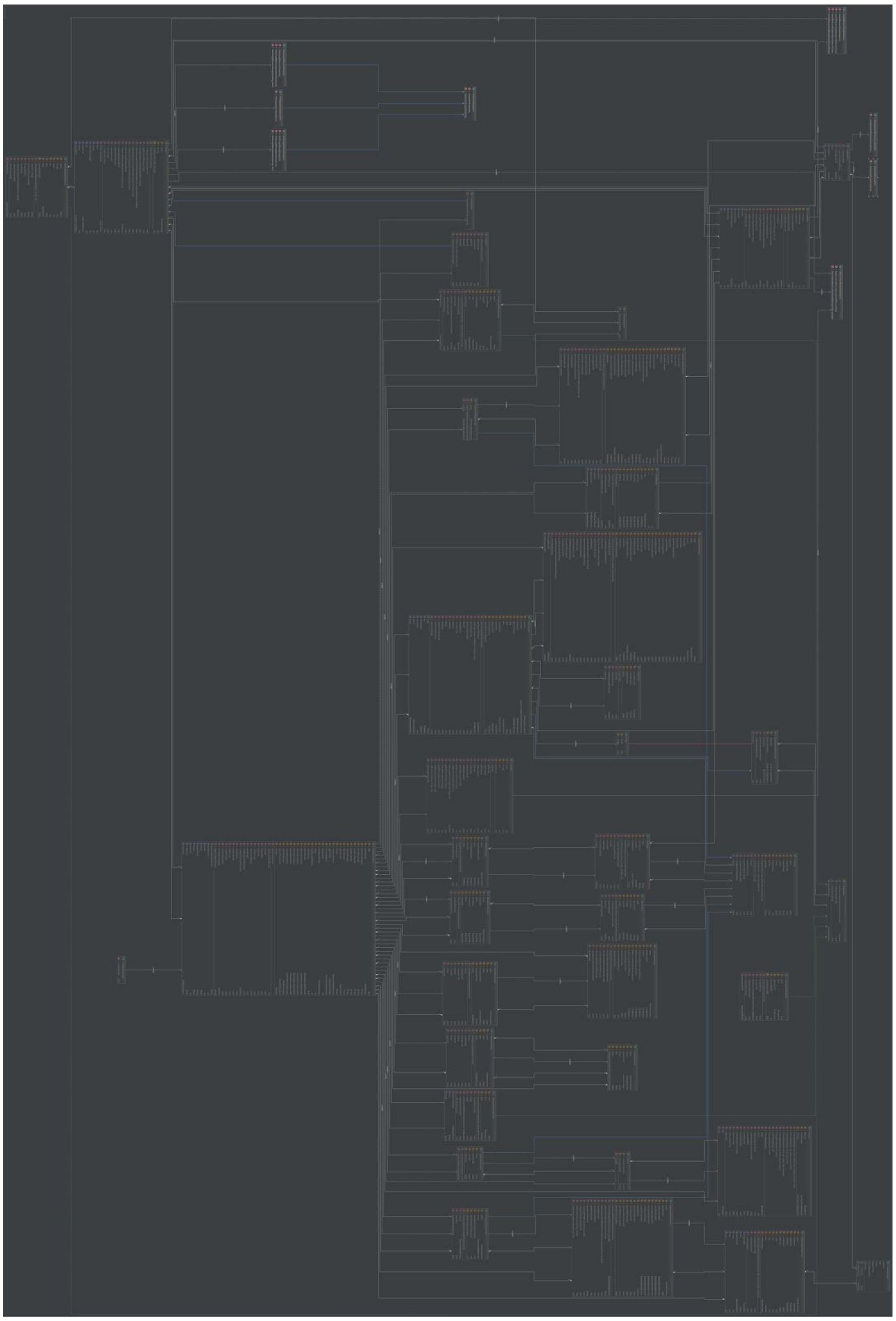
Architecture and Traceability

Duck Related Team Name's architecture for comparison (on their website).
Our UML class diagram (link to hi-res PNG version on our website so you can zoom in):



Full list of requirements.
The existing implementation was written in Java using libGDX[1]. The existing tests were written in JUnit[2], and Gradle[3] was used as the build tool. We made no changes to the tools / languages used as the existing tools were suitable for the requirements and changing them would cause more problems that would outweigh any benefits.

We did not make any major changes to the existing architecture as we were generally happy with it. We chose the project in part because of the separation between the user interface layer and the behavioural layer in the architecture. The existing architecture also used dependency injection throughout, which we continued as it makes the code significantly more testable.

To implement the vice-chancellor game (requirement #49), we made use of the already existing random event system that the game utilized. Building on top of the existing foundation was easy, all we had to do was add a new playerEffect in PlayerEffectSource for the vice-chancellor to appear as an event.
We made additions to GameScreenActors.java as it is part of the UI layer that acts upon the main game screen. These additions were to allow us to display a label to the player so they know when the vice-chancellor game has started and ended. We also added a property, chancellorsCaught to the Player class. This allows us to calculate the bonus score for winning the vice-chancellor mini-game at the end of the game.
We created Chancellor.java, a class in the UI layer. This contains code which handled the loading and unloading of the the vice-chancellor sprite, as well as drawing and animating it on the screen.
We updated GameScreen.java, the main game screen, to include fields for the chancellor mini game visuals that we described above, so that it could render them when the mini-game happens. The updates to GameScreen also detect when the user has caught (clicked) the vice-chancellor and calls the Player.caughtChancellor() method to so it can handle the behaviour separately from the UI. We realised that we could make use of existing code and avoid duplication of code by using the existing random events system to trigger the vice-chancellor game.

Adding support for up to four players (requirement #15) did not require many changes to the architecture.
Changes to the behavioural layer:
RoboticonQuest.reset() which resets the game's state and starts a new game, used to take a boolean parameter which controlled whether there was an AI player or not. Obviously this is not sufficient for more than 2 players so we updated it such that it takes the number of human players, and the number of AI players as integers.
RoboticonQuest.nextPlayer() is an extremely simple function that simply updated the current player index but was designed only for two players. It was updated to work for any number of players. It increments RoboticonQuest.currentPlayer except if the current player is the last player in the player list, in which case it is set to 0.
RoboticonQuest.getWinner() was renamed to getPlayerWithHigestScore() as it did not check if the game has ended, so the name was slightly misleading. The method was only designed for two players so it was updated to work for two or more players. getWinner() used to return

the winner as a string to display to the players at the end of the game. We split it into getPlayerWithHighestScore() which returns a player object, and getWinnerText() which creates the string announcing the winner so we had good separation of concerns.
Changes to the UI layer:
We changed the HomeMainMenu so that the user could configure how many human / ai players they wanted to play with (requirement #14). We added buttons to increase / decrease how many human and ai players there were in a game. We added some input handling code disabling buttons for invalid configurations ie. disabling the start game button if there is only one player and disabling add player buttons if there are already 4 players etc.
We changed EndGameScreen to display the scores of up to 4 players. The previous end of game screen implementation just had two labels in the EndGameActors table, which contained both players scores. When we updated to four player support (requirement #15), this was changed so that a label is created for every player that was in the game.

We replaced many uses of error codes (encoded as an Enum, see entity.enums.PurchaseStatus in the old architecture) and boolean return values with the use of exceptions. Using error codes in languages where exceptions are available is generally considered bad practice as the caller of a method may not check for an error code when the result is returned and treat the value as if it was expected data. This can create unexpected and hard to debug behaviour. Exceptions on the other hand will come with a stack trace and can give more information than a boolean value using a message. In Java you can also force the caller to surround your method with a try catch block if you declare your method can throw a checked exception. We made these changes to make the development of other features we had to implement easier, as the debugging would be much simpler.

We renamed fields and methods with poor names like implementPhase() to more descriptive things like runCurrentPhase() which more accurately described what they did. We also renamed some fields which did not follow Java convention and used python style names with underscores. These types of changes are safe and easy to make with tools like IDEs as they can check for usages in code and in documentation and change the everywhere at once. We made these changes because they made it more easy to understand, communicate about, and extend the code.

Bibliography

[1] Bad Logic Games. libGDX, [Online]. Available: https://libgdx.badlogicgames.com/ [Accessed: May. 1, 2017]

[2] JUnit. [Online]. Available: http://junit.org/ [Accessed: May. 1, 2017]

[3] Gradle. [Online]. Available: https://gradle.org/ [Accessed: May. 1, 2017]