General comments

Architecture Report

The majority of architecture class diagrams made good use of the UML notation but included unnecessary detail. Architecture diagrams should focus on the core components and functionality of the system and should not be a 1-1 mapping of the implementation reverse engineered into a class diagram. Methods and fields that are not discussed in the text should be omitted from the architecture diagram (if a property/method is important enough to appear in the architecture diagram, it's important enough to deserve some discussion in the report). Good traceability to requirements overall.

Update to deliverables

All reports covered all updated deliverables and changes were clearly highlighted. The absence of direct links to the updated documents in some reports made navigating to the latter unnecessarily awkward. Stronger reports included updates to the likelihood/severity of previously-identified risks on top of addressing omissions identified in the previous round of feedback.

GUI Report

The GUI reports tended to focus on describing visual and graphical interface aspects (like menus, layout, icons) instead of GUI design principles and interaction design. The strongest reports started with a brief and well motivated summary of key design principles (e.g., usability, accessibility), then explained their GUI design process and how the process was used to address these principles. Occasionally this included a discussion of how the player would be expected to interact with the interface. Some reports also traced their design decisions back to requirements.

Testing

Testing evidence was almost uniformly presented very well, in a structured and systematic way, often with a traceability matrix showing how requirements were covered by tests. A few reports showed that some requirements only had one test - but tended not to discuss whether or not this was a problem!

Descriptions of testing approaches were generally strong on explaining the methods that were used (e.g., white box testing) but not the overall process. Some reports left it unclear as to whether testing was done iteratively or incrementally, whether tests were designed up front and never changed, who did testing (was it the programmer or a separate sub-team), and whether testing was automated.

Test reports were quite well done overall, explaining the results of running tests of various types (e.g., unit tests, acceptance tests). Some reports went into a bit of detail about the challenges of doing testing with Unity. Weaker reports tended to omit summary statistics, along with an analysis of whether the team took the view that the testing evidence was sufficient to provide confidence that the code met its specifications.

Implementation

There was good evidence of following good programming practices in many of the submissions, in terms of using sensible programming and design patterns, providing clear interface documentation, and having a well-justified decomposition of the code into reasonable classes, methods and scripts. The stronger solutions avoided having several very large classes/scripts by distributing responsibility for functionality across multiple scripts/classes. Some commonly occurring problems included: not encapsulating lists of options in one place (e.g., a list of types of players or types of attacks, which appeared in the code in several places - this is difficult to maintain); using static unnamed constants liberally in the code; and poor documentation of business and game logic (though generally most submissions included sensible documentation of script/class interfaces).

The implementation reports were quite well done overall, and most teams commented on justification for not implementing certain requirements or partially implementing them. The strongest solutions had very clear traceability to requirements.