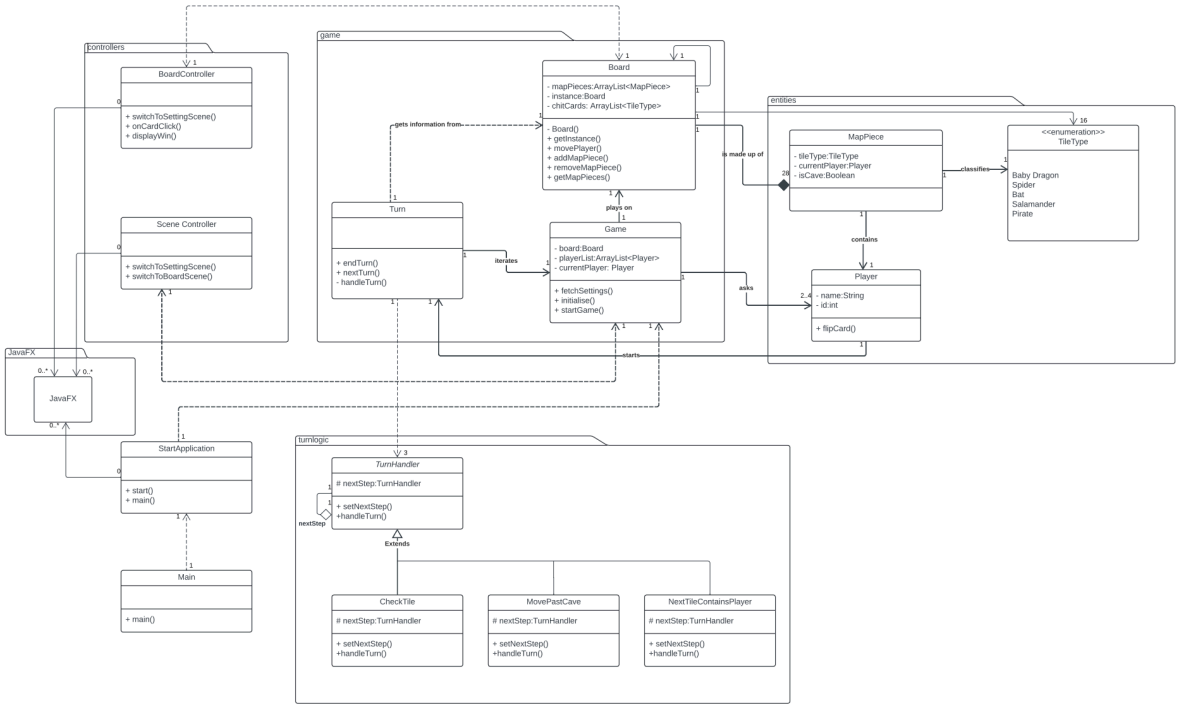
Object Oriented Design and Design Rationales

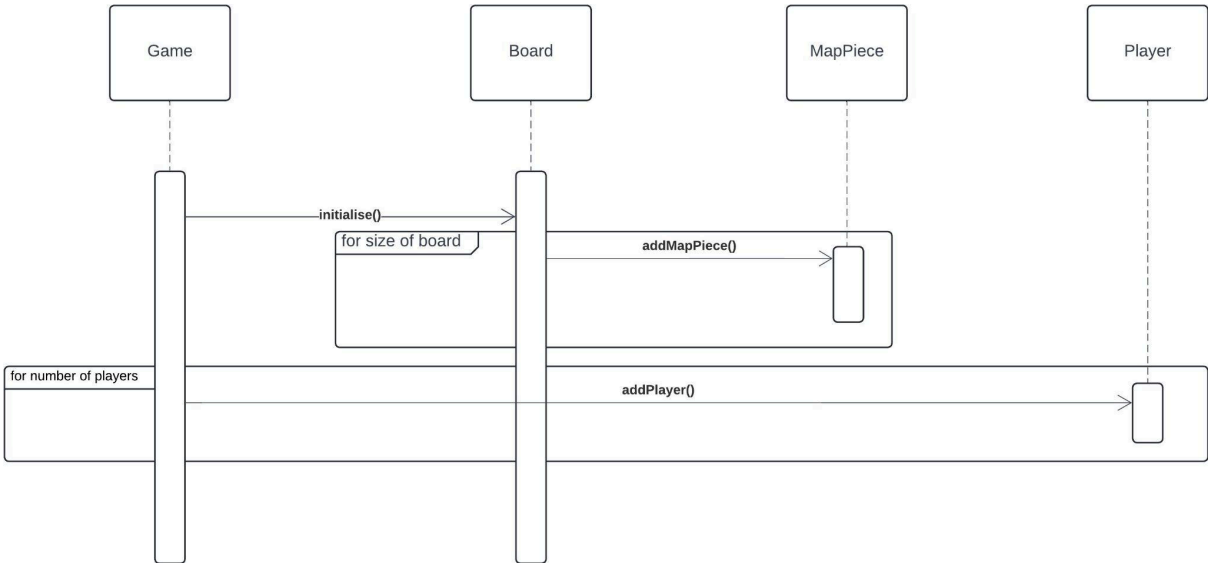
Class Diagram



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)

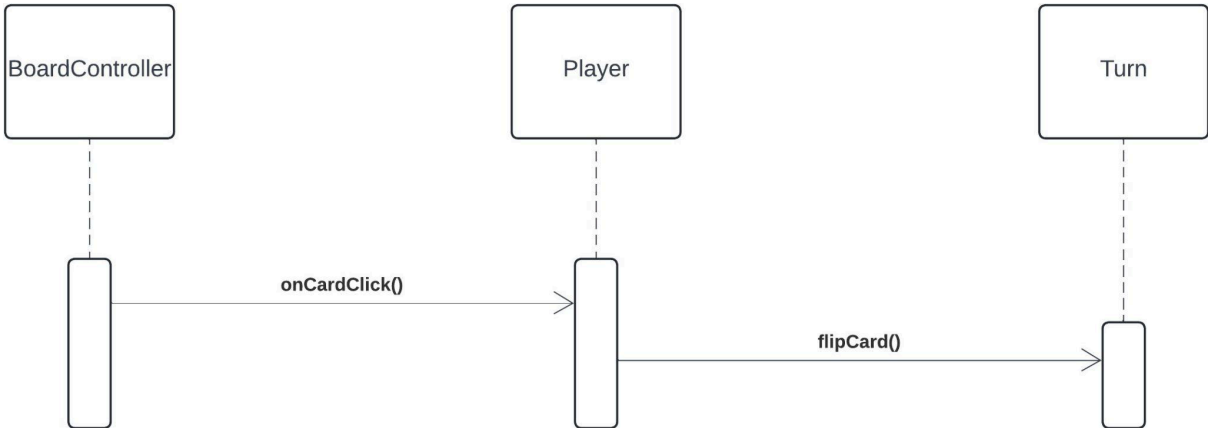
Setup of Initial Game Board



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

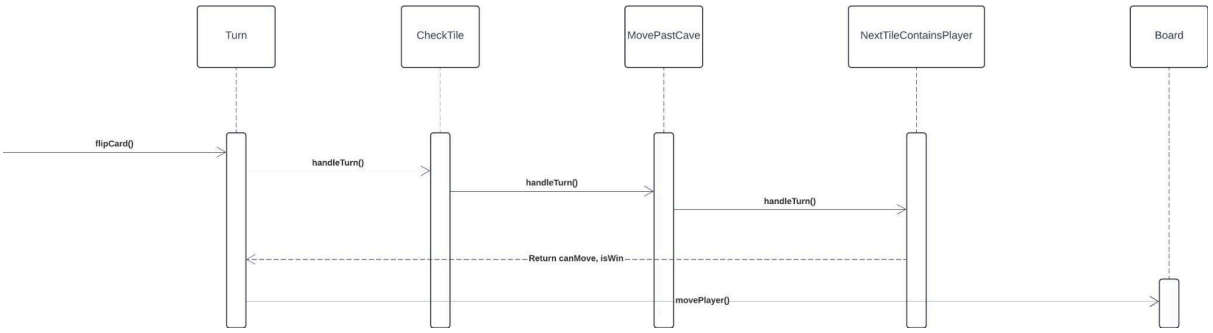
Board class will generate all the tiles. The Board class will then store these tiles in an attribute. The chit cards are also randomly generated by the Board class using some sort of random number generation.

Flipping of Dragon Cards



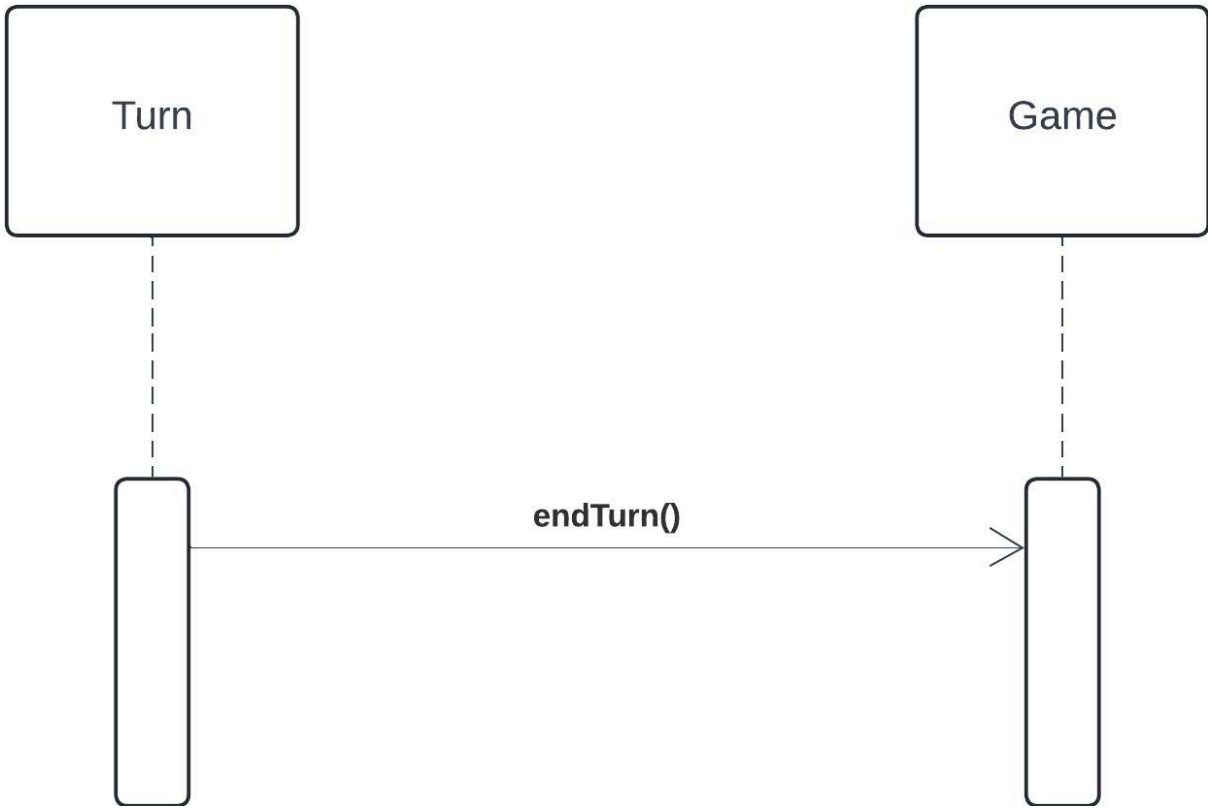
This is primarily done by the Player class. The Player class will prompt each turn for a user to choose one of the Dragon cards to flip using the flipCard() method.

Movement of Dragon Tokens



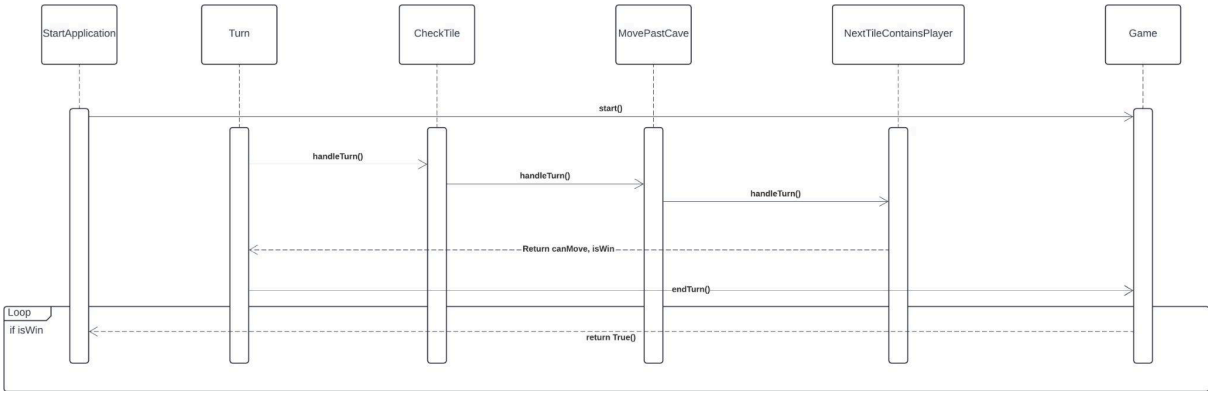
This is handled by the Board class. After the user has chosen which dragon card to flip, the Turn and TurnHandler classes will figure out where the player will move to another tile, if movement is required. The TurnHandler abstract classes and its concrete child classes will handle turn logic including 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 and Game class. The Game class has a list of all players in the game. Having an attribute stating the current player. Once the Turn class calls endTurn() method, the attribute changes to the next player. 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. The diagram does not show much interaction, but the switching of next player is a change in attribute in Game.

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 initiate the logic behind calculating a turn. Each turn will figure out whose turn it is, and what the player chose to do. This then decides what logic needs to follow such as ending the turn and or continuing flipping cards.

Relationships

A key relationship is the composition between Board and MapPiece. This is a composition because in order for the game to function correctly, MapPieces are required to build up the board. There is an association between Board and MapPiece. The MapPieces define the structure of the Board. This cannot be a composition because the association between the two classes is strong. The Board is responsible for the lifecycle of MapPiece such as when a new game starts with different board settings.

Another relationship is the dependency between Turn and Board. Since there is only one Board object at any given time, by calling Board.getInstance() returns that Board object. This allows the turns to view information of the Board at any point in time. This dependency is necessary to facilitate the calculation of turn logic. Information of the board is required to check whether or not a player is able to move somewhere, what tile the player is on and whether or not the player will win the game. Without this dependency, these calculations are much harder to implement.

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

An important cardinality is the cardinality between the Board and MapPiece. The cardinality describes a 1 to 28 relationship between Board and MapPiece. In this context, each MapPiece refers to a tile a dragon token is able to be placed in. This tile can be a cave and or a regular

tile with an animal. The 28 refers to how many tiles in a standard game per the rule book, 24 normal tiles and 4 cave tiles. This is chosen rather than having a set of 3 tiles because the set of tiles don't actually have a functionality gameplay wise. Furthermore, for each MapPiece, it can only be on a singular Board.
Another important cardinality is between Turn and TurnHandler. The 1 to 3 cardinality describes that for every Turn object, it will have dependencies to always 3 different TurnHandler objects. This is because the three different TurnHandler objects will always be called each turn to check for logic, no object will be skipped. If an object is skipped, some crucial turn logic will also be skipped, thus it is important for this cardinality to be 1 to 3.

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.