FIT3077 Sprint 2

Name: Vansh Batas, Student ID: 32196741

1. Summary of Key Functionalities

• Initial Game Board Setup:

- The GameBoard class is responsible for setting up the initial game board.
- In the __init__ method, the GameBoard class creates a circular surface to represent the game board, loads the arena image, and creates a grid of random animal images.
- The create_volcano_cards method generates the volcano cards, each with a randomly chosen animal image, and positions them around the circular game board.
- The create_dragon_cards method creates the deck of dragon cards, which contain the animals and pirates.
- The place_dragons_and_caves method places the dragon tokens and caves at the center of the game board.

• Flipping of Dragon Cards:

- The handle_input method in the GameBoard class handles user input, specifically mouse clicks on the game board.
- When a user clicks on a grid square, the method toggles the flipped_card attribute between None, the currently flipped card, and the newly clicked card.
- The update method then renders the game board, drawing the flipped card image on the grid square.
- The actual logic for the effects of flipping a dragon card (e.g., moving the dragon tokens) is not implemented in this code, but it would likely be handled in the play_turn method or a separate method that processes the drawn dragon card.

2. Rationale

GameBoard Class

The core of the game's logic is the GameBoard class, which is in charge of establishing the starting game state, processing user input, modifying the game state, and managing how the many game elements—dragons, caverns, volcanic cards, and dragon cards—interact with one another.

This functionality is contained in a different class, which improves the code's modularity, maintainability, and testability. The majority of modifications to the game's mechanics or rules can be altered within the GameBoard class, with little to no effect on the other game elements.
Furthermore, since the GameBoard class can serve as a central interface for engaging with the game, keeping the game board management logic in its own class facilitates future reuse or expansion of the game.