CRC Cards for Fiery Dragons

TODO: CRC cards for all of the main required classes are provided, including a sound description of the purpose of each class. [DONE]
TODO: The correct format of CRC cards used for all classes is used. [DONE]
TODO: Evidence of alternative distribution of responsibilities is provided and their rejection is discussed. [DONE]

Player

This class represents each player/token on the board. When the player is initialised, it is given a specific cave where it starts from. Each player also keeps track of consecutive pirate chits clicked. Moreover, it also keeps track if it is currently their turn or not and if it has finished the game.

Player	
Starting cave Check if current turn Move player to cave at the start Track consecutive pirate chits Check if player is finished	Cave Position

AnimalFactory

This class returns all the required instances of animals that are required to be used by the various board pieces. For each type of board piece, it accordingly creates the correct number and type of animals using the specific Animal class (which is a superclass of the five different types of animals available). For example, for the cave objects, AnimalFactory outputs two, three or four unique animals, depending on the number of players.

AnimalFactory	
Create animal instances required for the chits Create animal instances required for the caves Create animal instances required for the volcano cards	Animal

BoardFactory

This class returns all the required instances of board piece objects. Each chit, cave and volcano card have corresponding animals, which are taken from AnimalFactory. Then, according to Fiery Dragons specific, each board piece object is created using the specific BoardPiece class (which is a superclass).

BoardFactory	
Create all the instances of the chits Create all the instances of the caves Create all the instances of the volcano cards	AnimalFactory BoardPiece

Board

This class represents the whole board consisting of all the board pieces. It creates all the Player instances and uses BoardPieceFactory to get all the instances of the other board pieces. The board is represented using a 2D matrix, each element corresponding to a position on the screen. Each piece is given a position, which is calculated depending on the type of piece and how they're supposed to be randomised according to the specification. Finally, all board pieces are drawn onto the given position on the matrix.

Board	
Chits Volcano cards Caves Players Position and draw all the board pieces (including tokens) onto the screen	BoardFactory Player Position

Main

This is the main game class where pygame is initialised and the instance of the board is created using the Board class. The board then communicates with the GameFlow class to carry out the game. This class also displays a welcome screen where the user can input the number of players.

Main	
Display welcome screen Number of players Initialize board Run game flow	Board GameFlow

GameFlow

This class controls the game flow and implements the template and iterator design pattern. Its primary function is to ensure the game loop runs according to the required functionality. It checks which player is playing currently and listens for clicks on chits, and accordingly moves the player. After the current player's turn is done, the game flow changes the current player. Finally, when a player has won, GameFlow displays the 'win game' screen and terminates.

GameFlow	
Current player Board Listen for chit clicks Flips/unflips chits accordingly Moves player on the board Updates board in game loop Change player turn End/win game	Board Player Position

Alternative Distribution of Responsibilities

The team considered combining BoardPieceFactory and AnimalFactory into one class 'Factory', which shares both sets of responsibilities. However, to ensure the code follows the single-responsibility principle, two separate classes were used. Also, BoardPieceFactory depends on AnimalFactory, and having this dependency defined explicitly allows for better understandability and modifiability (suppose we need to add an extra animal for Sprint 4).

Currently, the players are initialised in the Board class. An alternative considered was initialising them in GameFlow since the logic for player turn and game state is defined here. However, since Board is responsible for drawing itself (including players) onto the display and Player has an association with the positions on the board, it is much easier to implement to create players in Board, hence resulting in encapsulation.

Initially, flipping chits and moving players were implemented in the Board class. However, placing all these methods in the same class makes it over-complex. This causes the code not following many notable software qualities, particularly maintainability and flexibility. Therefore, we decided that all chit flipping functionalities will be implemented in gameflow.