Assessment Criteria

- Completeness of the solution direction
    - Functional completeness
    - Functional correctness
    - Faultlessness
- Rationale behind the chosen solution direction
    - Functional appropriateness
- Understandability of the solution direction
    - Appropriateness Recognizability
    - Learnability
    - Operability
- Extensibility of the solution direction
    - Modifiability
    - Reusability
- Aesthetics of the UI
    - User Engagement

Shen's:

- Completeness of the solution direction
    - Functional completeness:
        - The majority of the functionalities are completed. The game is missing the dragon pirate chit cards. However, everything else has been successfully set up, including the set up of the board (initialisation of dragon tokens, tiles and chit cards) and the flipping of chit cards works seamlessly.
    - Functional correctness
        - The game board is set up correctly with the correct cave/normal tiles. Randomisation of chit cards and tiles works correctly. Relative to the game rules, the game board is still missing four dragon pirate chit cards. The game board does not provide accurate results when used by the users since they won't be able to pick a dragon pirate chit card.
    - Faultlessness
        - It randomises chit cards and tiles correctly without causing the chit cards to have the same coordinates which would then result in an overlap of the chit cards.
- Rationale behind the chosen solution direction
    - Functional appropriateness
        - [later] The game carries out the correct functionality with appropriate methods. For instance, Shen created a box with a safe area to ensure the chit cards do not overlap
- Understandability of the solution direction
    - Appropriateness Recognisability
        - It has all the functionalities that satisfy the user's needs. It's highly distinguishable between the animals on the normal tiles, chit cards area and the cave tiles with their associated animal. Chit cards are all spaced out. However, the dragon tokens were a bit small in size and

different in colour. The back of the chit cards are similar to the dragon tokens which makes it hard to differentiate between them.

    - Learnability
        - Players can easily pick up the game based on reading the rules of the game. However, developers who maintain the code might find it a bit harder to understand because of the number of classes interacting with each other though there are documentation and comments throughout the code.
    - Operability
        - The game is highly operable. For instance, we can add more chit cards and more tiles on the game board, and everything still works perfectly.
- Extensibility of the solution direction
    - Modifiability
        - You can adjust the cave tile to be at any position on the board. You can also adjust the dragon token's position to start at their cave or on normal tiles. You can adjust the board size as well to make it larger or smaller.
    - Reusability
        - All methods are highly reusable. Users can draw additional chit cards or tiles using the same method(s). For instance, *get_draw_clickable_assets_instructions()* under the class *ModularClickableSprite* tells you how to draw a chit card and tiles by returning the drawing instruction and the object to click itself.
- Aesthetics of the UI
    - User Engagement
        - The design is very minimalistic and easy to navigate.

Rohan's

- Completeness of the solution direction
    - Functional completeness
        - All required functionalities are completed. The initial setup of the board is correct and the flipping of chit cards works.
    - Functional correctness
        - The game board is set up correctly with the right amount of chit cards, dragon tokens and tiles. There are 4 types of chit cards present on the game board as required. The game board provides accurate results as users can pick and flip any chit cards and won't be able to flip an already flipped chit card. The randomisation of chit cards and volcano tiles works accurately with no overlaps.
    - Faultlessness
        - There is a minor aesthetic fault when flipping the chit cards, there is a black border around the chit cards when flipped. Every other method works perfectly with no faults.

- Rationale behind the chosen solution direction
    - Functional appropriateness

        - Functions each perform their designated functions (creating board, drawing chit cards, draw game, drawing the tiles, shuffling the chit cards). There are not many interdependencies between the functions in the class, and they operate singly on the state of the class.

- Understandability of the solution direction
    - Appropriateness Recognizability
        - Very obvious game board, dragons and chit cards. It's easy to see which component is linked to the game rules as the pictures used make it easy to relate. However, the dragons cover the animals on the tiles which may make it hard to match chit cards if the user forgets their animal on their tile.

    - Learnability
        - Comments may need to be summarised as they are a bit long. Implementation itself is easier to locate and understand as all related details are grouped in the appropriate classes (e.g game_board.py), but the class itself is lacking type hinting which makes modifying the state more difficult. The grid system used to draw the board and its chit cards are easy to understand.

    - Operability
        - A bit difficult to modify the constants required to change the characteristics of the game board, and the players. The game configuration file contains many constants that require an innate understanding of the class it is used for before it can be modified (i.e for increasing number of tiles on the game board or modifying the sequence of volcano cards on the board)

- Extensibility of the solution direction
    - Modifiability
        - It is slightly modifiable. The constants in the configuration file allow the sequences of volcano tiles to be modified. All functionalities are located in one or two classes (e.g god class game_board.py). Makes it hard to add new/modify the type of chit cards or dragons without having to modify existing code. The god class is dependent on concrete classes (i.e AnimalChitCard, CaveTile, PirateChitCard, etc.), making changing the assets used for each of the classes hard.

    - Reusability
        - Not reusable. Classes such as game_board.py depend on concrete classes such as AnimalChitCard, and can't be interchanged with any other chit card without potentially causing issues. The game board also depends on the NonCaveTile and CaveTile concrete classes. Overall, there is no use of interfaces or abstract classes, which does not allow code to be reusable (i.e code needs to be modified in game_board.py if you want to change the type of chit card or tiles)

- Aesthetics of the UI
    - User Engagement

Ian's

- Completeness of the solution direction
    - Functional completeness
        - All functionality has been completed including setting up the game board(creating tiles, caves, chitcards, and dragon token, randomise the chitcard position) and flipping chitcard
    - Functional correctness
        - The gameboard is set up correctly with original game functionality(4 caves, 4 dragon tokens, 16 chitcards and 24 tiles). Every time the game is initialised, chitcards and tiles are initialised with different positions and each chitcards and tile do not overlap with each other.
    - Faultlessness
        - The chitcards can be flipped even though we clicked the position not within the range of the chitcards image but near the chitcards which can make users confused which chitcards they are currently selecting.
- Rationale behind the chosen solution direction
    - Functional appropriateness
        - Each function does exactly the same as the purpose of it such as default_init() which is creating all the components on the board and randomise_animal_tiles() and randomise_chit_card_position for doing randomise the position of animal tiles and chitcards. But for draw_chit_cards() in Board class, the method is checking if the chitcards is flipped everytime the Board class is drawing it. We can separate the code of checking out of the method. Therefore, the Board class can draw every item without concerning the state of each entity to reduce the interdependency of the Board class with other entities.

- Understandability of the solution direction
    - Appropriateness Recognizability
        - The game has been implemented corresponding to the need of the game base functionality. Everything on the board can be distinguished obviously. Positions of all components are positioned accurately without making the game hard to play. However, the dragon token is positioned above the position of the cave which makes it hard to observe which animal is in each corresponding cave.
    - Learnability
        - The way of implementation of game set up and functionality make users easily play it if they know the rules of fiery dragon. The docstring of code and comment make the other developers easier to understand what each line of code is doing. The game screen is designated with a grid which makes each class can be positioned based on the row and column without concerning the exact coordinates on the screen.
    - Operability
        - The game is harder to control if the system needs to add player, chitcards, tiles because everything about setting up the gameboard is

hardcoded in game class and board class. For example, if we need add more chitcards in the game, we need to append the chitcards list in Board class and need to reconsidering the position of each chitcards again to avoid overlapping)

- Extensibility of the solution direction
    - Modifiability
        - The code is hard to modify since the number of chitcards and the position of chitcards and tiles are hardcoded in the Board init method and Game class init method which make the both class as a potential god class. The advantage of the code implementation is we can edit the screen width and height, the row and column on the screen in the BoardConfig file easily.

    - Reusability
        - It is hard to reuse especially for creating tiles and chitcards since the game cannot change the number and position of tiles and chitcards due to the hardcoding. A module can be reused if we can use the module without concerning the implementation of the module. However, we need to change the code of the module of creating tiles and chitcards to address any changes in the game.

- Aesthetics of the UI
    - The game ui is simple and minimalistic and the size of each component is designed nicely. However, the image can be loaded for an unflipped chitcard instead of a plain black image.

Desmond's

- Completeness of the solution direction
    - Functional completeness
        - All the initial required functionalities have been implemented and work correctly. The initial setup of the board is correct and the chit cards can be flipped when they are clicked on.
        - Faultlessness
            - There is a minor aesthetic fault when flipping the chit cards, there is a black border around the chit cards when flipped. Every other method works perfectly with no faults.
        -
    - Functional correctness
        - The game board is set up correctly at initial start up with the correct amount of cards, dragon tokens (depending on how many players were selected at the start screen) and Tiles. All 4 chit cards are displayed when flipped. The game board flips the correct chit cards when clicked on. However, the chit cards can be flipped back if they are already flipped. We should not be able to flip already flipped chit cards backed to their unflipped position when clicked on. The

randomisation of the chit cards and tiles work correctly with no overlapping tiles or chit cards when randomised each game.

- Faultlessness
    - There is a minor fault where users are able to unflip chit cards they have already flipped by clicking on them again. Chit cards should only be unflipped when the current active player's turn has ended. All other methods work perfectly with no fault's.

- Rationale behind the chosen solution direction
    - Functional appropriateness
        - Each function carries out its dedicated functionality correctly. Each function carries out its own task. There are only a few interdependencies between functions in the class.
- Understandability of the solution direction
    - Appropriateness Recognizability
        - Very clear and distinct game board, dragons and chit cards. Assets are very engaging and easy to distinguish. Clear which component is linked to the game rules as the assets are very easy to distinguish and relate.
    - Learnability
        - The comments are succinct and easy to understand. Each function does only one simple job without too much work being done in one function. This makes it easy to understand which function is doing what which allows us to implement each function correctly later on. The way the tiles and chit cards are drawn is easy to understand.
    - Operability
        - It is a bit difficult to modify the number of certain elements that make up the game board. The game configuration file contains quite a few constants that require the users understanding how each of the classes work to modify the constants.

- Extensibility of the solution direction
    - Modifiability
        - It is decently modifiable. The constants in the configuration file allow the sequence of the volcano tiles to be randomised by shuffling the positions list. We can also append more coordinates to extend the game board. The user has avoided too much functionality in one class allowing for modifiability and extensibility of future game implementations.

    - Reusability
        - The code is partially reusable. The class game board depends on concrete classes such as ForwardChitCard and BackwardChit card which means they can't be interchanged without potentially causing any additional issues. However, there is a drawable interface which means what is rendered can be easily changed based on new classes implementing the drawable interface.

- Aesthetics of the UI

- The game UI is very aesthetically pleasing with a warm, inviting look to it. Users are also not just thrown straight into the game but are able to select the number of players playing the game.