Architecture

Architectural Tools

As the final system is a continuation of the initial concrete implementations, the tools used are largely unchanged, as it would be impractical and unnecessary to refactor to different languages or programs in this case.

StarUML has been used to display the final concrete architecture. It was previously used by 'Duck Related Team Name' in Assessment 2 and we were able to continue from the earlier model they created, rather than needing to recreate the already existing classes from scratch. The software is available as a free download from its website and is designed for the UML standard we are using to describe our architecture, making it suitable for our use. It is also less strict than some other programs in terms of formatting, making it quicker and easier to create and update the diagrams.

The game itself is written in Java, as it was considered the most suitable out of those the whole group has experience with. We have also used the library libGDX to provide a premade set of abstract tools for rendering the game. This library is open-source and well-documented, making it an excellent choice to use. Our UML diagram is too large to be readable in this document and as such can be found on our website at:
http://gandhi-inc.me/downloads/FinalUMLArchitectureImage.png
We have also created a traceability matrix to help track the changes made and modules that have been added in order to satisfy the requirements criteria. It can be found on our website at:
http://gandhi-inc.me/images/TracabilityMatrix.png

Classes

Existing from previous architectural documentation

Main/SplashScreen/MainMenu

Main simply serves to initialise the game and create the splash screen. This in turn renders the team logo before presenting the main menu.

The MainMenu creates a Drawer and a set of TTFonts and uses these to display the initial menu for the game using a table of buttons. This allows the user to view the Leaderboard or How to Play views, as well as launch the game itself.

The MainMenu has had some more options added since the last architecture report and the SplashScreen logo has changed with the groups working on the program.

GameScreen

The GameScreen class provides functions for the rendering and manipulation of the game's GUI that is presented to the player, including the map [1.1.1], tile grid and HUD [1.1.3]. The GameScreen makes use of tables, Labels and TextButtons from libGDX to create the interface and updates them as values change. The resources, money and un-deployed roboticons the current player has are displayed and buttons to claim tiles [2.1.1], access the minigame and pause the game timer are provided. It also initialises the GameEngine, which provides the values to be displayed in the GUI.

This class has had some substantial changes as more features have been added and new elements have been added to the GameScreen to provide interaction with the player.

GameEngine

The GameEngine controls the game state, maintaining information on the current phase [5.1.1,2], containing and calling upon various game elements, such as the GameTimer, Players, Tiles and Market, and providing functions for manipulating the game systems [4.1.3]. It is also able to determine when the last tile has been claimed and begin the endgame procedures at the end of the round, including determining the winner based on calculated player scores [10.1.1,2].

Like the GameScreen, this is a core class for the functions of the game and has had a number of changes to support more recently added features. Parts have also been reworked during the previous development phase to support additional players.

GameTimer

Some game phases have limited durations, so this provides the capability to track and display the remaining time [5.1.3]. It uses a timer object from libGDX, which decrements as time elapses, and passes its output to the parent GameEngine. This class has remained unchanged from earlier stages.

Drawer/TTFont/LabelledElement/College

The Drawer class provides functions for rendering frequently used objects, such as simple shapes and text, to reduce complexity and repetition elsewhere in the game programming. It accepts function calls with arguments such as dimensions and colour, and then uses these with the libGDX shape renderer to display them on the screen.

LibGDX requires fonts to be in a bitmap format to be displayed, so this class converts fonts from ttf to bitmap format. It takes a ttf file and parameters for size and colour and outputs a corresponding object that libGDX can render. All displayed text is implemented through this class.

LabelledElement is needed to allow labels to be recognised by libGDX and is utilised for the creation of various GUI elements.

The College class assigns the logo image and colours for identifying each player, styled after the university's colleges.

This set of classes was fully implemented in earlier stages and remains largely unchanged.

Tile

The Tile class implements a libGDX button to detect when it has been clicked. Each tile corresponds to a specific area of the game map and stores various pieces of information about itself, such as the owning Player, assigned Roboticon [6.1.5] and production values [1.1.2], as well as providing functions for rendering tooltips, calculating resource production [7.1.x] and displaying images for the 'chancellor' minigame.

This class has seen some small additions to support the chancellor minigame and display additional information, but the core functions remain largely unchanged. The tile now stores a reference to the parent GameEngine, allowing it to retrieve and reward the appropriate player when the Chancellor is caught.

Player

The Player class represents the different players in the game and stores information related to them, such as stocks of resources, money and owned Tiles. It provides functions to retrieve and manipulate these values, as well as calculate the player's score [10.1.1].

This class has seen some changes during the previous development phase, but has needed little alteration during this one. Human players can also now be given names, which will be recorded in the leaderboard scores if they win.

Roboticon

Roboticons are stored within their assigned Tile rather than the GameEngine and have a Player and a Tile that they are associated with. They store their productivity levels, which modify the Tile's output [3.1.1], and provide functions for retrieving and updating these, but generally have a passive role within the game implementation.

This class remains unchanged from earlier phases of development.

Overlay

The overlay class is an extension of the libGDX stage and is used to create 'popup' messages and menus, such as the Roboticon upgrade menu [6.1.4].

This class remains unchanged from earlier phases of development.

Market

The Market class is an extension of the libGDX table and is stored within the GameEngine. It provides the interface and functions required to allow players to buy and sell Resources and to trade with each other [6.1.1,2][8.1.x].

The functionality for buying and selling resources remains unchanged from earlier phases of development, but support for alternatively trading with other players has been added since the previous architectural document.

MiniGameScreen

The MiniGameScreen class implements the gambling minigame and is created by the GameScreen. It presents the player with three buttons, styled after facedown playing cards, of which the player can choose one. Upon selection, the player will then receive one of several rewards, which could be money, an additional roboticon, or nothing, and a monetary fee will be deducted for playing [9.1.1].

PlayerSelectScreen

The PlayerSelect class is created by the MainMenu and creates the GameScreen in turn. It provides a table of buttons and text labels, allowing the user to set the number of human and AI players in the game [4.1.1,2]. It also allows players to enter their names and assigns them a college, determining their icon and colour.

Trade

The Trade class provides support for trading between players. Each instance of the class represents a single trade offer and stores the information involved, including the quantity of resources, price, sending player and target player [8.1.3].

RandomEvent/Earthquake/Malfunction

The RandomEvent class is an abstract class that was added during the previous development phase and provides a foundation for various random events that may occur. Earthquake and Malfunction are implementations of the base RandomEvent class, each implementing functions for a specific random event and its associated effects.

Earthquake randomly selects a set of tiles and ‘damages’ them, temporarily reducing their productivity values. Malfunction randomly selects a single roboticon and temporarily sets its productivity modifier to zero; preventing it from produces resources until the event expires.

AiPlayer

This class is an extension of Player and provides a set of functions that can be called by the GameEngine to execute turn phases autonomously, creating a simple AI that can be used for opposing players [4.1.1].

Newly implemented classes since previous architectural documentation

Chancellor

This class is responsible for the ‘Catch the Chancellor’ minigame that appears at the end of phase 3. When called it will run for a set period of time (15 seconds), during which it will randomly select tiles on the map and display the ‘chancellor’ on them for a short time. If the current players manages to click on a tile occupied by the chancellor, they will receive a small monetary reward (50 monetary units).

HowToPlay

The HowToPlay class is created by the MainMenu and contains text elements explaining the game’s ‘story’ and provides a link to an online pdf of the user manual, which explains the various elements of the user interface, along with how to play the game.

LeaderboardFrontend/LeaderboardBackend

This pair of classes work together to provide a leaderboard of player scores. The Frontend is created by the MainMenu and provides the graphical display and user interaction for the leaderboard, making use of the Drawer and TTFont classes. The top three scores are displayed in order.

The Backend is created by the FrontEnd and provides the actual functionality for the leaderboard, storing, retrieving and organising scores in a text file through use of standard file IO functions. Winning players and their scores are written to the text file, while another function reads the file as an arraylist and extracts and returns the top three.