Implementation Report

Stakeholders: Richard Paige, University of York Communications Office
Team: Barney Morgan, Cameron Smith, Harry Berge, Jake Phillips, Matthew Wilkie, Rob Weddell

Repository Link: https://github.com/SEPR4/AHOD2
NB: Submission content on "master" branch.

Approach

The implementation portion of assessment 3 required the team to overtake an existing project and develop the remaining requirements. The development team were able to implement the stated requirements whilst remodelling the inherited game to one that we feel would best implement the given requirements.

The architecture of the software was very problematic to work with, with any changes made to the code leading to many errors in other areas. It was decided to go down the route of a project overhaul, with changes to make the code more object-oriented and less script-like whilst developing the new game which is heavily inspired by their old game and documentation. This will allow for easier development as the developers won't be fighting the old code to implement changes any of the necessary changes as well as providing a clear structure for unit testing the new code.

The overhaul has led to some more general changes, such as loading all assets (cards, encounters, colleges, levels, departments) from JSON files which can now be seen throughout the project to allow for easy editing, as well as many other requirement specific changes that can be seen below.

One of the main changes to the existing implementation, as elaborated below, was the generation of the node map for the player to move around on. In order to better satisfy their requirements (particularly A4.1, B2.2), we made the node map directional meaning that the player can only travel in one direction (up the map). The reason for this is two-fold: the difficulty of the game can increase as the player advances, and areas are not strangely restricted so the progression seems more natural.

Additionally, we have introduced levels - each level has its own node map. The player is transferred to the next node map after defeating the boss of the last.

While all the requirements and milestones have been satisfied, there is a particular test, as seen in the document listed below, that is not met. The test (id 43) regards the balancing of the battles and encounters in the game meaning that the difficulty has not been properly tested and adjusted accordingly.

The changes as noted below are based on the updated requirements and testing document as seen here;
https://sepr4.github.io/web/submission/assessment3/updated/Req3.pdf
https://sepr4.github.io/web/submission/assessment3/testing/Matrix.pdf

Key

Addition - new class, function or file.
Changed - class, function or file remains but its function has been changed.
Deletion - class, function or file has been removed.

Changes to previous software

Changes made	Class names changed (Visual to Screen)
Components affected	**SailVisual** to SailScreen, **EncounterVisual** to EncounterScreen, **BattleVisual** to BattleScreen, **ShipVisual** to ShipViewScreen.
Justification	These changes are to make the code more understandable. The aforementioned classes extend a class Screen, so changing the naming scheme makes the code easier to understand for future developers.

Changes made	ShipViewScreen (ShipVisual) functionality
Components affected	Changed: ShipViewScreen class
Justification	Instead of listing objectives here as in the existing implementation, this screen now shows the current level and the player's deck. The health information is no longer needed as it is present on the StatsHUD (elaborated below).

Additional Features

Components affected	Addition: AHODScreen class
Justification	This project has many classes extending LibGDX's abstract Screen class. Each of those classes must implement every abstract method in the Screen class, however, this results in duplicate code. This class extends the Screen class and implements these methods with default options. The class also provides a stage object which all our Screen classes also use, further reducing duplicate code.

Components affected	Addition: FileManager class
Justification	In order to keep the static assets of the project well organised, we have created a class to load them from. This makes loading textures very easy and avoids repeated code.

Components affected	Addition: SailInputProcessor class
Req Reference	B4.1
Justification	We have implemented the scrolling up and down the map on the SailScreen and decided to add this functionality to a single class for ease. The class is responsible for checking the status of key presses and updating the camera position accordingly.

Components affected	Addition: NodeUtil class
Justification	This class was created to isolate the functions required to generate our node map. This means that creating tests for these functions was trivial. The purpose of these functions is to generate the node map according to variable criteria (such as the depth of the map) and strict criteria (such as max size of each row).

Components affected	Addition: ShipFactory class
Req Reference	A2.8, A4.1
Justification	In order to keep the game interesting and the code modular. We have created a ShipFactory class which allows generation of enemy ships. The functions take a difficulty parameter which is used to increase the difficulty of the enemies generated as the game progresses. The greater the difficulty, the greater the size of the enemy ship's deck and the power of its cards.

Components affected	Addition: CardManager / EncounterManager / BuildingManager class
Justification	In keeping with our previous projects and goals, we have made changes so that cards are stored in a JSON file and loaded at runtime. This allows for easy editing and addition of cards. These classes are responsible for loading the data into memory at game launch and also have utility functions relevant to each store of data.

Components affected	Addition: StatsHUD /MessageHUD / AnimationHUD class
Req Reference	B2.4
Justification	In order to avoid bad practice by having duplicate code in many classes, we have abstracted our on-screen elements as much as possible into the classes above. These classes provide tables containing actors which can be drawn to any screen. The StatsHUD contains the gold/health/score at the top of the screen and the MessageHUD displays status messages set by the player's actions. The AnimationHUD is used by the BattleScreen to display combat effects such as damage and heal splats. Each class extending AHODScreen can easily enable these interfaces with a single line of code.

Components affected	Addition: StyleManager class
Req Reference	B4.2
Justification	Many classes in our game use repeated and predictable structures to generate components related to the look-and-feel of the game. We have added the previously mentioned class to make it easy to create styles for components such as TextButtons, Labels and Fonts. This also helps to make user interfaces more friendly and easy to navigate.

Components affected	Changed: SailScreen class Addition: GameLevel class and levels.json
Req Reference	A2.2, A4.2
Justification	In order to add more direction and interest to the game, we have decided to create a series of levels rather than general objectives. This helps to vary the difficulty of combat and encounters as the progress of the player increases.

Components affected	Addition: StartNode class
Justification	We added a new type of node to better indicate the need to select a node at the start of the game (starting location). This makes it easier as these nodes have a different texture (so are more easily identifiable).

Components affected	Addition: TransitionScreen class
Justification	To improve the look-and-feel of our game we have decided to implement a new type of screen which handles transitions from one screen to another. This screen fades out the old screen and fades in the new one.

Components affected	Addition: BattleAI class
Justification	This class provides functions relating to enemy AI logic during battle. Given the battle situation (deck, hand, mana, etc..) the class will determine the next move the AI should make (whether to use a card, draw a card or end turn).

Components affected	Addition: MinigameScreen class
Req Reference	A2.3, B2.3
Justification	As part of the milestones for assessment 3, we have added this new screen implementation to allow the player to play the minigame. The class handles the appearance and functionality of the minigame.

Components affected	Addition: CardSelectionScreen class
Req Reference	A1.1
Justification	To reward players for defeating enemies in combat, they will be able to choose a 1 card from a random selection to keep. This screen appears after victory in combat.

Components affected	Addition: EndScreen class
Req Reference	A2.4
Justification	We have added a class to switch to when the end of the game has been reached. The constructor for this class takes a boolean describing whether the player has won the game and will create dialogues accordingly.