Sprint 3 Deliverables

Table of Contents

Defining 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, rated on a scale of (poor) to 5 (excellent), code coverage percentage is converted to the rating scale out of 5.

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.
- Sound 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 the 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).

Sprint 2 Solution Reviews

Ahmad Sprint 2 Review

Functional completeness and functional correctness - 3

Most of the requirements were met, but the prototype was missing volcano card characters and cave cards were missing

Functional appropriateness - 3

- Design rationale covers all required areas
- Design patterns explained and justified not actually present in the class diagram or code.

Appropriateness recognizability - 5

- Neat and clear diagram
- Easy to comprehend

Modifiability - 2

- Very few instances of inheritance
- Not following open-close principle
- UI and business logic not separated

Maintainability - 3

- Does not include README
- Coding standards were followed, but comments were scarce
- Very few external libraries were relied upon
- Functions not very modular

User Engagement - 4

- UI is aesthetically pleasing
- UI is easy to understand and use
- But some UI elements were missing

Fahad Sprint 2 Review

Functional completeness and functional correctness

- All key functionalities are covered.

Functional appropriateness

- Some classes
- Responsibilities are not divided appropriately leading to a god class whose appearance was not justified.
- Design rationales are well thought out and encompasses everything
- Design patterns are identified and choices justified well

Appropriateness recognizability

- Class diagram is clear and easy to read
- Packages used which should not be in UML diagram
- Notations were correct

Modifiability

- Loosely coupled
- Use of principles like Open-Closed principles making it easy to extend the game functionality without breaking requiring opening of existing code.
- Use of interfaces or abstract classes for loose coupling and separation of concerns.
- Very good separation of UI and business logic.

Maintainability

- Adherence to coding standards and good use of inheritance.
- Low reliance on external libraries.
- Very modular; good use of packages

User Engagement

- All the game and all its elements are visible
- Appealing UI
- Appropriate feedback for all game interactions
- Player Turn was displayed at the bottom of the page letting the players know their turn

Hanif Sprint 2 Review

Functional completeness and functional correctness

- Implemented all critical game functionalities
- All relevant attributes, methods, relationships, and cardinalities are covered

Functional appropriateness

- Presence of God classes
- Design rationales are well thought out and encompasses everything
- Design patterns are identified and choices justified well

Appropriateness recognizability

- UML not very legible due to lack of colour coding or shading
- Packages used which should not be in UML diagrams
- Rest of the notations were correct

Modifiability

- Use of interfaces or abstract classes for loose coupling
- Dependency inversion used
- Good separation of UI and business logic

Maintainability

- Coding standards followed, but missing some docstrings

User Engagement

- Aesthetically pleasing UI
- Easy to understand and use
- All game elements visible

Umair Sprint 2 Review

Functional completeness and functional correctness

- Covers all functionalities
- Class relationships, methods, cardinalities, etc. cover all functionalities

Functional appropriateness

All design patterns were appropriately identified and used.
Justifications for the design pattern were not satisfactory.

Appropriateness recognizability

- Class diagram is clear and easy to read
- Packages used which should not be in uml diagram
- Appropriate notation not used in all places.

Modifiability

- Absence of abstractions where needed leading to a loss in the open closed-ness of the design.
- Lack of strategy pattern in appropriate locations leading to unnecessary modification of base class,
- Not very good separation of UI and business logic.

Maintainability

- Adherence to coding standards and good use of inheritance.
- But missing some docstrings
- Low reliance on external libraries.
- Missing readme and lack of use of packages.

User Engagement

- All game elements are visible.
- Moderately appealing UI.

Sprint 3: Consolidation of Ideas

For sprint 3, ideas from across the team members' designs were implemented into the new design. Key aspects that were carried over from each member's design was their implementation of certain design patterns.

The singleton pattern from Umair's design was incorporated in a fashion through the use of the EventManager class, with a single instance handling all events,

while providing a global access point to this manager. Umair’s design incorporated such managers in a variety of ways.

Sprint 3: New Ideas

Sprint 3 Solution Rationale

Functional completeness and functional correctness

Lorem ipsum dolor sit amet

Functional appropriateness

Lorem ipsum dolor sit amet

Appropriateness recognizability

Lorem ipsum dolor sit amet

Modifiability

Lorem ipsum dolor sit amet

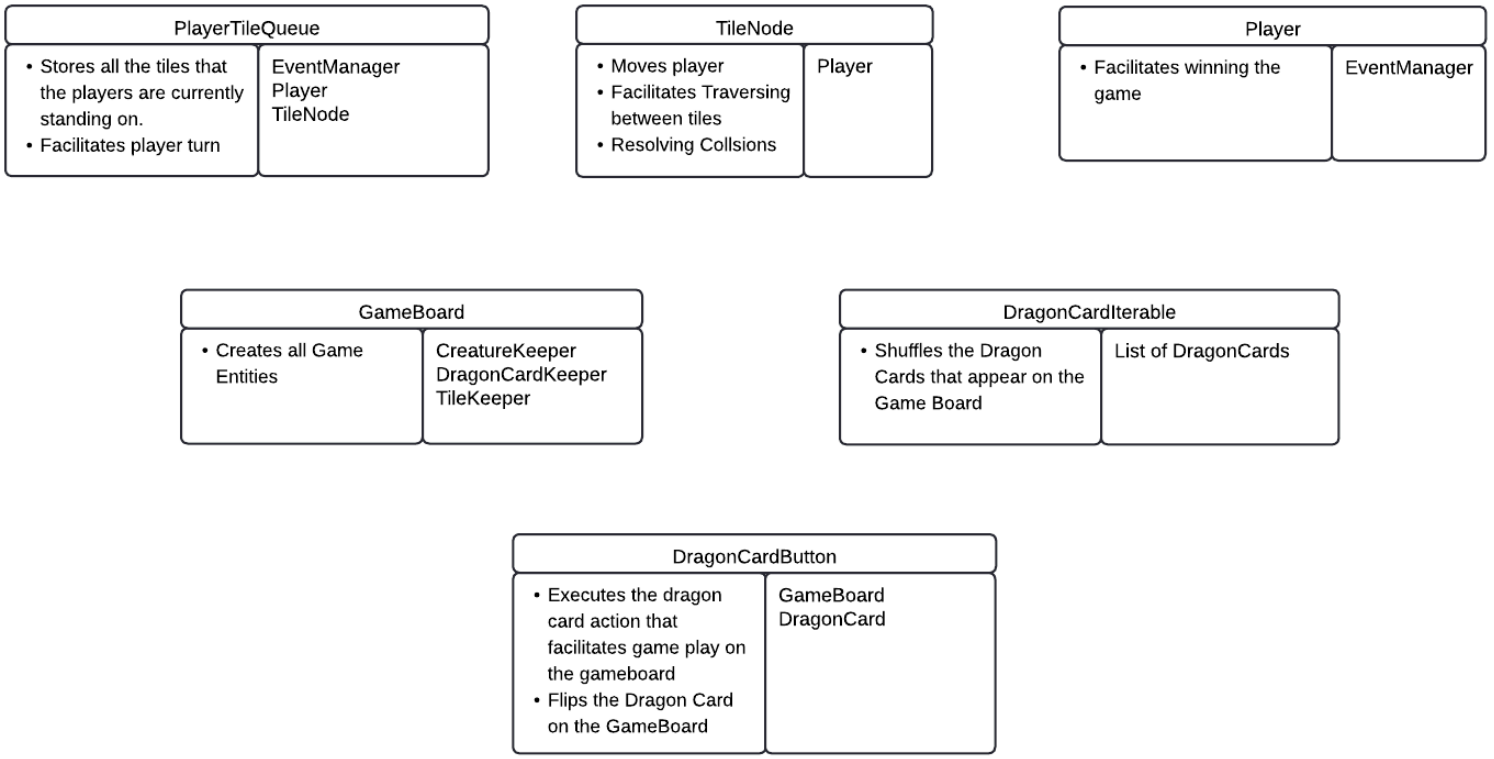
Maintainability

Lorem ipsum dolor sit amet

User Engagement

Lorem ipsum dolor sit amet

CRC



GameBoard

Acts as the main access to the backend of the game. Initialises the CreatureKeeper, DragonCardKeeper and the TileKeeper that in turn initialise all the different aspects of the game, helping create loosely coupled entities.
All major responsibilities of items under this entity are encapsulated within it.

DragonCardIterable

TileNode

The graph node class that stores different tile types like the VolcanoTile and the CaveTIle. Only knows about the tiles adjacent to it. Facilitates player movement using recursion to access certain tiles that are at steps distance away from it.

Player

The entity that represents the interface for the user to interact with the game. The player only knows about how many total moves it has traversed. Uses the total moves to facilitate the feature of winning the game.