Defining assessment criteria and the corresponding Metrics:

Completeness of the solution direction (i.e. to what extent are all key functionalities covered in the design) [Functional completeness; functional correctness]

- All critical game functionalities are covered, and the submission clearly explains how they are addressed.
- All classes have relevant attributes, methods, relationships, and cardinalities, covering all required functionality.

Metric: The number of critical functionalities covered out of all the expected ones (e.g. 80% coverage)

The rationale behind the chosen solution direction [Functional appropriateness]

- Design rationale covers all required areas: classes, relationships, inheritance, cardinality, and design pattern.
- Solid and compelling justifications for the architecture and design choices are provided.
- Used Design Patterns are identified, their choice is explained and used appropriately, or the lack of application of Design Patterns is well justified.
- Good object-oriented design principles are followed; "God classes" are avoided.

Metric: Clarity and justification of the design decisions and trade-offs, rated on a scale of 1 (poor) to 5 (excellent), then converted to a percentage.

Understandability of the solution direction [Appropriateness recognizability]

- A clear and neat Class Diagram for the Fiery Dragons game.
- Correct usage of notations in the class diagram.

Metric: Ease of comprehension of the design and implementation, rated on a scale of 1 (difficult to understand) to 5 (very easy to understand), then converted to a percentage.

Extensibility of the solution direction (in anticipation of extensions to the game for Sprint 4) [Modifiability]

- Use of principles like Open-Closed principles.
- Use of interfaces or abstract classes for loose coupling and separation of concerns.
- Separation of UI and business logic.

Metric: Degree of modularity and adherence to design principles (like Open-Closed Principle), rated on a scale of 1 (low extensibility) to 5 (highly extensible), then converted to a percentage.

Quality of the written source code (e.g., coding standards, reliance on case analysis and down-casts) [Maintainability]

- The branch includes a 'README' or similar documentation that describes its organisation and details the locations of specific files.
- Adhering to coding standards like naming conventions
- Ensuring most repeated entities are inherited and polymorphed to the lowest level (abstract class/ interface)
- Reliance on very few external libraries
- Modularity of the code (Functions are appropriately sized, code is organised into packages, directories or namespaces)

Metric: Adherence to coding standards, appropriate use of abstractions, and avoidance of code smells, rated on a scale of 1 (poor maintainability) to 5 (excellent maintainability), then converted to a percentage.

Aesthetics of the user interface [User engagement]

- The game and all its elements are visible
- The visual appeal of the game (layout, colour schemes)
- The game has well-defined affordability (Know how to interact with the game)
- Appropriate feedback for all game interactions.

Metric: Visual appeal, layout, colour schemes, and clarity, rated on a scale of 1 (unappealing and hard to use) to 5 (highly appealing and easy to use).