Object Oriented Design and Design Rationales

Class Diagram

Key Game Functionalities
(Note: the term ‘player’ in this context refers to both the dragon token and the users that will play the game, meaning they represent the same thing conceptually)

Setup of Initial Game Board
This is covered by the Game and Board class. The Game class will prompt users to input settings such as number of players and board size using the fetchSettings() method. Then the Board class will generate all the tiles. The Board class will then store these tiles in an attribute. The chit cards are also randomly generated by the Board class using the help of a Random Number Generator class.

Flipping of Dragon Cards
This is primarily done by the Player class. The Player class will prompt each turn for a user to choose one of the Dragon cards to flip using the flipCard() method.

Movement of Dragon Tokens
This is handled by the Board class. After the user has chosen which dragon card to flip, the Turn and TurnHandler classes will figure out where the player will move to another tile, if movement is required. The TurnHandler abstract classes and its concrete child classes will handle turn logic including whether or not the card flipped is matching the one the player token is standing on. The Board class will use the movePlayer() method to move the player from one tile to another since all the logic will be handled elsewhere.

Change of Turn to Next Player
This is done by the Turn class. The Turn class has a list of all players in the game. Looping through each player, it will call the method nextTurn() which will then ask the user to play out a turn. Since the game rules allows players to flip multiple cards, each card flipping and consequently movement of a player token (if required) counts as one turn. If a player decides to flip multiple cards, it will count as multiple turns.

Winning the Game
This is done through the TurnHander class. The turn handler class will be able to determine whether or not a player will win the game with the card they have flipped. The class will check if the player has flipped the exact number of correct animals in that card to move it forward to exactly the cave where the player started. If the player has won, the turn class will notify the board that the game has finished and the startGame() method in Board class will return a value indicating it has finished.

Design Rationales
Classes
Two key classes are the Board and Turn class. The board is considered the canvas of the game and the GUI. Similar to playing the game in real life, the board is an overview of all the

pieces on the table. The board has the sole responsibility of moving the player which is a key feature of the Fiery Dragons game, following the Single Responsibility Principle. It is not suitable for the Board or moving of player to a single method as there are various pieces of information that is required for a player to be moved such as can the player even move to a location, where the player is located, to what location to be moved to.
The Turn class is included in the design as there must be a way to calculate the logic behind each turn. For each turn, there must be logic required to check if a player can move from one tile to another. It will also hold information on what card the player has flipped. It also holds a list of players to loop through for the game to run continuously. It is not appropriate for the Turn class to be a method as multiple prerequisites are required for a turn to run as explained above.

Relationships

Inheritance
Inheritance is only used in one specific case in the design. The Chain of Responsibility design pattern is used which indicates that inheritance is used. The inheritance allows for the Turn class to calculate the turn logic. Since there are multiple specific steps for each turn to run, each step is separated into a child class, inheriting from an abstract class called TurnHandler. This follows the Single responsibility principle, and further allows for future extensions and modifications to be made without causing errors to the entire turn system.

Cardinalities

Design patterns
The Singleton design pattern is used on the Board class in this design. This is because there will only ever be one Board class object. This is beneficial in this scenario because it allows other classes to have access to the board. Such as real life, a player is able to have access to the board and move a token at any given time and check whether or not the player is able to move their token.
The Chain of Responsibility behavioural design pattern is used in this context. This is due to the structural and ordered nature of how the game is run. The classes that utilise this design pattern are the TurnHandler class as well as its child classes.
For each turn in the Fiery Dragons game, there will be a list of various checks and steps to be completed before any updates should be made such as checking if the animal flipped matches the tiles the player is on, if the player wins the game etc. This allows for the practise of the Single Responsibility Principle as well as allowing for further extensions and features to be implemented as the game increases in complexity.

Start game
Create new turn request of sorts
Pass request into turnhandler 1 2 and 3
If all 3 turn handlers return true or whatever
Then move the player
Else

Dont move the player at all