this test.

UF3.2

This test was captured using Grabilla. This test can be viewed in the file called UF3.2.mp4. For this test I invited a stakeholder to try using the map navigation to move around the map and discuss with the other stakeholders whether the navigation was appropriate. I recorded the stakeholder doing this. My stakeholders then agreed that the navigation was appropriate and so this test is a success.

UF3.3

For this test I input the test data into the map settings menu and showed the resulting map to my stakeholders where they considered if the map met their expectation. Evidence of the input test data is shown below:

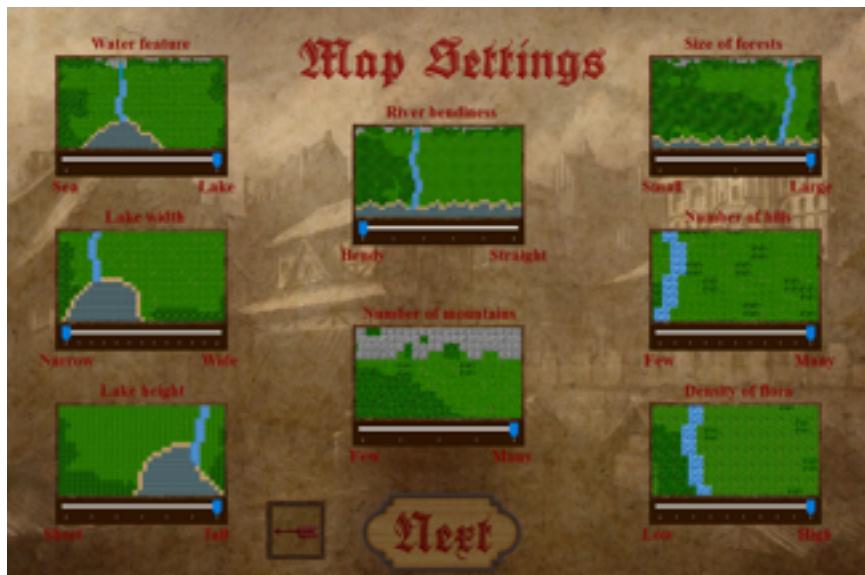


The map that was generated from these test data is shown on the following page.



My stakeholders unanimously agreed that the features of this map accurately reflected their expectations of the map generation based upon the input map data.

Evidence of the test data being input for the second map is shown below:



The map that was generated is shown below:



Once again my stakeholders all agreed that the test accreted reflected their exceptions based upon the input test data. However my stakeholders were concerned that all the plains tiles were replaced by flora with this setting; and in this implementation of the solution buildings cannot be placed upon flora, so players using this setting would not be able to build anything. In response I said that buildings would be able to be placed upon flora in a later version of the game. As my stakeholders agreed that the map settings were reflected in the map, this test is a success.

UF4.1 and UF4.3

This test was captured using Grabilla. This test can be viewed in the file called UF4.1 UF4.3.mp4. The video demonstrates that each civilisation has access to a unique building and that there is at least one building in each level of civilisation. Thus is test is successful.

UF4.2

This test was captured using Grabilla. This test can be viewed in the file called UF4.2.mp4. The video demonstrates that 10 unique buildings can be placed, thus is test is successful.

Since all the usability features tests passed (or in the case of UF1.2, partially passed) I can conclude the usability features testing and discuss my evaluation of the project and the solution.

Evaluation of the Solution

Now that the post-development and the usability features testing is complete, I can draw upon the conclusions of those tests to help inform my evaluation of the solution.

Success of the solution

I will compare the final implementation of the solution to the success criteria and the success criteria requirements. By referencing the results of the during development and post development testing I can evaluate whether the requirements of the success criteria have been met and hence whether the success criteria have been fully, partially or not met. The table on the following page lists the success criteria requirements; the during development and post development tests that demonstrate the implementation of the requirement in the solution;

and whether or not the success criteria requirements have been fully, partially or not met. The justification for why the listed tests demonstrate the implementation of the respective requirement is documented after the table.

Success criteria requirement	During development and post development tests that demonstrate the implementation of this requirement in the solution	Success criteria requirement met? (Fully, partially or not met)
SCR1.1	PD1.1 DD1.1; DD1.2; DD1.4	Fully.
SCR1.2	PD1.2 DD1.1; DD1.2; DD1.4	Fully.
SCR1.3	PD1.3 DD1.1; DD1.2; DD1.4	Fully.
SCR2.1	PD4.1	Fully.
SCR2.2	N/A.	Not met.
SCR2.3	N/A.	Not met.
SCR3.1	PD1.4	Fully.
SCR3.2	PD1.5	Fully.
SCR4.1	PD2.1.1; PD2.1.2; PD2.1.3	Fully.
SCR4.2	PD2.2.1; PD2.2.2; PD2.2.3 DD2.10; DD2.11; DD2.12	Fully.
SCR4.3	PD2.3	Fully.
SCR4.4	PD2.4	Fully.
SCR4.5	PD2.5	Fully.
SCR4.6	PD2.6	Fully.
SCR4.7	PD2.7	Fully.
SCR5.1	N/A.	Not met.
SCR5.2	PD1.6	Fully.
SCR5.3	PD1.7	Fully.
SCR6.1	N/A.	Not met.
SCR6.2	N/A.	Not met.
SCR6.3	N/A.	Not met.
SCR7.1	PD1.8	Fully.
SCR7.2	PD1.9	Fully.
SCR7.3	PD1.10; PD3.1	Fully.
SCR8.1	PD1.11	Fully.
SCR8.2	PD1.12	Fully.
SCR8.3	PD1.13	Fully.

Success criteria requirement	During development and post development tests that demonstrate the implementation of this requirement in the solution	Success criteria requirement met? (Fully, partially or not met)
SCR9.1	PD1.14	Fully.
SCR9.2	PD1.15	Fully.
SCR9.3	PD1.16	Fully.
SCR10.1	PD1.17 DD1.5	Fully.
SCR11.1	PD4.2 DD4.2; DD4.3	Fully.
SCR11.2	N/A.	Not met.
SCR11.3	PD4.3 DD4.4	Fully.
SCR11.4	PD4.4 DD4.5	Partially.
SCR11.5	N/A.	Not met.
SCR12.1	PD4.1	Partially.
SCR12.2	N/A.	Not met.
SCR12.3	N/A.	Not met.
SCR13.1	N/A.	Not met.
SCR13.2	N/A.	Not met.
SCR13.3	N/A.	Not met.
SCR13.4	N/A.	Not met.
SCR13.5	N/A.	Not met.
SCR13.6	N/A.	Not met.
SCR14.1	N/A.	Not met.
SCR14.2	N/A.	Not met.
SCR14.3	N/A.	Not met.

Below is the justification of how each success criteria requirement is demonstrated to have been implemented by during development and post development testing.

SCR1.1

- PD1.1 demonstrates that the user can select the Seljuk Turks by clicking the relevant button on the select civilisation menu.
- DD1.1 and DD1.4 demonstrate that the user must choose a civilisation before advancing.
- DD1.2 demonstrates that selecting the Seljuk Turks civilisation gives the player visual

feedback, which helps the user choose a civilisation.

SCR1.2

- PD1.2 demonstrates that the user can select the Kingdom of England civilisation by clicking the relevant button on the select civilisation menu.
- DD1.1 and DD1.4 demonstrate that the user must choose a civilisation before advancing.
- DD1.2 demonstrates that selecting the Kingdom of England civilisation gives the player visual feedback, which helps the user choose a civilisation.

SCR1.3

- PD1.3 demonstrates that the user can select the Yuan Dynasty civilisation by clicking the relevant button on the select civilisation menu.
- DD1.1 and DD1.4 demonstrate that the user must choose a civilisation before advancing.
- DD1.2 demonstrates that selecting the Yuan Dynasty civilisation gives the player visual feedback, which helps the user choose a civilisation.

SCR2.1

- PD4.1 demonstrates that a unique building is available to each civilisation.

SCR3.1

- PD1.4 demonstrates that the user can enter text into a town name menu which becomes the name of the town.

SCR3.2

- PD1.5 demonstrates that a label exists on the in-game screen whose text is equal to the text entered in the town name menu.

SCR4.1

- PD2.1.1, PD2.1.2 and PD2.1.3 demonstrate that the three constituent parts of a river generation (river source tile, stream tiles and the main river) are generated in each map.

SCR4.2

- PD2.2.1 demonstrates that a lake is generated if the user decides to have a map with a lake.
- PD2.2.2 demonstrates that a sea is generated if the user decides to have a map with a sea.
- PD2.2.3 demonstrates that either water features is correctly lined with coastline tiles.
- DD2.10 and DD2.11 demonstrates that if a lake is generated it is generated correctly and randomly.
- DD2.12 demonstrates that if a sea is generated, then it is done so correctly.

SCR4.3

- PD2.3 demonstrates that mountain ranges are randomly generated at the top of the map.

SCR4.4

- PD2.4 demonstrates that forests are randomly generated around the remaining three edges of the map.

SCR4.5

- PD2.5 demonstrates that hills are randomly generated on the map.

SCR4.6

- PD2.6 demonstrates that flora is randomly generated around the map.

SCR4.7

- PD2.7 demonstrates that the remaining empty tiles are filled with plains tiles.

SCR5.2

- PD1.6 demonstrates that the help menu can be opened from the main menu.

SCR5.3

- PD1.7 demonstrates that the help menu can be opened from in-game.

SCR7.1

- PD1.8 demonstrates that the in-game screen correctly displays population count.

SCR7.2

- PD1.9 demonstrates that the in-game screen correctly displays wealth and income.

SCR7.3

- PD1.10 demonstrates that the in-game screen correctly displays the in-game date.
- PD3.1 demonstrates that the date is correctly formatted and that the in-game time advances at a reasonable pace.

SCR8.1

- PD1.11 demonstrates that the building menu opens correctly when the respective button is clicked.

SCR8.2

- PD1.12 demonstrates that the town management menu opens correctly when the respective button is clicked.

SCR8.3

- PD1.13 demonstrates that the treasury menu (contains detailed town information) opens correctly when the respective button is clicked.

SCR9.1

- PD1.14 demonstrates that the start menu (which contains the options specified in SCR9.1) opens correctly when the respective button is clicked.

SCR9.2

- PD1.15 demonstrates that the save game button appears correctly in the start menu.

SCR9.3

- PD1.16 demonstrates that the button on the start menu that returns the user to the main menu functions correctly.

SCR10.1

- PD1.17 demonstrates that each menu has a path back to the main menu by testing all the menu navigation buttons.
- DD1.5 demonstrates the navigation buttons on each menu working correctly.

SCR11.1

- PD4.2 demonstrates that the user has access to 10 unique buildings that can be placed.
- DD4.2 demonstrates that buildings can only be placed with the prerequisite resources.
- DD4.3 demonstrates that the user is provided with guidance to allow easier placement.

SCR11.3

- PD4.3 demonstrates that certain buildings are prevented from being placed in certain locations.
- DD4.4 demonstrates that certain buildings cannot be placed in invalid locations.

SCR11.4

- PD4.4 demonstrates that certain buildings can be successfully placed in the correct locations.
- DD4.5 demonstrates that certain buildings can be placed in valid locations.

The reason why this success criteria requirement is only partially met is described in the next section.

SCR12.1

- PD4.1 demonstrates that there is at least one building in each level of civilisation that re-enforces the concept of levels of civilisation.

The reason why this success criteria requirement is only partially met is described in the next section.

Based upon the success and failures of the success criteria requirements I can now deduce whether or not the success criteria have been fulfilled. A table illustrating this is shown on the following page.

Success Criteria	Success criterion requirements that were met	Success criteria requirements that were partially or not met	Success criterion met? (Fully, partially or not met)
SC1	SCR1.1; SCR1.2; SCR1.3	None	Fully.
SC2	SCR2.1	SCR2.2; SCR2.3	Partially.
SC3	SCR3.1; SCR3.2	None	Fully.
SC4	SCR4.1; SCR4.2; SCR4.3; SCR4.4; SCR4.5; SCR4.6; SCR4.7	None	Fully.
SC5	SCR5.2; SCR5.3	SCR5.1	Partially.
SC6	None	SCR6.1; SCR6.2; SCR6.3	Not met.
SC7	SCR7.1; SCR7.2; SCR7.3	None	Fully.
SC8	SCR8.1; SCR8.2; SCR8.3	None	Fully.
SC9	SCR9.1; SCR9.2; SCR9.3	None	Fully.
SC10	SCR10.1	None	Fully.
SC11	SCR11.1; SCR11.3	SCR11.2; SCR11.4; SCR11.5	Partially.
SC12	None	SCR12.1; SCR12.2; SCR12.3	Partially.
SC13	None	SCR13.1; SCR13.2; SCR13.3; SCR13.4; SCR13.5; SCR13.6	Not met.
SC14	None	SCR14.1; SCR14.2; SCR14.3	Not met.

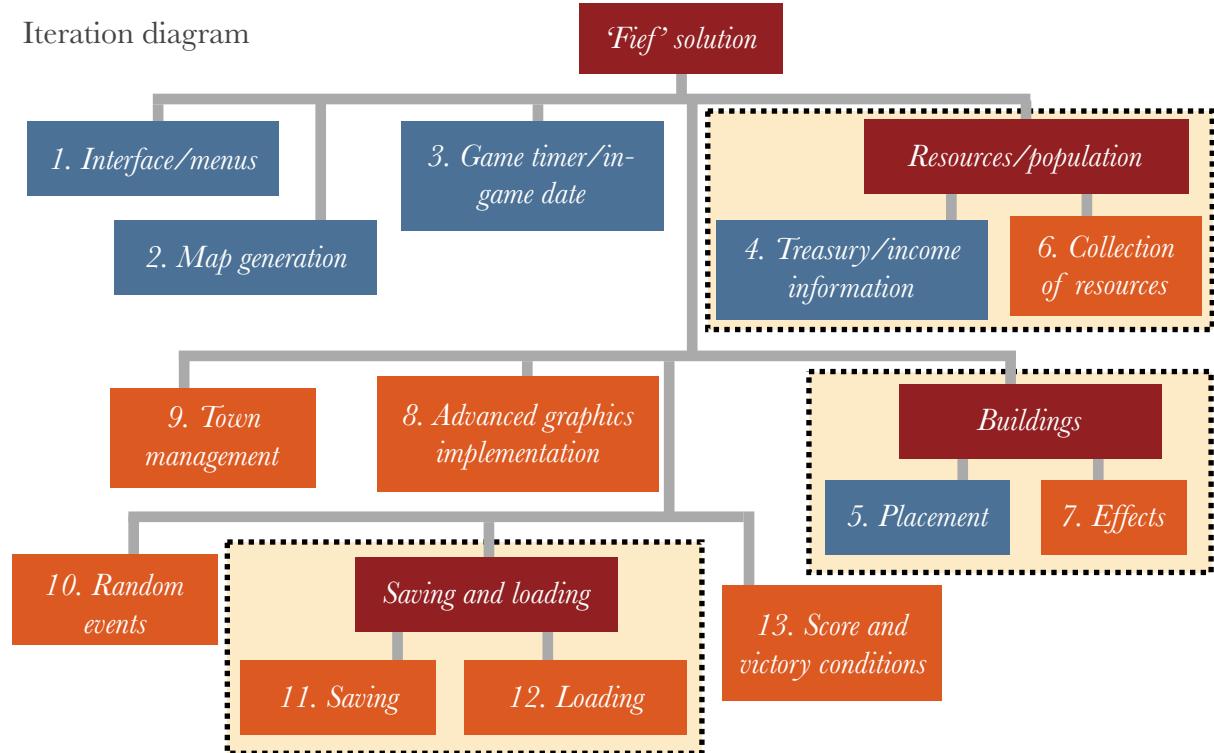
Before I discuss how partially and unmet criteria could be addressed in future development, I will briefly mention why SCR11.4 and SCR12.1 were only partially met.

As mentioned in UF3.3, buildings cannot currently be placed upon flora tiles, which is an issue when the entire map has flora tiles instead of plains tiles. It was intended that buildings could be placed upon flora tiles, however this implementation of the solution does not include this feature. Hence SCR11.4 is only partially successful.

SCR12.1 required three levels of civilisation to be implemented. Whilst the foundations of this were laid in this version (as seen in the building menu) there was no mechanic in this version which implemented any differences in these civilisations. There were buildings available to each level of civilisation, but the user has access to them at the start of the game and so there was no tangible difference between the three levels of civilisation. For this reason this success criterion requirement is only partially successful.

I will now discuss how the partially and unmet criteria could be addressed in the future.

Elements of the solution that were not implemented in this version were omitted due to time constraints. Referencing a diagram I produced prior to iterative development commence (duplicated below: this will be referred to as the iteration diagram), only the content of the 1st, 2nd, 3rd, 4th, 5th (as well as the extra content related to map settings) iterations (shown in blue) were implemented (recall that some of the iterations were conglomerated).



The content of the remaining iterations were simply not implemented and thus many of the success criteria were only partially met or not met at all. I shall list the unmet and partially met success criteria requirements and describe how the content related to these unmet/partially met success criteria could be implemented hence addressing such success criteria requirements.

SCR2.2 (Not met)

The content of this success criterion requirement would have been addressed in iteration 7 of the iteration diagram. Where technology would have been implemented as part of the building's effects. Depending upon the chosen civilisation, certain buildings would give/deny access to certain technologies. This would meet this success criterion requirement.

SCR2.3 (Not met)

The content of this success criterion requirement is rather ambiguous and the bonus/disadvantage of each civilisation is still uncertain. Whilst the implementation of unique buildings and unique technology trees could be interpreted as this success criterion requirement being met, I believe that it would be disingenuous to use these as justification of this success criterion requirement being fulfilled as my stakeholders were clear that any bonus/disadvantage added should be separate from the previously mentioned variations in civilisations (unique buildings and technology trees). The content of this success criterion requirement could have been addressed in iterations 6, 7, 9 or 10 from the iteration diagram. If a later version were developed these iterations would be implemented and thus this success criterion requirement would be met.

SCR5.1 (Not met)

The content of this success criterion requirement was to be implemented as a part of iterations 6, 7, 9, 10 and 13 from the iteration diagram to describe the features added in these iterations in the help menu. As these iterations would be implemented in a future version, the help menu would have the relevant content added to it, which would meet this success criterion requirement.

SCR6.1 (Not met)

The content of this success criterion requirement was to be implemented as a part of iteration 11 in the iteration diagram. This iteration would be implemented in a future version and so this success criterion requirement would be met.

SCR6.2 (Not met)

The content of this success criterion requirement was to be implemented as a part of iteration 12 in the iteration diagram. This iteration would be implemented in a future version and so this success criterion requirement would be met.

SCR6.3 (Not met)

The content of this success criterion requirement was to be implemented as a part of iteration 11 in the iteration diagram. This iteration would be implemented in a future version and so this success criterion requirement would be met.

SCR11.2 (Not met)

The content of this success criterion requirement was to be implemented as a part of iterations 6 and 7 in the iteration diagram. As each building would either have a specific effect or would be used to collect resources, the implementation of these iterations in a future version would meet this success criterion requirement.

SCR11.4 (Partially met)

As discussed previously this success criterion requirement was only partially met as buildings cannot be placed upon flora tiles. This could be addressed in a future version so that buildings can be placed upon flora tiles and this would completely fulfil this success criterion requirement.

SCR11.5 (Not met)

The content of this success criterion requirement was to be implemented as a part of iteration 7 in the iteration diagram. If this iteration were implemented in a future version then this success criterion requirement would be met.

SCR12.1 (Partially met)

More substance would have been given to the levels of civilisation in iteration 7 and 9 in the iteration diagram. These iterations would be implemented in a later version which would fully meet this success criterion requirement.

SCR12.2 and SCR12.3 (Neither met)

The content of both these success criteria requirements would have been added in iteration 9 of the iteration diagram. As this would be implemented in a future version these success criterion requirements would be met.

SCR13.1, SCR13.2, SCR13.3, SCR13.4, SCR13.5, SCR13.6 (None met)

The content of all these success criteria requirements were to be addressed in the final iteration of the iteration diagram. As this iteration would be implemented in a later version these success criteria requirements would all be met.

SCR14.1, SCR14.2, SCR14.3 (None met)

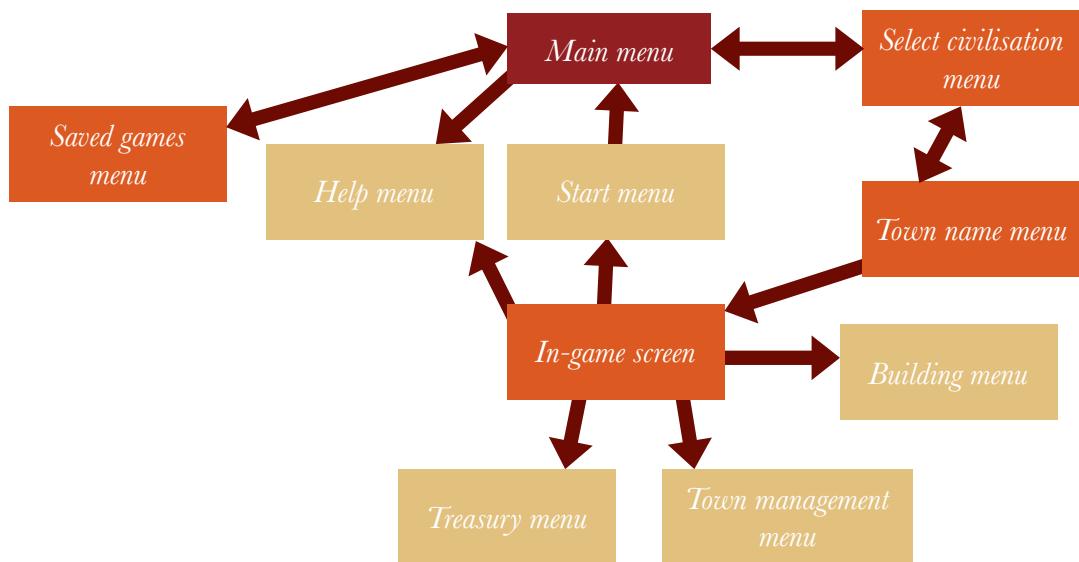
The content of all these success criteria requirements were to be addressed in iteration 10 of the iteration diagram. As this iteration would be implemented in a later version these success criteria requirements would all be met.

Describe the final product

I will compare the usability features that were planned to be implemented in each iteration as described in the ‘Describe the solution (1)’ section of each iteration to the actual implementation of the usability features of each iteration. I will do so by examining each iteration’s usability features individually.

First iteration:

The first iteration implemented much of the menus and user interface that remain in the final solution. I described the navigation of the menus in the first iteration using a diagram which I have duplicated below:



If compared to the current implementation, it is almost identical to the current implementation. Navigation between menus works in the same way as specified in the ‘Describe the solution (1)’ section of this iteration, however in the 3rd iteration the map settings menu was added which fitted in between the select civilisation menu and the town name menu. Whilst the implementation is different to what was described in iteration 1, it is still functional in the same way and the content of the other menus has been unaffected by the addition of the map settings menu. The navigation between the menus has been shown

to work in many tests including UF1.3, PD1.6, PD1.7, PD1.11, PD1.12, PD1.13, PD1.14, PD1.15, PD1.16 and PD1.17. This aspect of the usability features is a success.

I also designed many wireframes in the first iteration and described the functionality of each menu. The comparisons between these designs and the final implementation of these menus will be done by examining each menu and form individually.

Main menu:

The original wireframe design of the menu is shown below:



The final version of the main menu is shown below:



The background has three possible images which change randomly every time the menu is loaded. One of them is shown in the example above. In terms of functionality it is identical to the wireframe design. The functionality of the main menu is demonstrated to work in such

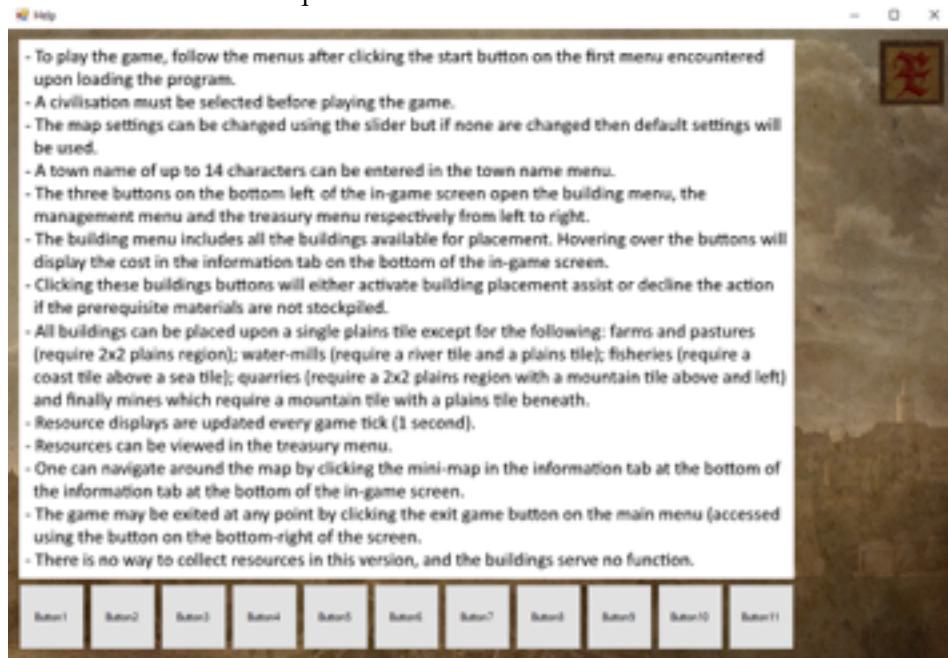
a way in PD1.17. As the final implementation of the main menu is identical to that originally described, this aspect is a success.

Help menu:

The original wireframe design of the help menu is shown below:



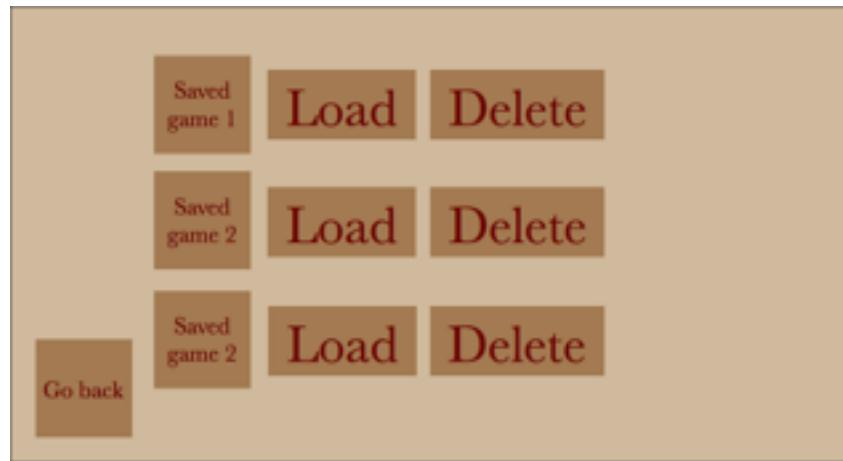
The final version of the help menu is shown below:



The design of this menu is identical to that of the wireframe and the close menu button works as specified. However none of the help buttons do anything and currently is functionally pointless. This menu requires far more work in later versions before this usability feature can be considered a success. As mentioned when discussing what action could be taken in future versions to fulfil SCR5.1 in the previous section, the functionality of this menu would be implemented in later versions. To facilitate the usage of the program in its current form, some instructions to aid the user have been added. This usability feature is only partially successful.

Saved games menu:

The original wireframe design of the saved games menu is shown below:



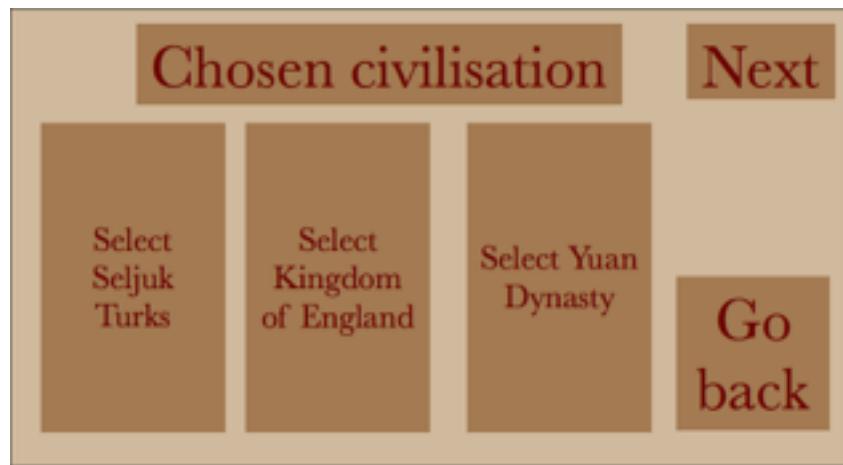
The final version of the saved games menu is shown below:



In the same way that the help menu is functionally pointless in this version of the solution, the saved games menu serves no purpose other than laying the foundations of a saved games feature. The navigation and design of the menu is identical to that originally described. As mentioned when describing how SCR6.2 and SCR6.3 could be implemented in future versions, the functionality of this menu would be implemented in a future version. As the menu serves no function this usability feature is only a partial success.

Select civilisation menu:

The wireframe and final version of the select civilisation menu are shown on the following page. This menu's design is very similar to the final version of the menu.



The only difference in design is the way the chosen civilisation is displayed. In the final version an icon temporarily appears when the mouse hovers over a civilisation's button, and changes permanently when a civilisation is chosen. These icons are shown below:

Kingdom of England:



Seljuk Turks:



Yuan Dynasty:



These provide visual feedback on the civilisation chosen. The navigation is also identical to that described originally. Finally the user is prevented from advancing until a civilisation is

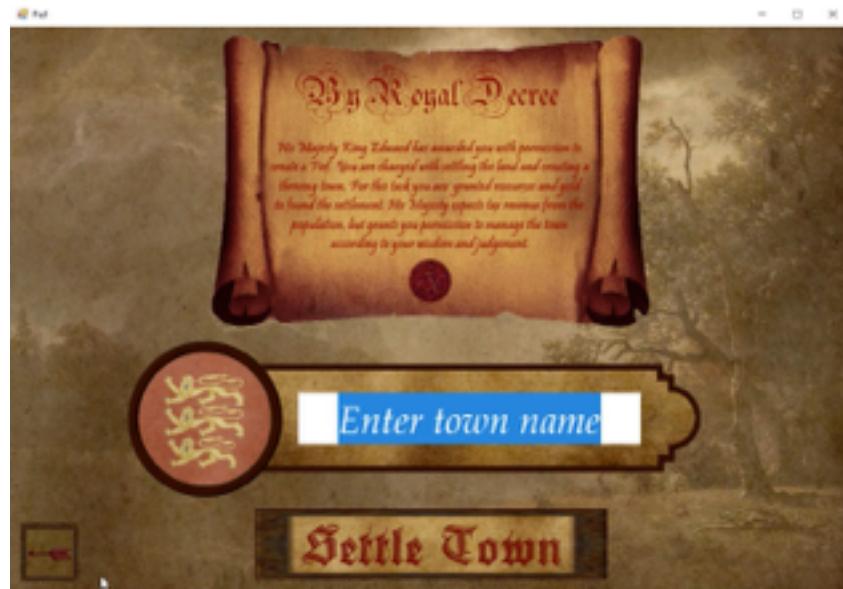
chosen. The final implementation of this menu fits the original description as so this usability feature is a success.

Town name menu:

The original wireframe design of the menu is shown below:



The final version of the menu is shown below:



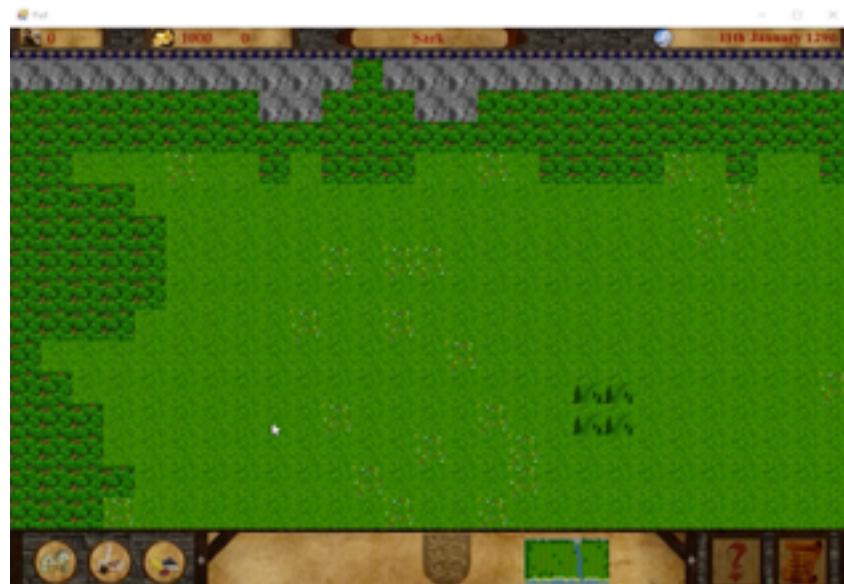
The design and functionality of the menu is identical to that specified in the describe the solution (1) section of iteration 1. The navigation works as originally described. There is also a 14 character limit of the town name as specified. there are two other designs of this menu based upon the civilisation chosen. As the final version of this menu is the same as described, this usability feature is a success.

In game form:

The original wireframe design of the menu is shown below:



The final version of the in game form is shown below:



There are many similarities between the original design and the final version. The information display on the top of the screen is as described originally. The menus available from at the bottom of the screen are also identical. The major difference is the mini-map which was added in the 3rd iteration. Another difference is the icon that illustrates the month. This was not mentioned in the original description of this form. Despite these edits to the design, all the features that were specified in the describe the solution (1) section of the 1st iteration are in the final version, hence this usability feature is a success.

Start menu:

The original design and the final version of this menu are shown on the following page

on the left and right respectively.



The layout of the start menu in the final version is identical to that originally described. The close menu and exit game buttons are both functional, however the save game button does nothing in the final version and so this usability feature is only partially successful.

Building menu:

The original design and the final version of this menu are shown below on the left and right respectively.



In accordance with the original description, the final version of the building menu contains the required 10 unique buildings. Also, hovering over each button displays the cost in the information section of the in-game screen. The original description made no reference to the unique buildings, however they have been added in later iterations. The final version also includes the 3 groups of buildings under each level of civilisation. As the final version

includes the 3 groups of buildings under each level of civilisation. As the final version of the building menu includes all the features described originally this usability feature is a success.

Treasury menu:

The original design and the final version of this menu are shown below on the left and right respectively.

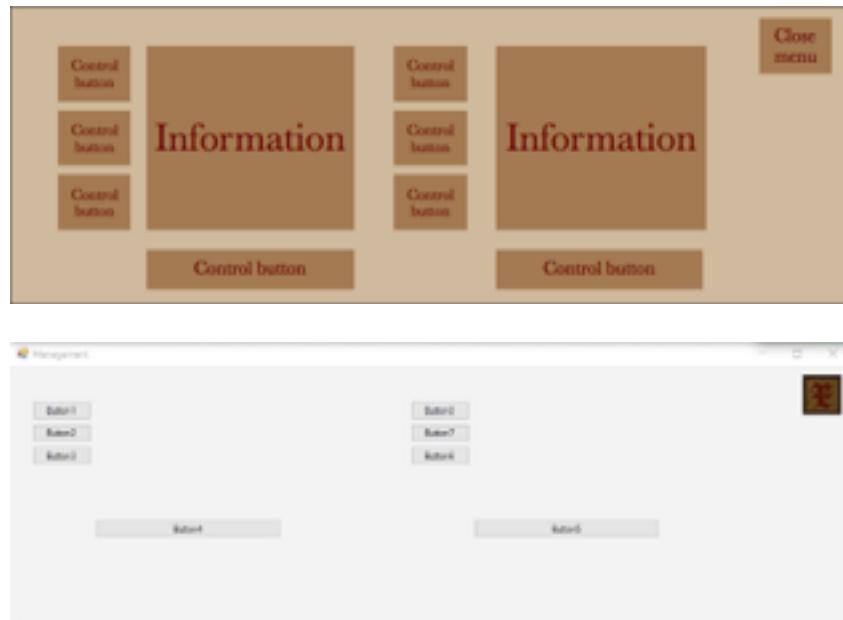


The original description of the treasury menu was admitted rather vague. However, the information that appears in the final version matches the information that was required by the original description. It is not particularly reasonable to directly compare these the original description of the menu to the final version as this menu was overhauled in the 4th iteration so it is of no surprise that these design are unfamiliar.

Much of the content of the final version of this menu has not been implemented to be functional yet. Whilst the resource stockpiles and gold count is functional, the tax levels and upkeep are not functional at all. For this reason this usability feature is only partially successful. As tax and upkeep would have been implemented in the building effects and the town management iterations as seen on the iteration diagram, and these iterations would be implemented in a future version, this usability feature would be made fully functional in that later version.

Management menu:

The original design and the final version of this menu are shown on the following page.



The final version of the management menu is utterly functionless. The contents of this menu were to be added in the town management iteration as seen in the iteration diagram. It is meaningless to discuss layout of this menu as the controls seen in the final version are arbitrary and serve no purpose. Thus this usability feature's implementation is a failure. This would be implemented in a future version in which the town management iteration is implemented.

Second iteration:

The second iteration implemented map generation. Unlike the first iteration there were no wireframe designs in this iteration so it is more difficult to make tangible comparisons.

The describe the solution (1) section of the 2nd iteration described the map array as having 30 rows and 54 columns. The final solution used an array that had 32 rows and 56 columns to facilitate a border to make map generation simpler. This was an improvement over the original description. The final version's map array utilised object oriented programming in the same way that was originally described. As the final version is sufficiently alike to the original description this usability feature is a success.

The size of the viewable map and the visible area in the final version is identical to the size described in the original version. This usability feature is also a success.

The texture set used to implement the graphics of the map is almost the same as originally described. The only difference is the mountain tile texture which I decided to change to accommodate better graphics for the mine and quarry buildings. This difference is

not significant enough to make this usability feature a failure. The image that I produced at the end of the describe the solution (1) section of iteration 2 is remarkably similar to what the map actually looks like. The similarities are sufficient for this usability feature to be successful.

Third iteration:

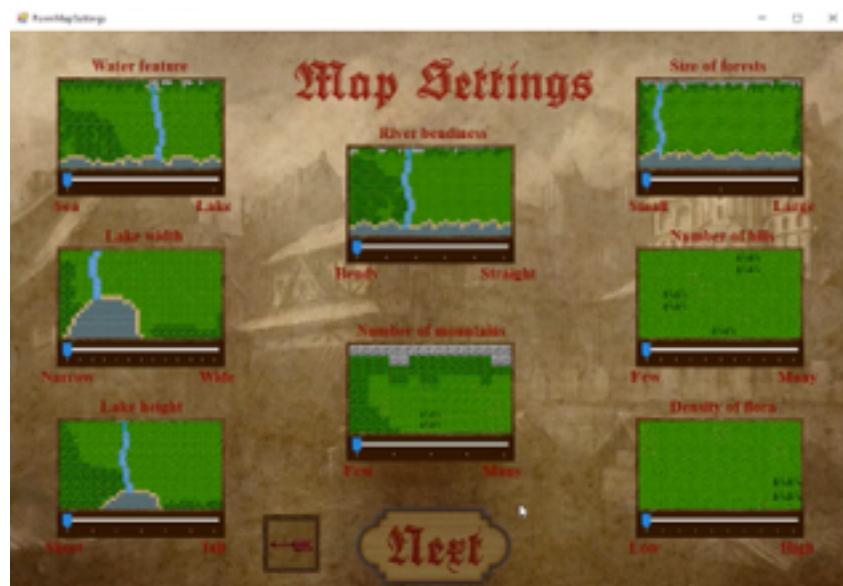
The third iteration implemented three main features: the map settings menu, the mini-map navigation and the in-game date. The usability features associated with these will be examined individually.

Map settings menu:

Below is the original wireframe design of the map settings menu:



Below is the final version of the map settings menu:



As originally described, the options were implemented using track bars. The options

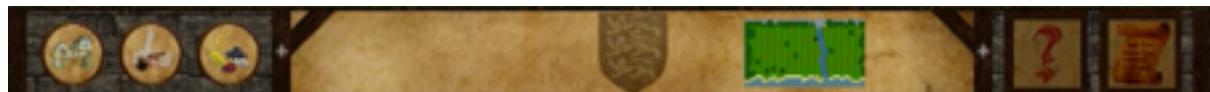
available to the user are the same as those specified. Each option has a dynamic visual as described by the wireframe design. As mentioned when discussing the usability features of the 1st iteration, the map settings menu was added into the game setup in-between the select civilisation menu and the town name menu as specified originally. As the final version sufficiently matches the description in the describe the solution (1) of the 3rd iteration this usability feature is a success.

Mini-map and navigation:

The mini-map was proposed to appear at the bottom of the in-game screen in the place illustrated by this image:



The final implementation of the map utilised this proposal and the mini-map appears in the same place in the final version of the solution as seen below:



The mini-map is made up of 4 picture boxes that have a representation of the map printed to them in the final version as described originally. Each has a click event associated with each of them to load the respective part of the map. The final version of this usability feature is functionality identical to the specified version in the describe the solution (1) of the 3rd iteration. Hence this usability feature is a success.

In-game date:

The in-game date is calculated using a timer as specified originally. The increment rate of the timer is identical to that described originally. The icons that were designed are in the final version as well. The in-game date has been tested multiple times and has been demonstrably proven to format and display correctly. This usability feature is a success.

Fourth iteration:

The fourth iteration overhauled the treasury menu and implemented building placement as well as resources. The usability features of this iteration will be analysed

individually.

Treasury menu and resources:

This usability feature has already been analysed in the first iteration, but there were a few extra features added to the treasury menu in the 4th iteration that should be addressed. A wireframe design of the treasury menu from the fourth iteration and the final version are shown below:



The 10 unique resources specified in the fourth iteration have all been implemented in the treasury menu as well as the gold count. As previously mentioned, the tax and upkeep features are to be added in a later version. The icons designed for the resources are in the final version. The amount of starting resources will need to be balanced in later versions and extra resources will need to be added. Due to the amount of the elements that must be added in later versions, this usability feature is only partially successful.

Building placement:

The describe the solution (1) section of the fourth iteration described the cost of each building and the icon to be used for each building. These values of cost have been successfully implemented in the final version of the solution. The final version also included unique buildings (although they cannot be placed).

The describe the solution (1) section of the fourth iteration also described the functionality of the buildings, which were not in the final version of the solution. This should mean that this usability feature is only partially successful, however as the effects of buildings were not to be added until a later iteration, it was preemptive of me to include the functionality of the buildings in the usability features of the fourth iteration; thus this will not effect the success of this usability feature.

The buildings in the final version are all of the same size as specified and can only be placed in the locations described originally. The graphics designed for the buildings are also all the same. The similarities of the original description and the final version are sufficient for this usability feature to be successful.

Finally the placement guide was also successfully implemented and gives a guide as to the size of the building. However whilst my stakeholders were testing the solution they said that they had no in-game help as to the permitted locations of each building. They suggested in a future version I implement a feature which changes the colour of the placement guide according to where the building is allowed to be placed. However despite this future amendment, this usability feature matches the original description and so the usability feature is a success; although the fact that there will need to be future changes made to this usability feature is significant.

I have discussed the effectiveness of all the usability features that have been implemented in this version of the solution. However before I continue to discuss maintenance and development, there are two major issues that have been discovered with the final version of the solution which were rather unprecedented. The two bugs that my stakeholders found did not occur on the computer I used to program the solution, but occurred on the computers they used to test the solution.

First whenever the map is loaded for the first time, the map display is replaced with a white screen and the user has to click the mini-map to bring up the the map display. This is not a serious issue but must be solved in a later version.

The second is a problem which is specific to the graphics implementation that Visual Studio 2015 uses called GDI+. Whenever the game is exited and an attempt is made to start a new game, when the in-game screen is loaded, all graphics are replaced with red error crosses and the solution crashes. There doesn't seem to be any good reason for this and I have sought advice from other programmers who have found the problem baffling. I will need to

continue my investigation into this problem in a later version. Note: this error does not occur if the program is restarted before attempting to start a new game.

My stakeholders agreed that these two problems were not significant enough for them to reject the final solution.

Maintenance and Development.

If the solution is to be revisited in later versions the solution must be maintainable. I will discuss any maintenance issues and such limitations with this version of the solution.

Maintenance issues and limitations and remedial development

There are some issues with the code in its current form that will make maintenance more difficult and will prove to be limitations to the solution. Each will be listed with examples and a description of how the program could be developed to solve these issues. These are as follows:

1. In the code there are instances of numbers being used without any explanation as to why those numbers are being used.
 - Examples:
 - FormBuildingMenu uses seemingly arbitrary numbers to reposition the form;
 - In FormBuildingMenu the parameters being sent to SubCheckBuildingCost are numbers that have no explanation to their value, one has to check the function to understand what the values represent;
 - The values that represent the civilisations are arbitrary and it would be difficult for other programmers to understand what the values represent;
 - FormInGame uses seemingly arbitrary numbers to reposition the screen;
 - Whenever subPrintMap or subPrintMinimap subroutines are called the parameters passed are numbers which to another programmer may seem arbitrary;
 - Instances of numbers being used in my solution to calculate the in-game date may cause confusion among other programmers;
 - FormStartMenu uses seemingly arbitrary numbers to reposition the form;
 - FormTreasuryMenu uses seemingly arbitrary numbers to reposition the form;
 - subPrintMinimap utilises seemingly arbitrary numbers to reference textures to print to the mini-map;
 - FormManagementMenu uses seemingly arbitrary numbers to reposition the form.

- This could be solved by defining several constants and assigning the respective value and using these constants in the code instead of numbers. This would make the code more readable and easier to understand.

2. Some of the code is rather clumsy and could be optimised and simplified.

- Examples:

- The sections of ModuleMapGeneration that generate the mountains and the forests are particularly clumsy.
- Some of the map generation is bias according to how the array is looped through. For example when forests are generated the array is looped through from left to right, meaning that large forests cannot be generated on the right hand side.
- The code to place a building upon a mouse click is very clumsy. The code checking if the terrain is valid for the building would be rather unclear to another programmer.

- This could be solved by redesigning the algorithms with greater care, or adding more comments to make the code's purpose clearer.

- Problems relating to generation bias could be resolved by rewriting the code so that each half of the map is generated separately. For example the forests on the left hand side of the map could be generated with a left bias (i.e. loop through the array from left to right) whilst the forests on the right hand side could be generated with a right bias (i.e. looping through the array right to left).

3. Resources (images) have been added to the project file using a project resource file rather than adding the resources as local resources. This has the effect of increasing compiling time and thus making the solution harder to maintain.

- If the resources were changed to local resources the time required to compile the solution would dramatically decrease.
- There are also some redundant resources that could be removed altogether which would reduce processor load.

Further development related to improvements

In the next version of the solution much of the proposed content would be added.

Unimplemented iterations would be implemented to complete the game. Other features such

as more civilisations; more buildings and resources; a larger map and more advanced combat could be added. These additions would give the game more depth and longevity. A later version could also attempt to optimise the current implementation and make the program run faster. I would suggest that a future version be developed on a different platform as the Visual Studio 2015 GDI+ graphics system is rather inadequate in my opinion. The current version of the solution implements a convincing foundation of a town builder game that has a large amount of room for development, an aspect of solutions that programmers often strive for.

END OF EVALUATION

APPENDIX

Code for FormBuildingMenu

Public Class FormBuildingMenu

Private Sub FormBuildingMenu_Load(sender As Object, e As EventArgs)
Handles MyBase.Load

'Position the building menu to a convenient location, assuming in-game
form is not moved (start location is centre and is repositioned):

 Me.Top -= 150

 Me.Left -= 715

End Sub

Private Sub btnExitBuildingMenu_Click(sender As Object, e As EventArgs)
Handles btnExitBuildingMenu.Click

'Hides the building menu:

 Me.Close()

End Sub

Private Sub btnCruckHut_Click(sender As Object, e As EventArgs) Handles
btnCruckHut.Click

'Check that the user has enough wood, stone, iron and gold; resources
and gold costs are passed as parameters to the SubCheckBuildingCost
subroutine respectively:

 If SubCheckBuildingCost(5, 0, 0, 10) = True Then

 'Update relevant variables so the solution knows which building is to
 be placed and to enable placement assist:

 FormInGame.strCurrentBuilding = "Cruck Hut"

 FormInGame.blnBuildingSelected = True

 End If

 End Sub

Private Sub btnFarm_Click(sender As Object, e As EventArgs) Handles
btnFarm.Click

'Check that the user has enough wood, stone, iron and gold; resources
and gold costs are passed as parameters to the SubCheckBuildingCost
subroutine respectively:

 If SubCheckBuildingCost(15, 0, 0, 100) = True Then

 'Update relevant variables so the solution knows which building is to
 be placed and to enable placement assist:

 FormInGame.strCurrentBuilding = "Farm"

 FormInGame.blnBuildingSelected = True

```
    End If  
End Sub
```

```
Private Sub btnFishery_Click(sender As Object, e As EventArgs) Handles  
btnFishery.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(20, 0, 0, 150) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
    FormInGame.strCurrentBuilding = "Fishery"
```

```
    FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnWaterMill_Click(sender As Object, e As EventArgs) Handles  
btnWaterMill.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(50, 0, 0, 150) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
    FormInGame.strCurrentBuilding = "Water Mill"
```

```
    FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnQuarry_Click(sender As Object, e As EventArgs) Handles  
btnQuarry.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(100, 0, 0, 500) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
    FormInGame.strCurrentBuilding = "Quarry"
```

```
    FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnPasture_Click(sender As Object, e As EventArgs) Handles  
btnPasture.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(25, 0, 0, 250) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Pasture"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnLumberjackHut_Click(sender As Object, e As EventArgs)  
Handles btnLumberjackHut.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(10, 0, 0, 50) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Lumberjack Hut"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnCottage_Click(sender As Object, e As EventArgs) Handles  
btnCottage.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(2, 3, 0, 200) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Cottage"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnReeve_Click(sender As Object, e As EventArgs) Handles  
btnReeve.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(50, 20, 0, 300) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Reeve House"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnBrewery_Click(sender As Object, e As EventArgs) Handles btnBrewery.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(20, 0, 0, 250) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Brewery"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Sub btnMine_Click(sender As Object, e As EventArgs) Handles btnMine.Click
```

'Check that the user has enough wood, stone, iron and gold; resources and gold costs are passed as parameters to the SubCheckBuildingCost subroutine respectively:

```
If SubCheckBuildingCost(200, 100, 0, 750) = True Then
```

'Update relevant variables so the solution knows which building is to be placed and to enable placement assist:

```
FormInGame.strCurrentBuilding = "Mine"
```

```
FormInGame.blnBuildingSelected = True
```

```
End If
```

```
End Sub
```

```
Private Function SubCheckBuildingCost(ByVal intWoodCost As Integer, ByVal intStoneCost As Integer, ByVal intIronCost As Integer, ByVal intGoldCost As Integer)
```

'Check wood cost against wood stockpile:

```
If FormInGame.intWoodStockpile < intWoodCost Then
```

```
MessageBox.Show("You require more wood")
```

```

        Return False
    End If
    'Check stone cost against stone stockpile:
    If FormInGame.intStoneStockpile < intStoneCost Then
        MessageBox.Show("You require more stone")
        Return False
    End If
    'Check iron cost against iron stockpile:
    If FormInGame.intIronStockpile < intIronCost Then
        MessageBox.Show("You require more iron")
        Return False
    End If
    'Check gold cost against gold count:
    If FormInGame.intGoldCount < intGoldCost Then
        MessageBox.Show("You require more funds")
        Return False
    End If
    'Check that the user has the required resources:
    If FormInGame.intWoodStockpile >= intWoodCost And
FormInGame.intStoneStockpile >= intStoneCost And
FormInGame.intIronStockpile >= intIronCost And FormInGame.intGoldCount
>= intGoldCost Then
        Return True
    End If
End Function

```

```

Private Sub btnCruckHut_MouseHover(sender As Object, e As EventArgs)
Handles btnCruckHut.MouseHover
    'Displays the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "5"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "10"
End Sub
Private Sub btnCruckHut_MouseLeave(sender As Object, e As EventArgs)
Handles btnCruckHut.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "

```

End Sub

```
Private Sub btnFarm_MouseHover(sender As Object, e As EventArgs)
Handles btnFarm.MouseHover
    'Displays the cost of the building in the information tab on the in-game
    screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "15"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "100"
End Sub
Private Sub btnFarm_MouseLeave(sender As Object, e As EventArgs)
Handles btnFarm.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
    screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

Private Sub btnLumberjackHut_MouseHover(sender As Object, e As EventArgs)
Handles btnLumberjackHut.MouseHover
    'Displays the cost of the building in the information tab on the in-game
    screen:
    FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image
    FormInGame.lblResource1.Text = "10"
    FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image
    FormInGame.lblResource2.Text = "50"
End Sub
Private Sub btnLumberjackHut_MouseLeave(sender As Object, e As EventArgs)
Handles btnLumberjackHut.MouseLeave
    'Removes the cost of the building in the information tab on the in-game
    screen:
    FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource1.Text = " "
    FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image
    FormInGame.lblResource2.Text = " "
End Sub

Private Sub btnCottage_MouseHover(sender As Object, e As EventArgs)
Handles btnCottage.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "2"  
FormInGame.pbResource2.Image = FormImgRef.pbStoneIcon.Image  
FormInGame.lblResource2.Text = "3"  
FormInGame.pbResource3.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource3.Text = "200"
```

End Sub

Private Sub btnCottage_MouseLeave(sender As Object, e As EventArgs)
Handles btnCottage.MouseLeave

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""  
FormInGame.pbResource3.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource3.Text = ""
```

End Sub

Private Sub btnBrewery_MouseHover(sender As Object, e As EventArgs)
Handles btnBrewery.MouseHover

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "20"  
FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource2.Text = "250"
```

End Sub

Private Sub btnBrewery_MouseLeave(sender As Object, e As EventArgs)
Handles btnBrewery.MouseLeave

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""
```

End Sub

Private Sub btnQuarry_MouseHover(sender As Object, e As EventArgs)
Handles btnQuarry.MouseHover

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "100"  
FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource2.Text = "500"
```

End Sub

```
Private Sub btnQuarry_MouseLeave(sender As Object, e As EventArgs)  
Handles btnQuarry.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""
```

End Sub

```
Private Sub btnReeve_MouseHover(sender As Object, e As EventArgs)  
Handles btnReeve.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "50"  
FormInGame.pbResource2.Image = FormImgRef.pbStoneIcon.Image  
FormInGame.lblResource2.Text = "20"  
FormInGame.pbResource3.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource3.Text = "300"
```

End Sub

```
Private Sub btnReeve_MouseLeave(sender As Object, e As EventArgs)  
Handles btnReeve.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""  
FormInGame.pbResource3.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource3.Text = ""
```

End Sub

```
Private Sub btnFishery_MouseHover(sender As Object, e As EventArgs)  
Handles btnFishery.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "20"  
FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource2.Text = "150"
```

End Sub

Private Sub btnFishery_MouseLeave(sender As Object, e As EventArgs)
Handles btnFishery.MouseLeave

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""
```

End Sub

Private Sub btnMine_MouseHover(sender As Object, e As EventArgs)
Handles btnMine.MouseHover

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "200"  
FormInGame.pbResource2.Image = FormImgRef.pbStoneIcon.Image  
FormInGame.lblResource2.Text = "100"  
FormInGame.pbResource3.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource3.Text = "750"
```

End Sub

Private Sub btnMine_MouseLeave(sender As Object, e As EventArgs)
Handles btnMine.MouseLeave

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""  
FormInGame.pbResource3.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource3.Text = ""
```

End Sub

Private Sub btnPasture_MouseHover(sender As Object, e As EventArgs)
Handles btnPasture.MouseHover

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "25"  
FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource2.Text = "250"
```

End Sub

```
Private Sub btnPasture_MouseLeave(sender As Object, e As EventArgs)  
Handles btnPasture.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""
```

End Sub

```
Private Sub btnWaterMill_MouseHover(sender As Object, e As EventArgs)  
Handles btnWaterMill.MouseHover
```

'Displays the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbWoodIcon.Image  
FormInGame.lblResource1.Text = "50"  
FormInGame.pbResource2.Image = FormImgRef.pbGoldIcon.Image  
FormInGame.lblResource2.Text = "150"
```

End Sub

```
Private Sub btnWaterMill_MouseLeave(sender As Object, e As EventArgs)  
Handles btnWaterMill.MouseLeave
```

'Removes the cost of the building in the information tab on the in-game screen:

```
FormInGame.pbResource1.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource1.Text = ""  
FormInGame.pbResource2.Image = FormImgRef.pbEmpty.Image  
FormInGame.lblResource2.Text = ""
```

End Sub

End Class

Code for FormCivilisationMenu

```
Public Class FormCivilisationMenu
```

```
Private Sub FormCivilisationMenu_Load(sender As Object, e As EventArgs)  
Handles MyBase.Load
```

```
'Initialise civilisation variable, the civilisation display icon and show the user that they cannot press the start game button.
```

```
Me.pbCivdisplay.Image = Nothing  
FormInGame.intCivilisation = 0  
btnStartGame2.Cursor = Cursors.No  
End Sub
```

```
'The code for the select England interface and buttons.
```

```
Private Sub btnSelectEngland_Click(sender As Object, e As EventArgs)  
Handles btnSelectEngland.Click  
    'Provide the user with feedback that they have selected England using pbCivDisplay:
```

```
    Me.pbCivdisplay.Image = FormImgRef.pbSelectEnglandIcon.Image  
    'Provide a backup of the image of pbCivDisplay for the mouse hover/leave events:  
    FormImgRef.pblImageTempSave.Image =  
    FormImgRef.pbSelectEnglandIcon.Image  
    'Update the civilisation variable to the value for England (1):  
    FormInGame.intCivilisation = 1  
    'Show the user they can now start the game:  
    btnStartGame2.Cursor = Cursors.Hand  
End Sub
```

```
Private Sub btnSelectEngland_MouseHover(sender As Object, e As EventArgs) Handles btnSelectEngland.MouseHover  
    'Provide the user with confirmation that they are selecting the civilisation of their preference by updating pbCivDisplay:
```

```
    Me.pbCivdisplay.Image = FormImgRef.pbSelectEnglandIcon.Image  
End Sub
```

```
Private Sub btnSelectEngland_MouseLeave(sender As Object, e As EventArgs) Handles btnSelectEngland.MouseLeave  
    'Show the user that they are no longer in the button to choose a civilisation by clearing pbCivDisplay.
```

```
    Me.pbCivdisplay.Image = FormImgRef.pblImageTempSave.Image  
End Sub
```

```
'The code for the select Seljuk Turks interface and buttons.
```

```
Private Sub btnSelectTurk_Click(sender As Object, e As EventArgs)  
Handles btnSelectTurk.Click  
    'Provide the user with feedback that they have selected Seljuk Turks using pbCivDisplay:
```

```

Me.pbCivdisplay.Image = FormImgRef.pbSelectTurkIcon.Image
'Provide a backup of the image of pbCivDisplay for the mouse hover/
leave events:
FormImgRef.pblImageTempSave.Image =
FormImgRef.pbSelectTurkIcon.Image
'Update the civilisation variable to the value for Seljuk Turks (2):
FormInGame.intCivilisation = 2
>Show the user they can now start the game:
btnStartGame2.Cursor = Cursors.Hand
End Sub

Private Sub btnSelectTurk_MouseHover(sender As Object, e As EventArgs)
Handles btnSelectTurk.MouseHover
'Provide the user with confirmation that they are selecting the civilisation
of their preference by updating pbCivDisplay:
Me.pbCivdisplay.Image = FormImgRef.pbSelectTurkIcon.Image
End Sub

Private Sub btnSelectTurk_MouseLeave(sender As Object, e As EventArgs)
Handles btnSelectTurk.MouseLeave
>Show the user that they are no longer in the button to choose a
civilisation by clearing pbCivDisplay.
Me.pbCivdisplay.Image = FormImgRef.pblImageTempSave.Image
End Sub

'The code for the select Yuan interface and buttons.
Private Sub btnSelectChina_Click(sender As Object, e As EventArgs)
Handles btnSelectChina.Click
'Provide the user with feedback that they have selected Yuan using
pbCivDisplay:
Me.pbCivdisplay.Image = FormImgRef.pbSelectChinalcon.Image
'Provide a backup of the image of pbCivDisplay for the mouse hover/
leave events:
FormImgRef.pblImageTempSave.Image =
FormImgRef.pbSelectChinalcon.Image
'Update the civilisation variable to the value for Yuan (3):
FormInGame.intCivilisation = 3
>Show the user they can now start the game:
btnStartGame2.Cursor = Cursors.Hand
End Sub

```

```
Private Sub btnSelectChina_MouseHover(sender As Object, e As EventArgs) Handles btnSelectChina.MouseHover
```

'Provide the user with confirmation that they are selecting the civilisation of their preference by updating pbCivDisplay:

```
    Me.pbCivdisplay.Image = FormImgRef.pbSelectChinalcon.Image
```

```
End Sub
```

```
Private Sub btnSelectChina_MouseLeave(sender As Object, e As EventArgs) Handles btnSelectChina.MouseLeave
```

'Show the user that they are no longer in the button to choose a civilisation by clearing pbCivDisplay.

```
    Me.pbCivdisplay.Image = FormImgRef.pblImageTempSave.Image
```

```
End Sub
```

'Code for the return to main menu button.

```
Private Sub btnReturnCivMain_Click(sender As Object, e As EventArgs) Handles btnReturnCivMain.Click
```

'Close the form, load the main menu and generate a random image for the main menu:

```
    Me.Close()
```

```
    FormMainMenu.MainMenuImgGen()
```

```
    FormMainMenu.Show()
```

```
End Sub
```

'Code for the start game button.

```
Private Sub btnStartGame2_Click(sender As Object, e As EventArgs) Handles btnStartGame2.Click
```

'Check that the user has selected a civilisation:

```
If FormInGame.intCivilisation > 0 Then
```

'If the user has not selected a civilisation they cannot start the game, this can be checked by comparing the value of intCivilisation

'(which will have the values 1, 2 or 3 if a civilisation has been selected) to zero.

'Returns pblImageTempSave to empty state:

```
FormImgRef.pblImageTempSave.Image = Nothing
```

'Close the form and open town name form:

```
    Me.Close()
```

```
    FormMapSettings.Show()
```

```
End If
```

```
End Sub
```

```
End Class
```

Code for FormHelpMenu

```
Public Class FormHelpMenu

    Private Sub btnExitHelp_Click(sender As Object, e As EventArgs) Handles
    btnExitHelp.Click
        'Closes the help menu, no need to open another form as the various
        routes the help menu do not close any forms:
        Me.Close()
    End Sub

End Class
```

Code for FormImgRef

```
Public Class FormImgRef

End Class
```

Code for FormInGame

```
Imports System.Drawing

Public Class FormInGame
    'Create some global variables that must be accessible in multiple forms:
    Public intCivilisation As Integer
    Public strTownName As String
    Public strCurrentBuilding As String = "None"
    Public blnBuildingSelected As Boolean = False
    Public strCurrentQuadrant As String = "Top Left"
    Dim intDateDay As Integer = 1
    Dim strDateDay As String
    Dim strDateMonth As String = "January"
    Dim intDateYear As Integer = 1296
    Dim intMouseX, intPrevMouseX, intMouseY, intPrevMouseY As Integer
    Public intGoldCount, intWoodStockpile, intStoneStockpile, intIronStockpile,
    intBarleyStockpile, intWheatStockpile, intOatStockpile, intWoolStockpile,
    intBeefStockpile, intFishStockpile, intAleStockpile As Integer

    'Code to be executed upon loading to provide clear UI:
    Private Sub FormInGame_Load(sender As Object, e As EventArgs)
        Handles MyBase.Load
```

```
tmrGameTimer.Start()
```

'Position the in-game menu menu to a convienient location, assuming in-game form is not moved (start location is centre and is repositioned):

```
Me.Top -= 150
```

```
Me.Show()
```

'Choose the UI for each civilisation and making unique buildings available by using the intCivilisation variable in a select case statement

```
Select Case intCivilisation
```

```
Case 1
```

```
    Me.BackgroundImage = FormImgRef.pbEnglandUI.Image
```

```
    FormBuildingMenu.btnAddFletcher.Visible = True
```

```
Case 2
```

```
    Me.BackgroundImage = FormImgRef.pbTurkUI.Image
```

```
    FormBuildingMenu.btnAddSpiceMarket.Visible = True
```

```
Case 3
```

```
    Me.BackgroundImage = FormImgRef.pbChinaUI.Image
```

```
    FormBuildingMenu.btnAddTerracedFarm.Visible = True
```

```
End Select
```

'Display the town name in the label lblTownName using the value of strTownName and display the month in pbMonthDisplay

```
lblTownName.Text = strTownName
```

```
pbMonthDisplay.Image = FormImgRef.pbJanuary.Image
```

'Generate the map

```
ModuleMapGeneration.subGenerateMap()
```

```
End Sub
```

```
Private Sub btnInGameHelp_Click(sender As Object, e As EventArgs)  
Handles btnInGameHelp.Click
```

'Loads the help menu, but does not close the main menu as the help menu is also be accessible from the main menu:

```
    FormHelpMenu.Show()
```

```
End Sub
```

```
Private Sub btnStartMenu_Click(sender As Object, e As EventArgs)  
Handles btnStartMenu.Click
```

'Opens the start menu with options to exit the game and to save the game:

```
    FormStartMenu.Show()
```

```
End Sub
```

```

Private Sub btnBuildingMenu_Click(sender As Object, e As EventArgs)
Handles btnBuildingMenu.Click
    'Displays the building menu where construction options are available:
    FormBuildingMenu.Show()
End Sub

Private Sub btnTreasuryMenu_Click(sender As Object, e As EventArgs)
Handles btnTreasuryMenu.Click
    'Displays the treasury menu where information about resource levels and
    income are described:
    FormTreasuryMenu.Show()
End Sub

Private Sub btnManagementMenu_Click(sender As Object, e As EventArgs)
Handles btnManagementMenu.Click
    'Displays the management menu where options for managing the
    population and town are available.
    FormManagementMenu.Show()
End Sub

Private Sub FormInGame_Paint(sender As Object, e As PaintEventArgs)
Handles pbMapDisplay.Paint
    'Prints the top left quadrant of the map to the screen:
    subPrintMap(1, 27, 1, 15)
    'Prints the minimap to four interactive picture boxes:
    subPrintMinimap(1, 27, 1, 15, gfxMinimapTL)
    subPrintMinimap(28, 54, 1, 15, gfxMinimapTR)
    subPrintMinimap(1, 27, 16, 30, gfxMinimapBL)
    subPrintMinimap(28, 54, 16, 30, gfxMinimapBR)
End Sub

Private Sub pbMinimapTL_Click(sender As Object, e As EventArgs)
Handles pbMinimapTL.Click
    'Print the top left corner of the map to the screen:
    subPrintMap(1, 27, 1, 15)
    'Update the cursors of the mini-map buttons:
    pbMinimapTL.Cursor = Cursors.Default
    pbMinimapBL.Cursor = Cursors.PanSouth
    pbMinimapTR.Cursor = Cursors.PanEast
    pbMinimapBR.Cursor = Cursors.PanSE
    'Update the relevant variable:
    strCurrentQuadrant = "Top Left"
End Sub

```

```
Private Sub pbMinimapTR_Click(sender As Object, e As EventArgs)
Handles pbMinimapTR.Click
    'Print the top right corner of the map to the screen:
    subPrintMap(28, 54, 1, 15)
    'Update the cursors of the mini-map buttons:
    pbMinimapTR.Cursor = Cursors.Default
    pbMinimapBL.Cursor = Cursors.PanSW
    pbMinimapTL.Cursor = Cursors.PanWest
    pbMinimapBR.Cursor = Cursors.PanSouth
    'Update the relevant variable:
    strCurrentQuadrant = "Top Right"
End Sub
```

```
Private Sub pbMinimapBL_Click(sender As Object, e As EventArgs)
Handles pbMinimapBL.Click
    'Print the bottom left corner of the map to the screen:
    subPrintMap(1, 27, 16, 30)
    'Update the cursors of the mini-map buttons:
    pbMinimapBL.Cursor = Cursors.Default
    pbMinimapTL.Cursor = Cursors.PanNorth
    pbMinimapTR.Cursor = Cursors.PanNE
    pbMinimapBR.Cursor = Cursors.PanEast
    'Update the relevant variable:
    strCurrentQuadrant = "Bottom Left"
End Sub
```

```
Private Sub pbMinimapBR_Click(sender As Object, e As EventArgs)
Handles pbMinimapBR.Click
    'Print the bottom right corner of the map to the screen:
    ModulePrintMap.subPrintMap(28, 54, 16, 30)
    'Update the cursors of the mini-map buttons:
    pbMinimapBR.Cursor = Cursors.Default
    pbMinimapBL.Cursor = Cursors.PanWest
    pbMinimapTR.Cursor = Cursors.PanNorth
    pbMinimapTL.Cursor = Cursors.PanNW
    'Update the relevant variable:
    strCurrentQuadrant = "Bottom Right"
End Sub
```

```
Private Sub tmrGameTimer_Tick(sender As Object, e As EventArgs)
Handles tmrGameTimer.Tick
    'Update the game date:
```

```

subUpdateGameDate()
'Update the displays of the resources:
lblWealth.Text = intGoldCount
FormTreasuryMenu.lblTreasuryGold.Text = intGoldCount
FormTreasuryMenu.lblWoodStockpile.Text = intWoodStockpile
FormTreasuryMenu.lblStoneStockpile.Text = intStoneStockpile
FormTreasuryMenu.lblIronStockpile.Text = intIronStockpile
FormTreasuryMenu.lblBarleyStockpile.Text = intBarleyStockpile
FormTreasuryMenu.lblWheatStockpile.Text = intWheatStockpile
FormTreasuryMenu.lblOatStockpile.Text = intOatStockpile
FormTreasuryMenu.lblAleStockpile.Text = intAleStockpile
FormTreasuryMenu.lblFishStockpile.Text = intFishStockpile
FormTreasuryMenu.lblBeefStockpile.Text = intBeefStockpile
FormTreasuryMenu.lblWoolStockpile.Text = intWoolStockpile
End Sub

```

```

Sub subUpdateGameDate()
'Increment the day:
intDateDay = intDateDay + 1
'Check that the date has reached the end of February (on a normal year)
If intDateDay = 29 And strDateMonth = "February" And intDateYear Mod
4 <> 0 Then
    'Reset the day count and increment the month count
    intDateDay = 1
    strDateMonth = "March"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbMarch.Image
End If
'Check that the date has reached the end of February (on a leap year)
If intDateDay = 30 And strDateMonth = "February" Then
    intDateDay = 1
    strDateMonth = "March"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbMarch.Image
End If
'Check that the date has reached the end of a short month (30 days)
If intDateDay = 31 And (strDateMonth = "April" Or strDateMonth = "June"
Or strDateMonth = "September" Or strDateMonth = "November") Then
    'Reset the day count and increment the month count according to the
    current month:
    intDateDay = 1
    Select Case strDateMonth
        Case "April"

```

```

strDateMonth = "May"
'Update the month display:
pbMonthDisplay.Image = FormImgRef.pbMay.Image
Case "June"
    strDateMonth = "July"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbJuly.Image
Case "September"
    strDateMonth = "October"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbOctober.Image
Case "November"
    strDateMonth = "December"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbDecember.Image
End Select
End If
'Check that the date has reached the end of a long month (31 days)
If intDateDay = 32 Then
    'Reset the day count and increment the month count according to the
    current month:
    intDateDay = 1
    Select Case strDateMonth
        Case "January"
            strDateMonth = "February"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbFebruary.Image
        Case "March"
            strDateMonth = "April"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbApril.Image
        Case "May"
            strDateMonth = "June"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbJune.Image
        Case "July"
            strDateMonth = "August"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbAugust.Image
        Case "August"
            strDateMonth = "September"
            'Update the month display:
            pbMonthDisplay.Image = FormImgRef.pbSeptember.Image
    End Select
End If

```

```

Case "October"
    strDateMonth = "November"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbNovember.Image
Case "December"
    strDateMonth = "January"
    'Update the month display:
    pbMonthDisplay.Image = FormImgRef.pbJanuary.Image
    intDateYear = intDateYear + 1
End Select
End If
'Create the final string for the month display:
Select Case intDateDay
    Case 1
        strDateDay = (intDateDay & "st")
    Case 21
        strDateDay = (intDateDay & "st")
    Case 31
        strDateDay = (intDateDay & "st")
    Case 2
        strDateDay = (intDateDay & "nd")
    Case 22
        strDateDay = (intDateDay & "nd")
    Case 3
        strDateDay = (intDateDay & "rd")
    Case 23
        strDateDay = (intDateDay & "rd")
    Case Else
        strDateDay = (intDateDay & "th")
End Select
lblDate.Text = (strDateDay & " " & strDateMonth & " " & intDateYear)
End Sub

```

```

Private Sub pbMapDisplay_MouseMove(sender As Object, e As
MouseEventArgs) Handles pbMapDisplay.MouseMove
    'This subroutine checks that when the mouse cursor is moved to a
    different tile the previous tile's graphic is replaced to facilitate the building
    placement guide
    Dim intXLower As Integer
    Dim intYLower As Integer
    'Stores the location of the mouse mouse position in terms of tile co-
    ordinates
    intMouseX = Math.Floor(e.X / 40)

```

```

intMouseY = Math.Floor(e.Y / 40)
'Checks whether the current tile is different to the last tile:
If intMouseY <> intPrevMouseY Or intMouseX <> intPrevMouseX Then
    Select Case strCurrentQuadrant
        'Updates the values of intXLower and intYLower to ensure the
        solution prints the tiles of the correct quadrant
        Case "Top Left"
            intXLower = 1
            intYLower = 1
        Case "Top Right"
            intXLower = 28
            intYLower = 1
        Case "Bottom Left"
            intXLower = 1
            intYLower = 16
        Case "Bottom Right"
            intXLower = 28
            intYLower = 16
    End Select
    'Prints the map tiles in a 2x3 region around the mouse cursor (the
    largest size the building placement guide can be)
    subPrintTile(intPrevMouseX, intPrevMouseY, intXLower, intYLower)
    subPrintTile(intPrevMouseX, intPrevMouseY - 1, intXLower,
    intYLower)
    subPrintTile(intPrevMouseX + 1, intPrevMouseY, intXLower,
    intYLower)
    subPrintTile(intPrevMouseX + 1, intPrevMouseY - 1, intXLower,
    intYLower)
    gfxGameMap.FillRectangle(Brushes.Transparent, intMouseX * 40,
    intMouseY * 40, 120, 120)
    If intPrevMouseY > 0 Then
        'Prevents the indices of the array being exceeded
        subPrintTile(intPrevMouseX, intPrevMouseY - 2, intXLower,
        intYLower)
        subPrintTile(intPrevMouseX + 1, intPrevMouseY - 2, intXLower,
        intYLower)
    End If
End If
'Updates the previous location of the mouse cursor
intPrevMouseX = intMouseX
intPrevMouseY = intMouseY

```

```

'Prints the building placement guide around the mouse cursor of the
appropriate size according to the building selected
If blnBuildingSelected = True Then
    Select Case strCurrentBuilding
        Case "Cruck Hut"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Farm"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 79, 79)
        Case "Lumberjack Hut"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Water Mill"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 79, 39)
        Case "Cottage"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Brewery"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Quarry"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
(intMouseY - 2) * 40, 79, 119)
        Case "Reeve House"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40, 39, 39)
        Case "Fishery"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 39, 79)
        Case "Mine"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 39, 79)
        Case "Pasture"
            gfxGameMap.DrawRectangle(Pens.Maroon, intMouseX * 40,
intMouseY * 40 - 40, 79, 79)
    End Select
End If
End Sub

```

```

Private Sub pbMapDisplay_MouseClick(sender As Object, e As
MouseEventArgs) Handles pbMapDisplay.MouseClick

```

'This subroutine checks whether a building is selected when the user clicks and places the building if possible

```
Dim intXLower As Integer  
Dim intYLower As Integer
```

```
Select Case strCurrentQuadrant
```

'Updates the values of intXLower and intYLower to ensure the solution prints the tiles of the correct quadrant

```
Case "Top Left"
```

```
    intXLower = 1  
    intYLower = 1
```

```
Case "Top Right"
```

```
    intXLower = 28  
    intYLower = 1
```

```
Case "Bottom Left"
```

```
    intXLower = 1  
    intYLower = 16
```

```
Case "Bottom Right"
```

```
    intXLower = 28  
    intYLower = 16
```

```
End Select
```

'Checks that a building is selected to prevent this code being run unnecessarily:

```
If blnBuildingSelected = True And arrMapBlueprint(intMouseY +  
intYLower, intMouseX + intXLower).ptyBuildingType = "Empty" Then
```

'Checks which building is selected:

```
Select Case strCurrentBuilding
```

```
Case "Cruck Hut"
```

'Checks the selected location is valid:

```
If arrMapBlueprint(intMouseY + intYLower, intMouseX +  
intXLower).ptyTileType = "Plains" Then
```

'Updates the building property of the tile/region:

```
arrMapBlueprint(intMouseY + intYLower, intMouseX +  
intXLower).ptyBuildingType = "Cruck Hut"
```

'Print the building:

```
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
```

'Remove the resources from the user:

```
intWoodStockpile = intWoodStockpile - 5
```

```
intGoldCount = intGoldCount - 10
```

'Deselect the building:

```
blnBuildingSelected = False
```

```
End If
```

```

Case "Farm"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower + 1).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "FarmBL"
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "FarmBR"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "FarmTL"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower + 1).ptyBuildingType = "FarmTR"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY - 1, intXLower,
intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 15
        intGoldCount = intGoldCount - 100
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Fishery"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Sea" And arrMapBlueprint(intMouseY + intYLower -
1, intMouseX + intXLower).ptyTileType = "Coast" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:

```

```

        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "FisheryB"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "FisheryT"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 20
        intGoldCount = intGoldCount - 150
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Water Mill"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower, intMouseX + intXLower + 1).ptyTileType = "River" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Water MillL"
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "Water MillR"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
        'Checks the other possible region of this building:
        ElseIf arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "River" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then
            'Updates the building property of the tile/region:
            arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Water MillL"
            arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "Water MillR"
            'Print the building:
            subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
            subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
            'Remove the resources from the user:

```

```

intWoodStockpile = intWoodStockpile - 50
intGoldCount = intGoldCount - 150
'Deselect the building:
blnBuildingSelected = False
End If
Case "Quarry"
'Checks the selected location is valid:
If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 2, intMouseX +
intXLower).ptyTileType = "Mountain" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower - 2, intMouseX +
intXLower).ptyBuildingType = "Empty" Then
'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "QuarryM"
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "QuarryBL"
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "QuarryBR"
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "QuarryTL"
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower + 1).ptyBuildingType = "QuarryTR"
'Print the building:
subPrintTile(intMouseX, intMouseY - 2, intXLower, intYLower)
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
subPrintTile(intMouseX + 1, intMouseY - 1, intXLower,
intYLower)
subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 100
intGoldCount = intGoldCount - 500
'Deselect the building:

```

```

        blnBuildingSelected = False
    End If
    Case "Mine"
        'Checks the selected location is valid:
        If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Mountain" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" Then
            'Updates the building property of the tile/region:
            arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "MineB"
            arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "MineT"
            'Print the building:
            subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
            subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
            'Remove the resources from the user:
            intWoodStockpile = intWoodStockpile - 200
            intStoneStockpile = intStoneStockpile - 100
            intGoldCount = intGoldCount - 750
            'Deselect the building:
            blnBuildingSelected = False
        End If
    Case "Pasture"
        'Checks the selected location is valid:
        If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX + intXLower +
1).ptyTileType = "Plains" And arrMapBlueprint(intMouseY + intYLower,
intMouseX + intXLower + 1).ptyTileType = "Plains" And
arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "Empty" And arrMapBlueprint(intMouseY +
intYLower - 1, intMouseX + intXLower + 1).ptyBuildingType = "Empty" And
arrMapBlueprint(intMouseY + intYLower, intMouseX + intXLower +
1).ptyBuildingType = "Empty" Then
            'Updates the building property of the tile/region:
            arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "PastureBL"
            arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower + 1).ptyBuildingType = "PastureBR"

```

```

        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower).ptyBuildingType = "PastureTL"
        arrMapBlueprint(intMouseY + intYLower - 1, intMouseX +
intXLower + 1).ptyBuildingType = "PastureTR"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY, intXLower, intYLower)
        subPrintTile(intMouseX + 1, intMouseY - 1, intXLower,
intYLower)
        subPrintTile(intMouseX, intMouseY - 1, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 25
        intGoldCount = intGoldCount - 250
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Lumberjack Hut"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Lumberjack Hut"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 10
        intGoldCount = intGoldCount - 50
        'Deselect the building:
        blnBuildingSelected = False
    End If
Case "Cottage"
    'Checks the selected location is valid:
    If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
        'Updates the building property of the tile/region:
        arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Cottage"
        'Print the building:
        subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
        'Remove the resources from the user:
        intWoodStockpile = intWoodStockpile - 2
        intStoneStockpile = intStoneStockpile - 3

```

```

intGoldCount = intGoldCount - 200
'Deselect the building:
bInBuildingSelected = False
End If
Case "Reeve House"
'Checks the selected location is valid:
If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Reeve House"
'Print the building:
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 200
intStoneStockpile = intStoneStockpile - 20
intGoldCount = intGoldCount - 750
'Deselect the building:
bInBuildingSelected = False
End If
Case "Brewery"
'Checks the selected location is valid:
If arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyTileType = "Plains" Then
'Updates the building property of the tile/region:
arrMapBlueprint(intMouseY + intYLower, intMouseX +
intXLower).ptyBuildingType = "Brewery"
'Print the building:
subPrintTile(intMouseX, intMouseY, intXLower, intYLower)
'Remove the resources from the user:
intWoodStockpile = intWoodStockpile - 20
intGoldCount = intGoldCount - 250
'Deselect the building:
bInBuildingSelected = False
End If
End Select
End If
End Sub
End Class

```

Code for FormMainMenu

```
Public Class FormMainMenu
```

```
    Private Sub FormMainMenu_Load(sender As Object, e As EventArgs)  
        Handles MyBase.Load
```

```
        'Subroutine to choose a background image
```

```
        MainMenuImgGen()
```

```
    End Sub
```

```
    Private Sub btnStartGame1_Click(sender As Object, e As EventArgs)  
        Handles btnStartGame1.Click
```

```
        'Closes the main menu and loads the civilisation menu:
```

```
        Me.Hide()
```

```
        FormCivilisationMenu.Show()
```

```
    End Sub
```

```
    Private Sub btnManageSaves_Click(sender As Object, e As EventArgs)  
        Handles btnManageSaves.Click
```

```
        'Closes the main menu and loads the manage saved games menu:
```

```
        Me.Hide()
```

```
        FormSavesMenu.Show()
```

```
    End Sub
```

```
    Private Sub btnHelp_Click(sender As Object, e As EventArgs)  
        Handles btnMainMenuHelp.Click
```

```
        'Loads the help menu, but does not close the main menu as the  
        help menu is also be accessible from in game:
```

```
        FormHelpMenu.Show()
```

```
    End Sub
```

```
    Private Sub btnExitGame_Click(sender As Object, e As EventArgs)  
        Handles btnExitGame.Click
```

```
        'Stops the game by closing the main menu.
```

```
        Me.Close()
```

```
    End Sub
```

```
Public Sub MainMenuImgGen()
```

```
    Dim RandomNumber As Integer
```

```

'Generate a random number between 1 and 3:
Randomize()
RandomNumber = CInt(Int((3 * Rnd()) + 1))
'Use the random number in a Select Case to choose a background
image:
Select Case RandomNumber
    Case 1
        Me.BackgroundImage = FormImgRef.pbMainMenu1.Image
    Case 2
        Me.BackgroundImage = FormImgRef.pbMainMenu2.Image
    Case 3
        Me.BackgroundImage = FormImgRef.pbMainMenu3.Image
End Select
End Sub

End Class

```

Code for FormManagementMenu

```

Public Class FormManagementMenu

    Private Sub FormManagementMenu_Load(sender As Object, e As
EventArgs) Handles MyBase.Load
        'Position the management menu to a convenient location, assuming in-
        game form is not moved (start location is centre and is repositioned):
        Me.Top += 403 '411
    End Sub

    Private Sub btnExitManagementMenu_Click(sender As Object, e As
EventArgs) Handles btnExitManagementMenu.Click
        'Hides the management menu:
        Me.Close()
    End Sub

End Class

```

Code for FormMapSettings

```

Public Class FormMapSettings

    Private Sub btnReturnMapCiv_Click(sender As Object, e As EventArgs)
Handles btnReturnMapCiv.Click

```

```

'Closes the map settings menu and opens the select civilisation menu:
Me.Close()
FormCivilisationMenu.Show()
End Sub

Private Sub btnMapSettingsNext_Click(sender As Object, e As EventArgs)
Handles btnMapSettingsNext.Click
'Closes the map settings menu and opens the town name menu:
Me.Close()
FormTownName.Show()
End Sub

Private Sub FormMapSettings_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
'Assigns default map settings:
bInSeaLake = True
intLakeWidth = 15
intLakeHeight = 4
intRiverBendiness = 3
intMountainFrequency = 0
intHillFrequency = 10
intFloraProbability = 10
intForestProbability = 5
End Sub

Private Sub trbWaterFeature_Scroll(sender As Object, e As EventArgs)
Handles trbWaterFeature.Scroll
'Update the value of the map setting variable as track bar moves and
update the map setting feedback display:
If trbWaterFeature.Value = 0 Then
    bInSeaLake = True
    pbWaterFeature.Image = FormImgRef.pbSeaOption.Image
Else
    bInSeaLake = False
    pbWaterFeature.Image = FormImgRef.pbLakeOption.Image
End If
End Sub

Private Sub trbLakeWidth_Scroll(sender As Object, e As EventArgs)
Handles trbLakeWidth.Scroll
'Update the value of the map setting variable as track bar moves and
update the map setting feedback display:
Select Case trbLakeWidth.Value

```

```
Case 0
    intLakeWidth = 15
    pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image
Case 1
    intLakeWidth = 16
    pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image
Case 2
    intLakeWidth = 17
    pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image
Case 3
    intLakeWidth = 18
    pbLakeWidth.Image = FormImgRef.pbLakeSmall.Image
Case 4
    intLakeWidth = 19
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 5
    intLakeWidth = 20
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 6
    intLakeWidth = 21
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 7
    intLakeWidth = 22
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 8
    intLakeWidth = 23
    pbLakeWidth.Image = FormImgRef.pbLakeMedium.Image
Case 9
    intLakeWidth = 24
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
Case 10
    intLakeWidth = 25
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
Case 11
    intLakeWidth = 26
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
Case 12
    intLakeWidth = 27
    pbLakeWidth.Image = FormImgRef.pbLakeLarge.Image
End Select
End Sub
```

```

Private Sub trbLakeHeight_Scroll(sender As Object, e As EventArgs)
Handles trbLakeHeight.Scroll
    'Update the value of the map setting variable as track bar moves and
    update the map setting feedback display:
    Select Case trbLakeHeight.Value
        Case 0
            intLakeHeight = 4
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 1
            intLakeHeight = 5
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 2
            intLakeHeight = 6
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 3
            intLakeHeight = 7
            pbLakeHeight.Image = FormImgRef.pbLakeShort.Image
        Case 4
            intLakeHeight = 8
            pbLakeHeight.Image = FormImgRef.pbLakeTall.Image
        Case 5
            intLakeHeight = 9
            pbLakeHeight.Image = FormImgRef.pbLakeTall.Image
        Case 6
            intLakeHeight = 10
            pbLakeHeight.Image = FormImgRef.pbLakeTall.Image
    End Select
End Sub

```

```

Private Sub trbRiverBendiness_Scroll(sender As Object, e As EventArgs)
Handles trbRiverBendiness.Scroll
    'Update the value of the map setting variable as track bar moves and
    update the map setting feedback display:
    Select Case trbRiverBendiness.Value
        Case 0
            intRiverBendiness = 3
            pbRiverBendiness.Image = FormImgRef.pbRiverVBendy.Image
        Case 1
            intRiverBendiness = 4
            pbRiverBendiness.Image = FormImgRef.pbRiverVBendy.Image
        Case 2
            intRiverBendiness = 5
            pbRiverBendiness.Image = FormImgRef.pbRiverLBendy.Image
    End Select
End Sub

```

```

Case 3
    intRiverBendiness = 6
    pbRiverBendiness.Image = FormImgRef.pbRiverLBendy.Image
Case 4
    intRiverBendiness = 7
    pbRiverBendiness.Image = FormImgRef.pbRiverStraight.Image
Case 5
    intRiverBendiness = 8
    pbRiverBendiness.Image = FormImgRef.pbRiverStraight.Image
End Select
End Sub

```

```

Private Sub trbMountainFrequency_Scroll(sender As Object, e As EventArgs) Handles trbMountainFrequency.Scroll
    'Update the value of the map setting variable as track bar moves and
    update the map setting feedback display:
    intMountainFrequency = trbMountainFrequency.Value
    If intMountainFrequency < 3 Then
        pbMountainFrequency.Image = FormImgRef.pbLMountainous.Image
    Else
        pbMountainFrequency.Image = FormImgRef.pbVMountainous.Image
    End If
End Sub

```

```

Private Sub trbHillFrequency_Scroll(sender As Object, e As EventArgs) Handles trbHillFrequency.Scroll
    'Update the value of the map setting variable as track bar moves and
    update the map setting feedback display:
    'Manipulate the value of intHillFrequency to facilitate the map setting in
    the context of the code of ModuleMapGeneration:
    intHillFrequency = trbHillFrequency.Value * 2 + 8
    If trbHillFrequency.Value > 6 Then
        pbHillFrequency.Image = FormImgRef.pbHilly.Image
    Else
        pbHillFrequency.Image = FormImgRef.pbFlat.Image
    End If
End Sub

```

```

Private Sub trbFloraFrequency_Scroll(sender As Object, e As EventArgs) Handles trbFloraFrequency.Scroll
    'Update the value of the map setting variable as track bar moves and
    update the map setting feedback display:

```

'Manipulate the value of intFloraProbability to facilitate the map setting in the context of the code of ModuleMapGeneration:

```
intFloraProbability = 11 - trbFloraFrequency.Value  
If intFloraProbability < 4 Then  
    pbFloraFrequency.Image = FormImgRef.pbFloraHigh.Image  
ElseIf intFloraProbability > 3 And intFloraProbability < 8 Then  
    pbFloraFrequency.Image = FormImgRef.pbFloraMedium.Image  
Else  
    pbFloraFrequency.Image = FormImgRef.pbFloraLow.Image  
End If  
End Sub
```

Private Sub trbForestSize_Scroll(sender As Object, e As EventArgs)
Handles trbForestSize.Scroll

'Update the value of the map setting variable as track bar moves and update the map setting feedback display:

'Manipulate the value of intForestProbability to facilitate the map setting in the context of the code of ModuleMapGeneration:

```
intForestProbability = 6 - trbForestSize.Value  
If trbForestSize.Value = 1 Then  
    pbForestSize.Image = FormImgRef.pbForestSmall.Image  
ElseIf trbForestSize.Value = 2 Then  
    pbForestSize.Image = FormImgRef.pbForestMedium.Image  
Else  
    pbForestSize.Image = FormImgRef.pbForestLarge.Image  
End If  
End Sub  
End Class
```

Code for FormSavesMenu

Public Class FormSavesMenu

Private Sub btnReturnSavesMain_Click(sender As Object, e As EventArgs)
Handles btnReturnSavesMain.Click

'Close the form, load the main menu and generate a random image for the main menu:

```
Me.Close()  
FormMainMenu.MainMenuImgGen()  
FormMainMenu.Show()  
End Sub
```

End Class

Code for FormStartMenu

Public Class FormStartMenu

```
Private Sub FormStartMenu_Load(sender As Object, e As EventArgs)  
Handles MyBase.Load
```

'Position the start menu to a convienient location, assuming in-game form
is not moved (start location is centre and is repositioned):

```
    Me.Left += 480
```

```
    Me.Top += 82
```

```
End Sub
```

```
Private Sub btnExitStartMenu_Click(sender As Object, e As EventArgs)  
Handles btnExitStartMenu.Click
```

'Hides the start menu:

```
    Me.Close()
```

```
End Sub
```

```
Private Sub btnExitInGame_Click(sender As Object, e As EventArgs)  
Handles btnExitInGame.Click
```

'Close all the menus relevant to the in-game form and load the main
menu:

```
    Me.Close()
```

```
    FormInGame.Close()
```

```
    FormBuildingMenu.Close()
```

```
    FormManagementMenu.Close()
```

```
    FormTreasuryMenu.Close()
```

```
    FormMainMenu.Show()
```

```
End Sub
```

```
End Class
```

Code for FormTownName

Public Class FormTownName

```
Private Sub FormTownName_Load(sender As Object, e As EventArgs)  
Handles MyBase.Load
```

'Picks the appropriate form user interface for the individual civilisations

```
Select Case FormInGame.intCivilisation
```

```
Case 1 'England
```

'Uses a relevant background image for the form:

```
    Me.BackgroundImage = FormImgRef.pbNameMenu1.Image
```

```

'Shows an appropriate UI for the town name entry:
Me.pbCivdisplay.Image = FormImgRef.pbTownEnglandIcon.Image
'Displays a brief appropriate for the civilisation:
Me.pbCivBrief.Image = FormImgRef.pbCivBrief1.Image
Case 2 'Turks
'Uses a relevant background image for the form:
Me.BackgroundImage = FormImgRef.pbNameMenu2.Image
'Shows an appropriate UI for the town name entry:
Me.pbCivdisplay.Image = FormImgRef.pbTownTurkIcon.Image
'Displays a brief appropriate for the civilisation:
Me.pbCivBrief.Image = FormImgRef.pbCivBrief2.Image
Case 3 'China
'Uses a relevant background image for the form:
Me.BackgroundImage = FormImgRef.pbNameMenu3.Image
'Shows an appropriate UI for the town name entry:
Me.pbCivdisplay.Image = FormImgRef.pbTownChinalcon.Image
'Displays a brief appropriate for the civilisation:
Me.pbCivBrief.Image = FormImgRef.pbCivBrief3.Image
End Select
End Sub

```

```

Private Sub btnReturnCivMain_Click(sender As Object, e As EventArgs)
Handles btnReturnCivMain.Click
    'Button is to be pressed if the user has reconsidered the chosen
    civilisation, thus the civilisation menu should return to its original state:
    'pbCivDisplay and pbImageTempSave returned to empty state:
    FormCivilisationMenu.pbCivdisplay.Image = Nothing
    FormImgRef.pbImageTempSave.Image = Nothing
    'Initialises the intCivilisation variable:
    FormInGame.intCivilisation = 0
    'Reminds the user to select a civilisation to continue:
    FormCivilisationMenu.btnStartGame2.Cursor = Cursors.No
    'Closes the form and opens the Civilisation menu:
    Me.Close()
    FormMapSettings.Show()
End Sub

```

```

Private Sub btnSettleTown_Click(sender As Object, e As EventArgs)
Handles btnSettleTown.Click
    'Update the strTownName variable to the value of the entered text of
    tbTownName:
    FormInGame.strTownName = tbTownName.Text

```

```
Me.Close()
'Closes the form and opens the in-game menu:
FormInGame.Show()
'Sets the resource stockpiles to the default levels:
FormInGame.intGoldCount = 1000
FormInGame.intWoodStockpile = 1000
FormInGame.intStoneStockpile = 1000
FormInGame.intIronStockpile = 1000
FormInGame.intBarleyStockpile = 1000
FormInGame.intWheatStockpile = 1000
FormInGame.intOatStockpile = 1000
FormInGame.intWoolStockpile = 1000
FormInGame.intBeefStockpile = 1000
FormInGame.intFishStockpile = 1000
FormInGame.intAleStockpile = 1000
End Sub
```

```
End Class
```

Code for FormTreasuryMenu

```
Public Class FormTreasuryMenu
```

```
Private Sub FormTreasuryMenu_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
```

```
'Position the treasury menu to a convenient location, assuming in-game
form is not moved (start location is centre and is repositioned):
```

```
    Me.Top -= 150
```

```
    Me.Left += 715
```

```
End Sub
```

```
Private Sub btnExitTreasuryMenu_Click(sender As Object, e As EventArgs)
Handles btnExitTreasuryMenu.Click
```

```
'Hide the treasury menu:
```

```
    Me.Close()
```

```
End Sub
```

```
End Class
```

Code for ModuleMapGeneration

Module ModuleMapGeneration

```
Public arrMapBlueprint(31, 55) As MapTile 'Map size 30, 54 and a border to
prevent exceeding the array
Public blnSeaLake As Boolean 'If true a sea will generate, if false a lake will
generate
Public intMountainFrequency, intForestProbability, intHillFrequency,
intFloraProbability, intLakeWidth, intLakeHeight, intRiverBendiness As Integer

Sub subGenerateMap()
    'Some general purpose variables to save space and simplify the solution
    Dim intArrYCoord, intArrXCoord, intCount, intTemp, intGraphX,
    intGraphY, intProbability As Integer

    'River related variables
    Dim intRiverX1, intRiverX2, intRiverY As Integer

    'Sea/Lake related variables
    Dim intLakeRight, intLakeLeft, intLakeLeftMax, intLakeRightMax,
    intLakeRightWidth, intLakeLeftWidth, intSeaHeight As Integer
    Dim dblLakeShape, dblSeaShape As Double

    'Fill the map with empty tiles and add a border
    For intY = 0 To 31
        For intX = 0 To 55
            arrMapBlueprint(intY, intX) = New MapTile
            If intY = 0 Or intY = 31 Or intX = 0 Or intX = 55 Then
                arrMapBlueprint(intY, intX).ptyTileType = "Border"
            Else
                arrMapBlueprint(intY, intX).ptyTileType = "Empty"
                arrMapBlueprint(intY, intX).ptyBuildingType = "Empty"
            End If
        Next
    Next

    '=====
    =
```

'First generate the river source location, should be randomly placed on the array somewhere in the co-ordinates: 1<=y<=2, 8<=x<=47

```
intArrYCoord = FuncRandomNumberGen(2, 3)
intArrXCoord = FuncRandomNumberGen(8, 47)
arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
```

"RiverSource"

```
'=====
=
```

'Next generate a stream leading from the source down to the start of the river (should extend 2-4 tiles and be 1 tile wide)

```
intCount = intArrYCoord + 1
For intArrYCoord = intCount To FuncRandomNumberGen(intArrYCoord +
2, intArrYCoord + 4)
    arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType = "Stream"
    Next
```

```
'=====
=
```

'Next generate the river beneath the stream leading down to the bottom of the screen (river should be 2 tiles wide)

```
'Find starting location of river
intRiverX1 = intArrXCoord
intRiverY = intArrYCoord
```

'The river should bend. The bendiness of the river depends on how often the river is given an opportunity to bend. This trait will be referred to as the magnitude of river meandering

'Decide the magnitude of river meandering - will be saved in the variable intTemp (should be between 4 and 8)

'Decide which side of the stream to expand (left or right, can be determined with a random number used in a select case)

Select Case FuncRandomNumberGen(1, 2)

Case 1

```
intRiverX1 = intRiverX1 - 1
intRiverX2 = intRiverX1 + 1
```

Case 2

```
intRiverX2 = intRiverX1 + 1
```

End Select

```

intCount = intRiverY
While intRiverY < 31
    For intRiverY = intCount To intRiverY + intRiverBendiness
        arrMapBlueprint(intRiverY, intRiverX1).ptyTileType = "River"
        arrMapBlueprint(intRiverY, intRiverX2).ptyTileType = "River"
        'Ensure that intRiverY does not exceed bounds of arrMapBlueprint
        If intRiverY = 30 Then
            Exit While
        End If
    Next
    'Decide which way to meander
    Select Case intRiverX1
        Case 3
            intTemp = 2
        Case 51
            intTemp = 1
        Case Else
            intTemp = FuncRandomNumberGen(1, 2)
    End Select

    Select Case intTemp
        Case 1
            intRiverX1 = intRiverX1 - 1
            intRiverX2 = intRiverX2 - 1
        Case 2
            intRiverX1 = intRiverX1 + 1
            intRiverX2 = intRiverX2 + 1
    End Select
    intCount = intRiverY
End While

```

=====
=

'Now that the river has been generated, the sea must now be generated
(which will always appear on the bottom of the screen)

'First the dimensions of the sea must be decided: width followed by
height (how far inland it reaches)

'Width: the sea can either take up all the columns of the bottom few rows,
or several - to appear as a lake: the chances of the sea...

'...taking up the whole of the bottom of the screen should be lower than
the probability of it appearing as a lake

If blnSeaLake = False Then

'The lake outline will be drawn using an arc function to find the array co-ordinate values of the lake outline

'Find out how far the lake will stretch either side of the river, ensure the lake size will not be reduced by edge of form

```
If intRiverX1 <= 27 Then
    intLakeLeftMax = intLakeWidth - 2

    If intRiverX1 - intLakeWidth < 1 Then 'Checking left Bounds
        intLakeLeftMax = intRiverX1 - 1
    End If

    intLakeLeft = intRiverX1 - FuncRandomNumberGen(2,
intLakeLeftMax)
    intLakeRight = intRiverX2 + (intLakeWidth - (intRiverX1 -
intLakeLeft))
    intLakeLeftWidth = intRiverX1 - intLakeLeft
    intLakeRightWidth = intLakeRight - intRiverX2
Else
    intLakeRightMax = intLakeWidth - 2

    If intRiverX2 + intLakeWidth > 54 Then 'Checking right Bounds
        intLakeRightMax = 54 - intRiverX2
    End If

    intLakeRight = intRiverX2 + FuncRandomNumberGen(2,
intLakeRightMax)
    intLakeLeft = intRiverX1 - (intLakeWidth - (intLakeRight -
intRiverX2))
    intLakeLeftWidth = intRiverX1 - intLakeLeft
    intLakeRightWidth = intLakeRight - intRiverX2
End If
```

'The arc equation is as follows: $Y = (\text{intlakeHeight}^{(\text{dbllakeShape})} - ((X-\text{intlakeRight})/(\text{intRightWidth}/\text{intlakeHeight}))^{(\text{dbllakeShape})})^{(1/(\text{dbllakeShape}))}$

'The equation for the left hand side is identical, replace intlakeRight with intlakeLeft and intRightWidth with -intLeftWidth

'Where Y-intercept is intlakeHeight and X-intercept is intlakeHeight * intlakeRightWidth

'Where X = 0 is the river, and Y = 0 is bottom of the screen i.e X = intRiverX1/intRiverX2, Y = 43

'Create outline for lake left hand side of river:

'First generate a random number between 0.8 and 3.2 for the intLakeShape variable

'Because of how this function graphs, the left hand side of the graph only allows decimal values that are divisible by 4 when multiplied by 10 (i.e 1.0 is not allowed as $1.0 * 10 = 10$ and $10 / 3 \neq$ integer)

dblLakeShape = (4 * FuncRandomNumberGen(2, 8)) / 10

For intGraphX = -intLakeLeftWidth To 0

 intGraphY = CInt((intLakeHeight ^ dblLakeShape) - ((intGraphX / (-intLakeLeftWidth / intLakeHeight)) ^ dblLakeShape)) ^ (1 / dblLakeShape)

 intArrXCoord = intRiverX1 + intGraphX

 intArrYCoord = 30 - intGraphY

 arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType = "Sea"

Next

'Create Outline for lake right hand side of the river:

'First generate a random number between 0.8 and 3.2 for the intLakeShape variable

'Because of how this function graphs, the right hand side of the graph cannot allow decimal values that are odd when multiplied by 10 (i.e 0.9 is not allowed as $0.9 * 10 = 9$)

dblLakeShape = (2 * FuncRandomNumberGen(4, 16)) / 10

For intGraphX = 0 To intLakeRightWidth

 intGraphY = CInt((intLakeHeight ^ dblLakeShape) - ((intGraphX / (intLakeRightWidth / intLakeHeight)) ^ dblLakeShape)) ^ (1 / dblLakeShape)

 'Moving the co-ordinates into the array bounds at the correct locations

 intArrXCoord = intRiverX2 + intGraphX

 intArrYCoord = 30 - intGraphY

 arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType = "Sea"

Next

'Now fill in the lake by locating the top of the lake in each column and filling down from there

For intArrXCoord = intLakeLeft To intLakeRight

 For intArrYCoord = 30 - intLakeHeight To 30

 If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType = "Sea" Then

 intCount = intArrYCoord

 For intGraphY = intCount To 30

 arrMapBlueprint(intGraphY, intArrXCoord).ptyTileType =

"Sea"

Next

```

        End If
    Next
Next
'The lake is now generated
Else
    'The sea shape will be produced by graphing a Y = Sin(X) or Y =
Cos(X) function. The shape of the Y = Sin(X) or Y = Cos(X) functions can be
varied using a variable.
    'The equation of the sea coast will either be Y =
dblSeaShape*Math.Sin(X/dblSeaShape) or Y = dblSeaShape*Math.Cos(X/
dblSeaShape)
    'The shape of the sea should also have a chance to change, every
trig-wavelength there will be a 2/3 chance that dblSeaShape will change for
the next iteration
    'First generate intSeaHeight and dblSeaShape
    'inSeaHeight is where the middle of the Y = Sin(X) or Y = Cos(X)
functions will be centred upon:
intSeaHeight = FuncRandomNumberGen(2, 4)
'intSeaShape will vary between 0.6 and 1.5:
dblSeaShape = FuncRandomNumberGen(6, 15) / 10

'Decide which trig-function will begin
Select Case FuncRandomNumberGen(1, 2)
    Case 1 'Y = dblSeaShape*Math.Sin(X/(2*dblSeaShape))
        For intGraphX = 1 To 54
            intGraphY = CInt(dblSeaShape * Math.Sin(intGraphX / (2 *
dblSeaShape)))
            intArrXCoord = intGraphX
            intArrYCoord = intGraphY + (30 - intSeaHeight)
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Sea"
            If intGraphY = 1 Then
                Select Case intTemp = FuncRandomNumberGen(1, 3)
                    Case 1 'Do nothing
                    Case Else
                        dblSeaShape = FuncRandomNumberGen(6, 15) / 10
                End Select
            End If
        Next
    Case 2 'Y = dblSeaShape*Math.Cos(X/dblSeaShape)
        For intGraphX = 1 To 54
            intGraphY = CInt(dblSeaShape * Math.Cos(intGraphX / (2 *
dblSeaShape)))

```

```

intArrXCoord = intGraphX
intArrYCoord = intGraphY + (30 - intSeaHeight)
arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Sea"
If intGraphY = -1 Then
    Select Case intTemp = FuncRandomNumberGen(1, 3)
        Case 1 'Do nothing
        Case Else
            dblSeaShape = FuncRandomNumberGen(6, 15) / 10
        End Select
    End If
    Next
End Select

```

'Now fill in the sea by locating the top of the sea in each column and filling down from there

```

For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Sea" Then
            intCount = intArrYCoord
            For intGraphY = intCount To 30
                arrMapBlueprint(intGraphY, intArrXCoord).ptyTileType =
"Sea"
            Next
        End If
        Next
    Next
End If

```

'The sea is now generated.

'By this point either a sea or a lake has been generated.

```
'=====
=
```

'Next the sea/lake must be lined with coastline tiles, this will be implemented by first looping down the columns and adding a coast tile above the first sea tile...

```

For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30

```

```

    If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Sea" Then
        If arrMapBlueprint(intArrYCoord - 1, intArrXCoord).ptyTileType
= "River" Then
            Exit For
        End If
        arrMapBlueprint(intArrYCoord - 1, intArrXCoord).ptyTileType =
"Coast"
        Exit For
    End If
    Next
    Next
    '...and secondly by looping through the rows and adding a coast tile to
the edge sea tiles
    For intArrYCoord = 1 To 30
        For intArrXCoord = 1 To 54
            If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Sea" And arrMapBlueprint(intArrYCoord, intArrXCoord - 1).ptyTileType =
"Empty" Then
                arrMapBlueprint(intArrYCoord, intArrXCoord - 1).ptyTileType =
"Coast"
            End If
        Next
        For intArrXCoord = 1 To 54
            If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Sea" And arrMapBlueprint(intArrYCoord, intArrXCoord + 1).ptyTileType =
"Empty" Then
                arrMapBlueprint(intArrYCoord, intArrXCoord + 1).ptyTileType =
"Coast"
            End If
        Next
    Next

```

'=====

=

'Next the mountains will be generated. These will only ever appear along
the top of the map, and will never extent down further than 4 tiles from the top.
There will always...

'...be mountains around the river source tile, but they will also be
generated in clumps. The clump size can be large and small, and the size will
be determined by probability.

```

'First generate mountains around river source tile. Accomplished by
locating river tile and filling in mountain tiles around it in preset pattern
'Find river source tile
For intArrYCoord = 2 To 4
    For intArrXCoord = 8 To 48
        If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"RiverSource" Then
            Exit For
        End If
    Next
    If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"RiverSource" Then
        Exit For
    End If
    Next
    arrMapBlueprint(intArrYCoord + 1, intArrXCoord - 1).ptyTileType =
"Mountain"
    arrMapBlueprint(intArrYCoord, intArrXCoord - 1).ptyTileType =
"Mountain"
    arrMapBlueprint(intArrYCoord - 1, intArrXCoord - 1).ptyTileType =
"Mountain"
    arrMapBlueprint(intArrYCoord - 1, intArrXCoord).ptyTileType =
"Mountain"
    arrMapBlueprint(intArrYCoord - 1, intArrXCoord + 1).ptyTileType =
"Mountain"
    arrMapBlueprint(intArrYCoord, intArrXCoord + 1).ptyTileType =
"Mountain"
    arrMapBlueprint(intArrYCoord + 1, intArrXCoord + 1).ptyTileType =
"Mountain"
    'Now loop along the top of the array and place single mountain tiles
randomly to act as anchors for the mountain ranges. Probability of anchor
mountains being spawned can be decided...
    'by comparing the values of two numbers, the frequency of anchor
mountains being spawned can be varied by changing the value of
intMountainProbability
    For intArrXCoord = 1 To 54
        If FuncRandomNumberGen(1, 4) > 1 Then
            arrMapBlueprint(1, intArrXCoord).ptyTileType = "Mountain"
        End If
    Next
    'Select the mountain generation sequence that fits the user map setting
choice:
    Select Case intMountainFrequency

```

```

Case 0
    subGenerateMountains(3, 1)
    subGenerateMountains(4, 2)
    subGenerateMountains(5, 3)
    subGenerateMountains(6, 4)
Case 1
    subGenerateMountains(2, 1)
    subGenerateMountains(3, 2)
    subGenerateMountains(4, 3)
    subGenerateMountains(5, 4)
Case 2
    subGenerateMountains(2, 1)
    subGenerateMountains(2, 2)
    subGenerateMountains(3, 3)
    subGenerateMountains(4, 4)
Case 3
    subGenerateMountains(1, 1)
    subGenerateMountains(2, 2)
    subGenerateMountains(2, 3)
    subGenerateMountains(3, 4)
Case 4
    subGenerateMountains(1, 1)
    subGenerateMountains(1, 2)
    subGenerateMountains(2, 3)
    subGenerateMountains(2, 4)
End Select
'Mountains now generated

```

```

'=====
=
'Now to generate forests:
'All empty tiles in the top three rows should be converted to forest
For intArrYCoord = 1 To 3
    For intArrXCoord = 1 To 54
        If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Empty" Then
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Forest"
        End If
    Next
Next

```

'Now to add some detail to the top forest by converting some of the empty tiles below the 3rd row forest into more forest

```
For intArrXCoord = 1 To 54
    If arrMapBlueprint(3, intArrXCoord).ptyTileType = "Forest" Then
        Select Case FuncRandomNumberGen(1, 2)
            Case 1 'Add a forest tile
                arrMapBlueprint(4, intArrXCoord).ptyTileType = "Forest"
        End Select
    End If
Next
```

'The forests will be generated in a similar fashion to the mountains (creating anchors and generating around them)

'First generate forest anchors on the edge columns of the map on empty tiles

```
For intArrXCoord = 1 To 54
    For intArrYCoord = 4 To 30
        If FuncRandomNumberGen(1, 4) > 1 And
arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType = "Empty" Then
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Temporary"
    End If
```

```
'Skip middle columns
If intArrXCoord = 2 And intArrYCoord = 30 Then
    intArrXCoord = 53
End If
```

```
Next
```

```
Next
```

'Now generate forest anchors on the bottom 2 rows of the map on empty tiles (obviously won't be generated if a sea is generated)

```
For intArrYCoord = 29 To 30
    For intArrXCoord = 1 To 54
        If FuncRandomNumberGen(1, 3) > 1 And
arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType = "Empty" Then
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Temporary"
    End If
```

```
Next
```

```
Next
```

'Next allow the forests to spread from all current forest tiles into empty tiles in all directions

```
For intArrYCoord = 3 To 30
    For intArrXCoord = 1 To 54
```

```

If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Temporary" Then
    intProbability = FuncRandomNumberGen(1, intForestProbability)
    Select Case intProbability
        Case 1 'Spread on all 8 tiles around the anchor
            If arrMapBlueprint(intArrYCoord + 1, intArrXCoord -
1).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord + 1, intArrXCoord -
1).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord, intArrXCoord -
1).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord, intArrXCoord -
1).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord - 1, intArrXCoord -
1).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord - 1, intArrXCoord -
1).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord - 1,
intArrXCoord).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord - 1,
intArrXCoord).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord - 1, intArrXCoord +
1).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord - 1, intArrXCoord +
1).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord, intArrXCoord +
1).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord, intArrXCoord +
1).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord + 1, intArrXCoord +
1).ptyTileType = "Empty" Then
                arrMapBlueprint(intArrYCoord + 1, intArrXCoord +
1).ptyTileType = "Temporary"
            End If
            If arrMapBlueprint(intArrYCoord + 1,
intArrXCoord).ptyTileType = "Empty" Then

```

```

        arrMapBlueprint(intArrYCoord + 1,
intArrXCoord).ptyTileType = "Temporary"
    End If
Case 2
    'Spread on 4 adjacent tiles
    If arrMapBlueprint(intArrYCoord, intArrXCoord -
1).ptyTileType = "Empty" Then
        arrMapBlueprint(intArrYCoord, intArrXCoord -
1).ptyTileType = "Temporary"
    End If
    If arrMapBlueprint(intArrYCoord - 1,
intArrXCoord).ptyTileType = "Empty" Then
        arrMapBlueprint(intArrYCoord - 1,
intArrXCoord).ptyTileType = "Temporary"
    End If
    If arrMapBlueprint(intArrYCoord, intArrXCoord +
1).ptyTileType = "Empty" Then
        arrMapBlueprint(intArrYCoord, intArrXCoord +
1).ptyTileType = "Temporary"
    End If
    If arrMapBlueprint(intArrYCoord + 1,
intArrXCoord).ptyTileType = "Empty" Then
        arrMapBlueprint(intArrYCoord + 1,
intArrXCoord).ptyTileType = "Temporary"
    End If
End Select
End If
Next
Next
'Now change the placeholder forest tiles into true forest tiles
For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Temporary" Then
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Forest"
        End If
    Next
Next

'=====
=

```

```
'Now generate plains tiles by replacing all remaining empty tiles with  
plains tiles
```

```
For intArrXCoord = 1 To 54  
    For intArrYCoord = 1 To 30  
        If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =  
"Empty" Then  
            arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =  
"Plains"  
        End If  
    Next  
Next
```

```
'=====
```

```
'Penultimately generate hill tiles in 2x2 clumps  
'Loop through array locating plains tiles with space to generate a 2x2 hill  
While intHillFrequency > 0  
    For intArrXCoord = 1 To 53  
        For intArrYCoord = 1 To 30  
            If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =  
"Plains" And arrMapBlueprint(intArrYCoord + 1, intArrXCoord).ptyTileType =  
"Plains" And arrMapBlueprint(intArrYCoord, intArrXCoord + 1).ptyTileType =  
"Plains" And arrMapBlueprint(intArrYCoord + 1, intArrXCoord + 1).ptyTileType  
= "Plains" Then  
                Select Case FuncRandomNumberGen(1, 100)  
                    Case 1 'Generate a hill  
                        arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType  
= "Hill"  
                        arrMapBlueprint(intArrYCoord + 1,  
intArrXCoord).ptyTileType = "Hill"  
                        arrMapBlueprint(intArrYCoord, intArrXCoord +  
1).ptyTileType = "Hill"  
                        arrMapBlueprint(intArrYCoord + 1, intArrXCoord +  
1).ptyTileType = "Hill"  
                        intHillFrequency = intHillFrequency - 1  
                    If intHillFrequency = 0 Then  
                        Exit For  
                    End If  
                End Select  
            End If
```

```

        Next
        If intHillFrequency = 0 Then
            Exit For
        End If
    Next
End While

```

```
'=====
=
```

'Finally generate flora tiles which will randomly replace plains tiles to vary the graphics more

```

For intArrXCoord = 1 To 54
    For intArrYCoord = 1 To 30
        Select Case FuncRandomNumberGen(1, intFloraProbability)
            Case 1
                If arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
                    "Plains" Then
                        arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
                    "Flora"
                End If
            End Select
        Next
    Next
End Sub

```

```

Function FuncRandomNumberGen(ByVal intLowerBound As Integer, ByVal
intUpperBound As Integer)
    Dim intRandomNumber = 0
    Randomize()
    intRandomNumber = CInt(Math.Floor((intUpperBound - intLowerBound +
1) * Rnd())) + intLowerBound 'Upper and lower bounds set ranges (inclusively)
    Return intRandomNumber
End Function

```

```

Sub subGenerateMountains(intProbability, intArrYCoord)
    'intTemp used to store input of intProbability
    Dim intTemp As Integer
    'Check row above
    intTemp = intProbability
    For intArrXCoord = 1 To 54

```

```

        If arrMapBlueprint(intArrYCoord - 1, intArrXCoord).ptyTileType =
    "Mountain" And arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType <>
    "RiverSource" And arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType
    <> "Stream" Then
        Select Case FuncRandomNumberGen(1, intProbability)
            Case 1 'Generate a mountain
                arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
    "Mountain"
                intProbability = intProbability + 1
            Case Else 'Do nothing
                intProbability = intTemp
        End Select

    End If
    Next
    'First loop left to right...
    intProbability = intTemp
    For intArrXCoord = 1 To 54
        If arrMapBlueprint(intArrYCoord, intArrXCoord - 1).ptyTileType =
    "Mountain" And arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType <>
    "RiverSource" And arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType
    <> "Stream" Then
            Select Case FuncRandomNumberGen(1, intProbability)
                Case 1 'Generate a mountain
                    arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
    "Mountain"
                    intProbability = intProbability + 1
                Case Else 'Do nothing
                    intProbability = intTemp
            End Select

        End If
        Next
        '...second loop right to left:
        intProbability = intTemp
        For intArrXCoord = 54 To 1 Step -1
            If arrMapBlueprint(intArrYCoord, intArrXCoord + 1).ptyTileType =
    "Mountain" And arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType <>
    "RiverSource" And arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType
    <> "Stream" Then
                Select Case FuncRandomNumberGen(1, intProbability)
                    Case 1 'Generate a mountain

```

```

arrMapBlueprint(intArrYCoord, intArrXCoord).ptyTileType =
"Mountain"
    intProbability = intProbability + 1
    Case Else 'Do nothing
        intProbability = intTemp
    End Select
End If
Next
End Sub

End Module

```

Code for ModulePrintMap

```

Imports System.Drawing
Module ModulePrintMap

    Public gfxGameMap As System.Drawing.Graphics =
FormInGame.pbMapDisplay.CreateGraphics()
    Public rectGraphicsSource As Rectangle
    Public rectGraphicsDestination As Rectangle
    Public bmpImage As Bitmap
    Public gfxMinimapTL As System.Drawing.Graphics =
FormInGame.pbMinimapTL.CreateGraphics()
    Public gfxMinimapTR As System.Drawing.Graphics =
FormInGame.pbMinimapTR.CreateGraphics()
    Public gfxMinimapBL As System.Drawing.Graphics =
FormInGame.pbMinimapBL.CreateGraphics()
    Public gfxMinimapBR As System.Drawing.Graphics =
FormInGame.pbMinimapBR.CreateGraphics()

    'Subroutine that prints an entire map quadrant to the screen
    Sub subPrintMap(ByVal intXLower As Integer, ByVal intXUpper As Integer,
ByVal intYLower As Integer, ByVal intYUpper As Integer)
        'Variables to locate the destination rectangle
        Dim intYCoord, intXCoord As Integer
        For intY = intYLower To intYUpper
            For intX = intXLower To intXUpper
                'Translate the array index values to co-ordinates:
                intYCoord = (intY - intYLower) * 40
                intXCoord = (intX - intXLower) * 40
                'Define the source and destination rectangles:
                rectGraphicsDestination = New Rectangle(intXCoord, intYCoord,
40, 40)

```

```

rectGraphicsSource = New Rectangle(0, 0, 40, 40)
Select Case arrMapBlueprint(intY, intX).ptyTileType
    Case "Plains" 'Generate a plains graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbPlainsTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Flora" 'Generate a flora graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbFloraTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Hill" 'Generate a hill graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbHillTLTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Forest" 'Generate a forest graphic at the location
        bmpImage = New
Bitmap(FormImgRef.pbForestCentreTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "RiverSource" 'Generate a riverSource graphic at the
location
        bmpImage = New
Bitmap(FormImgRef.pbRiverSourceTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Stream" 'Generate a stream graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbStreamTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "River" 'Generate a river graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbRiverTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Coast" 'Generate a coast graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbCoastTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Sea" 'Generate a sea graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbSeaTile.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Mountain" 'Generate a mountain graphic at the location
        bmpImage = New Bitmap(FormImgRef.pbMountainTile.Image)

```

```

        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    End Select

    Select Case arrMapBlueprint(intY, intX).ptyBuildingType
        Case "Cruck Hut" 'Print a cruck hut
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingCruckHut.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

        Case "Lumberjack Hut" 'Print a lumberjack hut
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingLumberjackHut.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

        Case "Cottage" 'Print a cottage
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingCottage.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

        Case "Reeve House" 'Print a reeve house
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingReeveHouse.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

        Case "Brewery" 'Print a brewery
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingBrewery.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

        Case "FarmTL" 'Print the top left of a farm
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingFarmTL.Image)
            gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

        Case "FarmTR" 'Print the top right of a farm
            bmpImage = New
            Bitmap(FormImgRef.pbBuildingFarmTR.Image)

```

```

gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "FarmBL" 'Print the bottom left of a farm
bmplImage = New
Bitmap(FormImgRef.pbBuildingFarmBL.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "FarmBR" 'Print the bottom right of a farm
bmplImage = New
Bitmap(FormImgRef.pbBuildingFarmBR.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FisheryT" 'Print the top of a fishery
bmplImage = New
Bitmap(FormImgRef.pbBuildingFisheryT.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "FisheryB" 'Print the bottom of a fishery
bmplImage = New
Bitmap(FormImgRef.pbBuildingFisheryB.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Water MillL" 'Print the left of a water mill
If arrMapBlueprint(intY, intX).ptyTileType = "Plains" Then
bmplImage = New
Bitmap(FormImgRef.pbBuildingWaterMillLP.Image)
Else
bmplImage = New
Bitmap(FormImgRef.pbBuildingWaterMillLR.Image)
End If
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "Water MillR" 'Print the right of a water mill
If arrMapBlueprint(intY, intX).ptyTileType = "Plains" Then
bmplImage = New
Bitmap(FormImgRef.pbBuildingWaterMillRP.Image)
Else
bmplImage = New
Bitmap(FormImgRef.pbBuildingWaterMillRR.Image)
End If

```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "QuarryM" 'Print the mountain section of a quarry

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryM.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "QuarryTL" 'Print the top left of a quarry

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryTL.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "QuarryTR" 'Print the top right of a quarry

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryTR.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "QuarryBL" 'Print the bottom left of a quarry

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryBL.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "QuarryBR" 'Print the bottom right of a quarry

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingQuarryBR.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "MineT" 'Print the top of a mine

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingMineT.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "MineB" 'Print the bottom of a mine

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingMineB.Image)
```

```
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,  
rectGraphicsSource, GraphicsUnit.Pixel)
```

Case "PastureTL" 'Print the top left of a pasture

```
bmplImage = New
```

```
Bitmap(FormImgRef.pbBuildingPastureTL.Image)
```

```

gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "PastureTR" 'Print the top right of a pasture
        bmplImage = New
    Bitmap(FormImgRef.pbBuildingPastureTR.Image)
        gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "PastureBL" 'Print the bottom left of a pasture
        bmplImage = New
    Bitmap(FormImgRef.pbBuildingPastureBL.Image)
        gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "PastureBR" 'Print the bottom right of a pasture
        bmplImage = New
    Bitmap(FormImgRef.pbBuildingPastureBR.Image)
        gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    End Select
    Next
    Next
End Sub

```

```

'Subroutine prints a quadrant of the mini-map
Sub subPrintMinimap(ByVal intXLower As Integer, ByVal intXUpper As
Integer, ByVal intYLower As Integer, ByVal intYUpper As Integer, ByRef
gfxPlaceholder As Graphics)
    For intY = intYLower To intYUpper
        For intX = intXLower To intXUpper
            'Define the destination rectangle:
            rectGraphicsDestination = New Rectangle(intX * 2 - intXLower * 2,
intY * 2 - intYLower * 2, 2, 2)
            Select Case arrMapBlueprint(intY, intX).ptyTileType
                Case "Plains" 'Generate a plains graphic at the minimap location
                    rectGraphicsSource = New Rectangle(0, 0, 2, 2)
                    bmplImage = New
                Bitmap(FormImgRef.pbMinimapSource.Image)
                    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
                Case "Flora" 'Generate a flora graphic at the minimap location
                    rectGraphicsSource = New Rectangle(0, 2, 2, 2)
                    bmplImage = New
                Bitmap(FormImgRef.pbMinimapSource.Image)

```

```

        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Hill" 'Generate a hill graphic at the minimap location
        rectGraphicsSource = New Rectangle(0, 4, 2, 2)
        bmpImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Forest" 'Generate a forest graphic at the minimap location
        rectGraphicsSource = New Rectangle(0, 6, 2, 2)
        bmpImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "RiverSource" 'Generate a riverSource graphic at the
minimap location
        rectGraphicsSource = New Rectangle(0, 8, 2, 2)
        bmpImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Stream" 'Generate a stream graphic at the minimap
location
        rectGraphicsSource = New Rectangle(0, 10, 2, 2)
        bmpImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "River" 'Generate a river graphic at the minimap location
        rectGraphicsSource = New Rectangle(0, 12, 2, 2)
        bmpImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Coast" 'Generate a coast graphic at the location
        rectGraphicsSource = New Rectangle(0, 14, 2, 2)
        bmpImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
        gfxPlaceholder.DrawImage(bmpImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Sea" 'Generate a sea graphic at the minimap location
        rectGraphicsSource = New Rectangle(0, 16, 2, 2)

```

```

bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
    gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
        Case "Mountain" 'Generate a mountain graphic at the minimap
location
            rectGraphicsSource = New Rectangle(0, 18, 2, 2)
            bmplImage = New
Bitmap(FormImgRef.pbMinimapSource.Image)
                gfxPlaceholder.DrawImage(bmplImage,
rectGraphicsDestination, rectGraphicsSource, GraphicsUnit.Pixel)
        End Select

        Next
    Next
End Sub

```

```

'Subroutine that prints a single maptile to the screen
Sub subPrintTile(ByVal intXCoord As Integer, ByVal intYCoord As Integer,
ByVal intXLower As Integer, ByVal intYLower As Integer)
    Dim intX, intY As Integer
    intY = intYCoord * 40
    intX = intXCoord * 40
    rectGraphicsDestination = New Rectangle(intX, intY, 40, 40)
    rectGraphicsSource = New Rectangle(0, 0, 40, 40)
    Select Case arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyTileType
        Case "Plains" 'Generate a plains graphic at the location
            bmplImage = New Bitmap(FormImgRef.pbPlainsTile.Image)
            gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "Flora" 'Generate a flora graphic at the location
            bmplImage = New Bitmap(FormImgRef.pbFloraTile.Image)
            gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "Hill" 'Generate a hill graphic at the location
            bmplImage = New Bitmap(FormImgRef.pbHillTLTile.Image)
            gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
        Case "Forest" 'Generate a forest graphic at the location
            bmplImage = New Bitmap(FormImgRef.pbForestCentreTile.Image)
            gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    End Select
End Sub

```

```

Case "RiverSource" 'Generate a riverSource graphic at the location
bmplImage = New Bitmap(FormImgRef.pbRiverSourceTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "Stream" 'Generate a stream graphic at the location
bmplImage = New Bitmap(FormImgRef.pbStreamTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "River" 'Generate a river graphic at the location
bmplImage = New Bitmap(FormImgRef.pbRiverTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "Coast" 'Generate a coast graphic at the location
bmplImage = New Bitmap(FormImgRef.pbCoastTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "Sea" 'Generate a sea graphic at the location
bmplImage = New Bitmap(FormImgRef.pbSeaTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
Case "Mountain" 'Generate a mountain graphic at the location
bmplImage = New Bitmap(FormImgRef.pbMountainTile.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
End Select

Select Case arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyBuildingType
Case "Cruck Hut" 'Print a cruck hut
bmplImage = New Bitmap(FormImgRef.pbBuildingCruckHut.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Lumberjack Hut" 'Print a lumberjack hut
bmplImage = New
Bitmap(FormImgRef.pbBuildingLumberjackHut.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Cottage" 'Print a cottage
bmplImage = New Bitmap(FormImgRef.pbBuildingCottage.Image)
gfxGameMap.DrawImage(bmplImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

```

```

Case "Reeve House" 'Print a reeve house
    bmpImage = New
    Bitmap(FormImgRef.pbBuildingReeveHouse.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Brewery" 'Print a brewery
    bmpImage = New Bitmap(FormImgRef.pbBuildingBrewery.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmTL" 'Print the top left of a farm
    bmpImage = New Bitmap(FormImgRef.pbBuildingFarmTL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmTR" 'Print the top right of a farm
    bmpImage = New Bitmap(FormImgRef.pbBuildingFarmTR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmBL" 'Print the bottom left of a farm
    bmpImage = New Bitmap(FormImgRef.pbBuildingFarmBL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FarmBR" 'Print the bottom right of a farm
    bmpImage = New Bitmap(FormImgRef.pbBuildingFarmBR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FisheryT" 'Print the top of a fishery
    bmpImage = New Bitmap(FormImgRef.pbBuildingFisheryT.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "FisheryB" 'Print the bottom of a fishery
    bmpImage = New Bitmap(FormImgRef.pbBuildingFisheryB.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

Case "Water Mill" 'Print the left of a water mill
    If arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyTileType = "Plains" Then
        bmpImage = New
        Bitmap(FormImgRef.pbBuildingWaterMillLP.Image)

```

```

    Else
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingWaterMillLR.Image)
    End If
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "Water MillR" 'Print the right of a water mill
        If arrMapBlueprint(intYCoord + intYLower, intXCoord +
intXLower).ptyTileType = "Plains" Then
            bmpImage = New
    Bitmap(FormImgRef.pbBuildingWaterMillRP.Image)
    Else
        bmpImage = New
    Bitmap(FormImgRef.pbBuildingWaterMillRR.Image)
    End If
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "QuarryM" 'Print the mountain section of a quarry
        bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryM.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "QuarryTL" 'Print the top left of a quarry
        bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryTL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "QuarryTR" 'Print the top right of a quarry
        bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryTR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "QuarryBL" 'Print the bottom left of a quarry
        bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryBL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)
    Case "QuarryBR" 'Print the bottom right of a quarry
        bmpImage = New Bitmap(FormImgRef.pbBuildingQuarryBR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "MineT" 'Print the top of a mine
        bmpImage = New Bitmap(FormImgRef.pbBuildingMineT.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

```

```

Case "MineB" 'Print the bottom of a mine
    bmpImage = New Bitmap(FormImgRef.pbBuildingMineB.Image)
    gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "PastureTL" 'Print the top left of a pasture
        bmpImage = New Bitmap(FormImgRef.pbBuildingPastureTL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "PastureTR" 'Print the top right of a pasture
        bmpImage = New
Bitmap(FormImgRef.pbBuildingPastureTR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "PastureBL" 'Print the bottom left of a pasture
        bmpImage = New Bitmap(FormImgRef.pbBuildingPastureBL.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    Case "PastureBR" 'Print the bottom right of a pasture
        bmpImage = New
Bitmap(FormImgRef.pbBuildingPastureBR.Image)
        gfxGameMap.DrawImage(bmpImage, rectGraphicsDestination,
rectGraphicsSource, GraphicsUnit.Pixel)

    End Select
End Sub

End Module

```

Code for ClassMapTile

```

Public Class MapTile
    Private strTileType As String
    Private strBuildingType As String

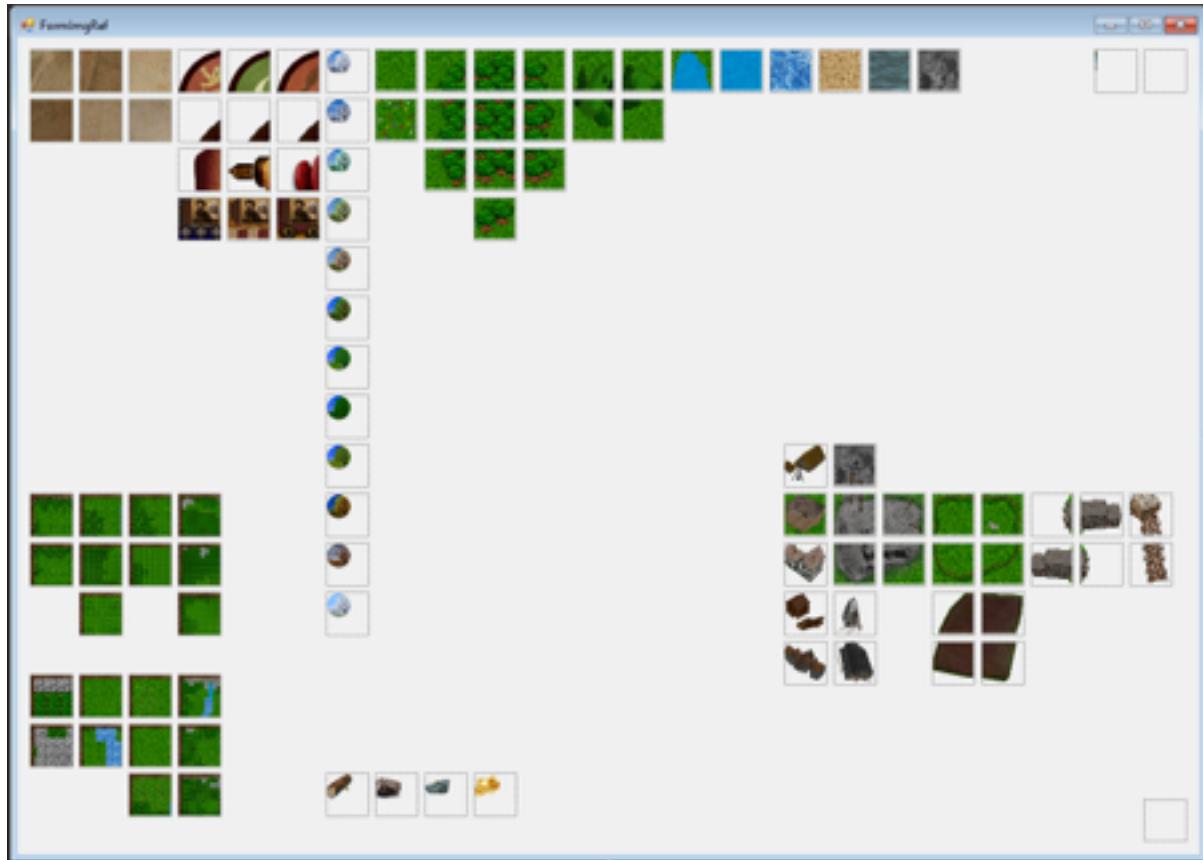
    Public Property ptyTileType As String
        Get
            Return strTileType
        End Get
        Set(value As String)
            strTileType = value
        End Set
    End Property

```

```
Public Property ptyBuildingType As String  
    Get  
        Return strBuildingType  
    End Get  
    Set(value As String)  
        strBuildingType = value  
    End Set  
End Property
```

End Class

Image of FormImgRef



END OF APPENDIX

END OF COURSEWORK