Implementation Report

Summary

As part of Assessment 3 we were required to take over an existing project and implement the remaining requirements of the scenario for this project. Our team was able to implement all of these requirements, including: a save and load system; the neutral player and a mini-game triggered by entering the Vice-Chancellor's sector.
The architecture of the software we received was very difficult to work with. Large amounts of the code were highly entangled making it very difficult to extend the code and follow how things worked. This lead to slower development and made it not possible to implement large amounts of unit testing. This was because functions that were used to modify the game state also updated properties of the scene meaning that it was not possible to isolate sections of code to automatically test. Without a fundamental redesign of the game's architecture extensive automated unit testing will not be possible to implement.

Format and Process

This document outlines the most significant additions and changes that have been made to the projects implementation since Risky Developments commenced working on the software. Each entry contains a description of the change that has been made; a list of the files significantly affected and an explanation of why the change was made and how the change was implemented.
Throughout the document the requirements for the project [1, 2] are referred to for justifying the additions and changes.

Additions

Changes Made:	Save and Load System
File Modifications:	GameData.cs; SavedGame.cs
Explanation:	Implemented save and load functionality. When calling the Save method an xml file is created that stores a serialized version of the GameData class, containing all the necessary properties to instantiate a game. The Load method deserializes a given xml file and which we can then instantiate a game from. This addition was made as per requirement F7 the game should be capable of being loaded and saved.

Changes Made:	Minigame System
File Modifications:	Bird.cs; MinigameManager.cs; MovingPillars.cs; Minigame.unity
Explanation:	Implemented the minigame which is triggered when the sector which contains the Vice Chancellor, (VC = True), is captured, as per requirement F3. A copy of the current game is saved to a temporary save slot and the scene is switched to the Mini Game Scene. The player flies a bird through some pipes in order to gain

	coins until the hit an obstacle or collect 10 coins. The scene is then switched back to the scene and the temporary save game is reloaded so that the main game may resume.And as per requirement F4 the player is rewarded based on the amount of coins that they collect.

Changes Made:	Dialog System
File Modifications:	TestScene.unity; Dialog.cs
Explanation:	As the game should be easy for new players to interact with, N12, we added a dialog system to communicate various information to players and to allow them to select input. Dialogs are specifically used for: the game pause menu; telling the player the outcome of the mini-game; displaying if a player has been eliminated or the game is over. To implement the dialog system a single dialog is used and by setting the dialog type different buttons are enabled and disabled. For example the EndGame Dialog type enables the Restart and Quit buttons and disables the other components but the SaveQuit type enables the Save & Quit button and the Quit button.

Changes Made:	Neutral Player
File Modifications:	Game.cs/CreatePlayers(); Player.cs; PlayerUI.cs
Explanation:	Added neutral attribute to Player.cs. Set and Get Controller methods were created to get/set if the player was being controlled by a human, neutral or none. PlayerUI.cs/Initialize(player player, int player_id) updated so that the UI displays if the player is neutral. This addition was made to fulfill requirement F1.

Changes Made:	Vice Chancellor Spawning
File Modifications:	Game.cs/InitializeMap(); Sector.cs
Explanation:	The InitializeMap() function has been extended so that when the map is setup one sector is selected to contain the Vice Chancellor. To support this the Sector.cs file had a VC attribute added which is false unless that sector was chosen to contain the Vice Chancellor. The Vice Chancellor's location is chosen by randomly selecting a sector until one that does not contain a landmark is found. If such a sector is found then its VC attribute is set to true; else another sector is chosen again. The addition was made in accordance with requirement F2.

Changes Made:	Main Menu Added

File Modifications:	MainMenu.unity; Menu.cs
Explanation:	Requirement N11 states that 'The game should have a simple Main Menu'. Therefore a Main Menu scene was added to the game where the player may choose to start a new game, with 3 or 4 players, or load a game from file. To meet requirement N3 every game must have 4 players in it so in the 3 player mode the game is started with 3 human players plus a computer controlled neutral player. The load game option enabled us to help fulfill F7.

Changes Made:	Player Eliminated and Game Over detection and notification
File Modifications:	Player.cs
Explanation:	The Defeat(Player player) was created to transfer all sectors owned by this player to the passed player, only if this player had been eliminated. Additionally the CheckForDefeatedPlayers() function was created which checks if a player has been eliminated and if they have then displays a dialog saying they have been eliminated.

Changes Made:	Added an End Turn button
File Modifications:	TestScence.unity; Game.cs/EndTurn()
Explanation:	An End Turn button was added to the main game scene which calls the EndTurn() method in Game.cs when pressed, this ends the current player's turn regardless of how many actions they have remaining. This was added to meet requirement N10.

Modifications

Changes Made:	Overhauled previous implementations commenting system
File Modifications:	Game.cs; Initializer.cs; Landmark.cs; Map.cs; Player.cs; PlayerUI.cs; Sector.cs; Unit.cs
Explanation:	The implementation inherited by this team contained very limited documentation of method functionality and what documentation was there did not use C#'s correct docstring notation. Therefore to help speed up our own work and make the project more manageable for any new team working on this project we replaced their method comments with proper docstring defining method functions; the parameters they take and what value they return.

Changes Made:	Game Initialization
File Modifications:	Initializer.cs/Start()

Explanation:	Initially the game when started would always start a new game. However implementing our requirements meant that we now needed to be able to start a new game, (with or without a neutral player) or load a game from file. To support this the Initializer.cs script was modified so that the type of game to start is read from the PlayerPrefs and then the game is started in this mode.

Changes Made:	Game Screen GUI
File Modifications:	TestScene.unity
Explanation:	The aesthetic of the main game screen was significantly overhauled to make it easier for the player to understand what is happening and so that it is more visually appealing, meeting requirement N12. A full justification of the changes made can be found in the Updated GUI Report [3].

Changes Made:	Rebalance conflict resolution algorithm
File Modifications:	Sector.cs/Conflict(Unit attackingUnit, Unit defendingUnit)
Explanation:	The conflict resolution algorithm has been overhauled as with the original implementation the outcome of combat was often very unpredictable, even with a high level unit vs a lower one. This made the game not very enjoyable to play. The algorithm was modified such that the outcome was based on the difference in unit strength plus bonuses. Then the winner was selected using a weighted random outcome where the higher strength player would have the higher chance to win and the weaker only a small chance. The modification lead to a much more predictable outcome, but with still a small chance of a surprise outcome. This made the game feel far more enjoyable to play. The change was made to better fulfill requirement N9 as too frequently stronger units were loosing to weaker ones making the combat just feel random and unfair.

Changes Made:	Unit levelup and material updating moved into separate methods
File Modifications:	Unit.cs/LevelUp(); Unit.cs/updateUnitMaterial()
Explanation:	LevelUp() initially contained the code for increasing a units level and updating the material showing a unit's level in the scene. The material update code was moved into its own method, updateUnitMaterial() so that the unit's material could be updated without changing the units level. This was needed as when a game is loaded the unit's materials need to be updated with their loaded level.

References

[1] SEPR “Requirements Document” Lazer Dolphin Games [Online]. Available: https://sepr-team-margaret.github.io/content/Req1U2.pdf [Accessed: Feb. 18, 2018].

[2] SEPR “Extended Requirements Elicitation” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/ExtendedRequirementsElicitation.pdf [Accessed: Feb. 18, 2018].

[3] SEPR “Updated GUI Report” Risky Developments [Online]. Available: http://www.riskydevelopments.co.uk/documents/UpdatedGUIReport.pdf [Accessed: Feb. 18, 2018].