Requirements

Fully Met Requirements

This list is an at-a-glance list of the requirements we have fully met, with a short explanation of how.
#Requirement number - Explanation of how requirement is met
#1 - Only one computer is required to play the game.
#2 - The game only needs a keyboard and mouse as input methods to play the game
#3 - The game will display on a monitor, at any resolution with 16:9 aspect ration maximized, or windowed otherwise
#5 - Extensive playtesting and removal of existing bugs(see the 'Features we've fixed' header) has yielded no new bugs. Of course we cannot be 100% sure of no bugs existing, but every unit test in the final build passes, and all of our black box tests have passed.
#6 - All code that we have written should be easily understandable. Where we have integrated with existing code which was not very maintainable we have commented why code is doing what it is doing.
All new methods (other than very trivial ones) have docstrings.
#7 - The interface and menus are intuitive, and convey what they do very well
#8 - The structure of the game is conveyed to the player in an easy to understand way, both via the use of a current phase display, and a help box that walks players through each phase.
#9 - All plots of land are unallocated at the start of the game.
#10 - All players start with the same amount of money. This amount is very easy to change in code.
#11 - The market starts with 16 units of food, 16 units of energy, 12 Robotics and no ore.
#13 - Via the use of the Main menu that we implemented(See 'Features we've added'), the name for any human player can be input before the game starts.
#14 - The game is split into 5 phases: Acquisition, Purchase and Customisation, Installation, Production and Auction. However, auction is split into two sub-phases, the first being the auction listing phase, where both players can list an auction each. The second being the auction buyout phase, where both players can buy out the auctions of the other player.
#15 - The game allows the player to purchase unallocated plots of land in the acquisition phase.
#16 - Players can optionally buy and sell resources in the market during the purchase phase.
#17 - Roboticons can be upgraded in the in-game UI to make them more effective at gathering certain resources.
#18 - Players may install multiple robotics on any plot of land that they own.
#19 - During the production phase, players are given resources that their tiles have produced that turn.
#21 - Players can buy out other players auctions, provided they have the required balance.
#22 - The game cuts to an end of game scene with a scoreboard and the option to return to the main menu or quit, when the production phase is reached without any tiles left to buy.
#23 - A player's score is calculated based on the resources on the tiles that they own, the roboticons installed on those tiles and the resources the player has in their inventory.
#24 - The player with the highest score is clearly shown as the winner in the end of game screen and is given the title of vice chancellor.
#25 - The markets buy and sell prices for resources vary depending on its stock. The market has a 'target' amount of resources for each resource. If the market has less than the target, the buy price will increase whilst the sell amount with decrease, and vice versa.
#26 - Same reasoning as #25.
#27 - This is also implemented as part of #25, the market will always charge more for its resources than it buys them for.
#28 - A gambling feature has been implemented, in the form of a double or nothing game that can be played during the market phase, where the player pays £10 to play and rolls a number between 0 and 100. If their roll is greater or equal to 50, then they get £20 back, otherwise they get nothing
#29 - Random events have been fully implemented. A full explanation and justification of our implementation can be seen in the new features list.
#30 - Random events only start during the production phase.
#31 - Random events are made obvious to the player by all affected tiles having a texture unique to that random event attached to them.

#32 - Any data that is required to be refreshed is updated when appropriate, like the roboticon list when a player purchases robotics, or the markets stock/balance when resources are bought and sold from it
#33 - Menu's are accessible by buttons placed on the resource bar at the top of the screen. When these menus are open, the map is still fully visible and can be interacted with.
#34 - An option menu with a resume and exit button exists and can be accessed via an onscreen button
#35 - Software speed is not an issue with the game, and all transitions are seamless.
#36 - The game runs way above 30fps on the lab computers, usually sitting around 60fps at all times.
#38 - The purchase and customization, installation and auction phases are all time-limited.
#39 - The game incorporates the university campus as a 3d model within the map. You can buy tiles of the map which contain parts of the university campus, to give the feeling of interacting with the university.
#40 - The UI was designed to invoke a futuristic aesthetic. Also, the events were designed to be post apocalyptic-esque to keep the futuristic theme.

Partially met requirements

#12 - Support for 2 players, one of which can be an AI, is included in the final version of the game. However, support for more than 2 players is not implemented, and we did not have time to implement it properly due to time constraints, as we implemented other higher priority requirements first.

Unmet requirements

#4 Cross platform compatibility - We decided against making the game playable on different platforms, for two reasons. One being that as a team of 6, we simply do not have the time to test each change to the project on Windows, Linux and MacOS.
#20 Access to market during auction phase - We did not implement this feature as the market could already be accessed during the Purchase phase. Making it available during this phase as well would have required substantial changes to the code base and underlying architecture. We came to the conclusion that these side effects outweighed the need for this low priority requirement.
#37 saving and loading - We could not realistically implement this feature as the existing code was not properly designed for this, or maintainable enough for this to be possible. Saving would require all game state to be serialized and stored. Bugfree appeared to be planning on implementing this using the built in C# attribute Serializable[1] (there were many classes where they had applied the attribute already), however they hadn't considered that to serialize a class all of its member types had to be serializable. However many classes had UnityEngine.GameObjects[2] as members of the classes, and they are not serializable. If they had separated out this dependency we could have serialized the classes. We could not separate the classes as they were extremely tightly coupled. We also considered serializing the objects manually, but this would be extremely time consuming, we would have to write serialization methods for each class. This would also be extremely hard to maintain as any changes to the class structure would likely break our serialization methods.

New, Changed & Fixed Features

Below is a list of new features, changed features and bug fixes we have created or would have liked to. These are described here to give you an understanding of the work we completed and more information on how/why requirements were met or not met. They also include a justification for why any existing software was removed or changed. When possible, related features are listed one after another.

New Features

Random events - RandomEvent.cs RandomEventManager.cs Events.json
There was a beginnings of a random events system but very little behaviour was actually implemented. The existing system was overcomplicated for what was required. We decided to start the system again from scratch. RandomEventManager controls the starting and update of events. Each type of random event gets its own instance of RandomEvent. RandomEvent holds all information about the event like what type of tiles it can be applied to and its duration etc. The manager then applies these events to tiles throughout the game. Each event has an icon which is displayed on all affected tiles. The effects events have on resources are multipliers which are applied to the tiles production, eg a +50% bonus to food.

RandomEvents are configured in the Events.json file, and all events in the file are loaded automatically. This means that new events can be added / removed by simply making changes to the config file. You can configure the events: title, description, duration, number of tiles to affect, whether it should affect only neighbouring tiles or random tiles all over the map, tiles with/without roboticons or both, the affect on resources, and its image all from the config file.

Phase timers - Timeout.cs, HumanGUI.cs, GameManager.cs & canvasScript.cs

Created a timeout class which is used in the Purchase, Installation and Auction phases to limit how long a player has to complete their turn. If the user runs out of time, they game automatically ends their phase. The class also has a seconds remaining property which is displayed to the user so that they can see how long they have.

Roboticon icons on tiles - TileObject.cs

During playtesting we found that it felt like nothing was happening when we installed a roboticon on a tile but the only visual change was the production numbers in the tile detail window. Now, when a roboticon is installed onto a tile, a roboticon icon is displayed on the tile so that users can quickly see which tiles they have roboticons installed on.

Displaying the AI's turn to the player - HumanGUI.cs

When it is an AI's 'turn', It will display the AI as the current player for 3 seconds, before they
actually act. This is to try and show that the AI is thinking, as the AI instantaneously taking it's turn made players confused whilst playtesting.

End of game screen - endGameScript.cs

At the end of the game, a scoreboard is displayed, showing player names and what score each of them got, in order of highest score to lowest.

For this, a ScoreboardEntry structure was created, which contains a player's name and score

A list of ScoreboardEntry's sorted in order of highest to lowest score is passed to the end of game script, so that it can easily output the contained data to the UI.

This scoreboard is also used to display the winner's name as the vice-chancellor.

Auction - AuctionManager.cs, auctionBuyWindowScript.cs, auctionSellWindowScript.cs

After the production phase, the game goes to the Auction List sub-phase allowing them to list an auction containing the resources they wish to sell and the price they want for it. After this it goes to the Auction Bid sub-phase, allowing the players to buy the resources if the other player listed an auction. At the end of the turn, the list of auctions is cleared.

Start of game menu - mainMenu.unity, mainMenuDayNightCylce.cs, MenuScript.cs

We created a main menu which features the ability to enter player names, select if the second player is AI or human and start or quit the game. The main menu was also edited slightly to look more aesthetically pleasing. This was done through re-using the terrain and map to add a background to the main menu. A simple day night cycle was also introduced to prevent the main screen looking too still.

Gambling - HumanGui.cs, Market.cs gamblingScript.cs

We created a double or nothing game. The brief stated this should be available during the purchase phase so we made it a part of the market, as they are accessible at the same time, and can make use of some of the same code.

Changed Features

How the map is stored - Models/Map/mapTerrain & Models/Map/mapBuildings

We changed the .blend map files to .obj files and overhauled how Bugfree textured the map so that it had a single texture, as obj files need a texture to be displayed properly, whilst the blend file had one stored internally.

Improved Exceptions - Throughout the entire code base

We changed any System.Exceptions that were thrown to more descriptive types of Exception (eg. InvalidOperationException, ArgumentException) to make debugging easier. This also prevents Try Catch statements from catching all errors (as System.Exception is the base class for all exceptions) even when the error should not be caught. Any new exceptions we introduced were not System.Exception.

Selecting and hovering tiles - mapManagerScript.cs & TileObject.cs

As well as the border of a tile changing colour when a tile is selected / hovered over by the user, the whole tile is slightly coloured to make the affect easier to see. This is especially useful on smaller / lower resolution screens and for those with poor eyesight.

Help box - helpBoxScript.cs

We stopped the help box from appearing every turn after the player has clicked the hide button, as it got in the way quite frequently whilst playtesting.

Unit tests - Assets/Editor/Tests

We converted all existing tests into NUnit/UnityTestTools[3] tests. Bugfree had created their own test suite, but we did not know if the test suite was correct as it had no tests for itself. Therefore we decided to migrate all tests to the new framework. This also gave us an opportunity to check their correctness. All our new tests were also written for UnityTestTools.

Pause menu - canvasScript.cs

Removed the save button from the pause menu, as it is no longer a feature in the game. For more information see the unmet requirements above.

AI / Computer player - Player.cs & AI.cs

We have used Bugfree's AI class to implement a simple AI that can play the game. In the Acquisition phase the AI chooses a tile to buy at random that it can afford and then purchases it.

Fixed Features

Install roboticon button fix - roboticonWindowScript.cs

We fixed a bug where roboticons which had already been installed still had an install button displayed next to them in the roboticons window. Now, there "Installed" is displayed instead.

No installed roboticons crash fix - roboticonWindowScript.cs

We fixed a bug where if the user did not own any roboticons, and opened the roboticons window in the purchase phase, the entire UI would disappear.

Roboticon list not being updated properly fix - roboticonWindowScript.cs

We fixed a bug where nothing would be displayed in the Roboticon list when opened, if players bought Roboticons when the Roboticon window was not open.

Roboticon upgrade menu fix - roboticonWindowScript.cs

A bug where in the Roboticon upgrade menu, it would always display 'No' next to the 'installed' field. The text next to the 'installed' field now correctly shows whether the Roboticon is installed or not

Buying Roboticons from the market fix - Market.cs

When the player bought Roboticons from the market, the markets stock wasn't actually being changed, nor was its balance. This allowed for an unlimited amount of Roboticons to be purchased from the market. This has now been fixed, so that when the player buys Roboticons from the market, the stock and balance update accordingly

Hover and select tile fix - TileObject.cs & mapManagerScript.cs

We also fixed two bugs with tiles. One where the hovering effect would apply to the wrong tile if the mouse moved off of the map or onto a selected tile, and another where it would be impossible to select the last selected tile from last turn until the next player selected another tile first

Bibliography

[1] Microsoft. "Serialization (C#)", msdn.microsoft.com. [Online]. Available: https://msdn.microsoft.com/en-gb/library/mt656716.aspx [Accessed: Feb. 20th 2017]

[2] Unity Technologies. "Unity - Scripting API: GameObject", docs.unity3d.com. [Online]. Available: https://docs.unity3d.com/ScriptReference/GameObject.html [Accessed: Feb. 20th 2017]

[3] Unity Technologies. "Unity - Unity Test Tools" unity3d.com. [Online]. Available: https://unity3d.com/learn/tutorials/topics/production/unity-test-tools [Accessed: Feb. 20th 2017]