Team Evaluation

In this section, a scale ranging from 1 to 5 is used to assess our work. Scores reflect the extent of omissions detected (the present stage of the design is not workable and is not complete), from major omissions found (the design may be more feasible than a design with a rating of 1, but it still requires a lot of work to remedy these significant shortcomings), many minor omissions (the design seems sound in general, there are a lot of nuances that need to be fixed), minor omissions (the design is nearing completion, so with little modification it should work as intended) to no omissions (the design seems to be finished and prepared for use) detected.

1. Game Initialisation

	Teoh You Xian	Lim Jian Tao	Hang Jui Kai	Lee Yi Mei
Completeness	**Score: 5** ● It sets up the game board with correct number of chit cards, squares, caves and tokens base on the number of players selected ● Player dashboard is displayed with the number of steps left for each player to win ● Consist of timer and the current player who needs to make a move	**Score: 5** ● Able to randomly initialise animals on the game board without error ● Number of tokens is generated based on the player count ● Each token is assigned to a specific cave	**Score: 5** ● It can allocate a certain square and cave to each volcano card according to the instructions, considering that the cards may organise at random ● It enables generating the correct amount of Chit Card that is specified in the instruction ● It will initialise the token with the number of players who are provided	**Score: 5** ● Able to randomly assign the animal Squares, Chit Cards and Caves accurately ● Able to assign player's token correctly to the cave with different numbers of players ● Each token is assigned to a specific cave

Correctness	**Score: 3** ● Number of steps for each player to win is set correctly when the game starts ● Chit card is randomly assign at the beginning of each game ● squares in the game board is not initialised randomly	**Score: 5** ● The volcano cards are correctly randomised according to the requirements ● The caves are generated in the correct spot when there are 2 to 4 players	**Score: 4** ● Each cave will be allocated to the specific square while generating the square ● It will accurately initialise the number of Chit Cards, Square, and Cave ● The token will be associated with a certain Cave pursuant to their unique identifier	**Score: 4** ● The number of Chit Cards, Squares and Caves amount is generated correctly ● The token is able to locate the correct Cave based on the input number of players ● The gameboard is able to be generated correctly based on the information given ● The randomization of Cave might be incorrect
Code Quality	**Score: 3** ● The code follows JavaFX conventions and separates concerns appropriately. ● However, there are opportunities for improvement in terms of	**Score: 3** ● Each component has its own dedicated class along with their own responsibility ● Builder design pattern is used but not applied on	**Score: 3** ● There are certain parts of initialising operations that consist of redundant code ● Certain design patterns have been used in	**Score: 3** ● The ability of Turn Manager could be improved as it's acting a god class now ● The creation of Chit Cards is too specific that

	code organisation and readability. ● The naming of some variables and methods could be more descriptive, and there is room for better error handling.	all core game components ● A lot of repeating codes when creating chit cards ● The game logic and the user interface code is mixed together, which violates the single responsibility principle	specific processes on initialising the game ● The code on handling user interface and logic have been compiled together, which do not adhere to the single responsibility principle	might affect future modification for some special cases such as chit cards with numbers of 3, 5, 6 ● Game Logic and the user interface code is mixed together, this violates the single responsibility principle

Extendability & Maintainability	**Score: 4** ● The use of Builder pattern for the creation of game board components, allowing for easy addition or modification of component types without altering existing code.	**Score: 4** ● Implementing animal as classes offers great extendability ● Volcano creation perfectly follows the open-closed principle ● UI related sections are bad for extensibility as everything is tightly coupled and hard to maintain	**Score: 4** ● The implementation of chit card and square has been utilised with abstract factory method which promise future ease of improvement ● There isn't a magic number associated with the implementation that enables the adaptability of altering the initialization rule	**Score: 4** ● The use of Abstract Factory Method allows the creation of Chit Cards and Squares to ease the extensibility of new types of Chit Cards and Squares ● The use of Turn Manager and Turn Resettables is able to easily control each Turns requirement
Usability	**Score: 4** ● The entire game board, chit card, tokens is clearly displayed ● home button and chit card is responsive	**Score: 4** ● The entire game board, chit cards and tokens are displayed neatly ● Each of the buttons are responsive, with clear indications signifying that it is a button	**Score: 4** ● The components of game board which include the square, cave, chit card and token will be precisely displayed	**Score: 5** ● The Animal's picture might be a little bit unclear ● The entire game boards, chit cards and token are displayed neatly

		● The entire game board is colourless		● Each button is responsive and nice looking ● Overall it is colourful, each animal, Caves and Token has its own colour

2. Key game functionality

	Teoh You Xian	Lim Jian Tao	Hang Jui Kai	Lee Yi Mei
Functionality Chosen	Skipping turn	Flip chit cards	Winning the game	Winning the game
Completeness	**Score: 3** ● Able to switch player after 12 seconds, or after a chit card is being flipped ● Skip turn functionality is not available after player select a chit card that matches their current square position	**Score: 5** ● Able to randomly initialise the type and the count of the animals on the chit card ● The chit cards can be flipped ● The chit cards can be unflipped after each players' turn	**Score: 5** ● Once the token was moved into the cave, examined its entire square movement to ensure the requirement was met ● The winning page will show up whenever the token is won	**Score: 5** ● Able to return to the token into the cave that it is belonged when the displacement required is met ● Able to show the winning page
Correctness	**Score: 4** ● Able to switch to the correct player in the next turn ● The player will be switch	**Score: 5** ● The chit cards are correctly randomised according to the requirements	**Score: 4** ● Upon determining the token's total movement, if it exceeds 26 steps, the token will be placed	**Score: 5** ● The token will move back to its belonging cave when the number of displacement is met the

	automatically after 12 second even though they did not make a move	• The chit cards show the correct animal type and animal count when flipped • All of the flipped chit cards are correctly unflipped when their turn is over	inside the cave; if not, the cave will advance to the next square • The winning page will display the winner of the game and the remaining steps to win for each player	game winning condition • The token will not win the game if the total displacement is not met • The winning page is able to show the correct winner of the game
Code Quality	**Score: 3** • The PlayerTurnManager adheres well to SRP as it focuses solely on managing player turns • However, the GameViewController manages multiple responsibilities including UI interactions, game initialization, and game state management. This breaches the single responsibility principle.	**Score: 2** • Flip action related code is mixed with other functionalities, which doesn't obey the single responsibility principle • There isn't a middle class that manages the player's authorisation for which actions they are allowed to perform	**Score: 3** • The move action method might not execute properly, currently only able to move forward. • Methods to indicate the winner of the game might not be designed in a proper way	**Score: 3** • moveToken() is now one of the method that is included within Square which might be required to improve to met the single responsible requirement • Token will be in charge with the displacement done, that might violates the single responsibility principle

Extendability & Maintainability	**Score: 4** ● Player Turn Manager is designed in a way that partly supports extensibility, especially if a strategy pattern were adopted to manage different turn-taking strategies, such as random or sequential turns. This would allow adding new turn strategies without altering existing code. ● The game controller class is less extensible due to its tightly coupled nature. The method of initialising and managing game elements is rigid and specific to the current game setup	**Score: 2** ● There is no class dedicated for the flip action, which is bad for extensibility as it violates the open- closed principle ● Flip action related code is also hard to maintain as it merges with other code functionality	**Score: 4** ● The method to indicate the game winner is not extensible and requires some modification to improve the code to adhere to the open closed principle ● Observer design pattern has been utilised in the game, which enables on notifying multiple objects about any events that happen to the object they're observing	**Score: 4** ● The move token method is not extendable ● The turn manager will allows extra extendable for the game winning
Usability	**Score: 4** ● Timer is shown clearly on the top right with how many seconds left before	**Score: 4** ● Each of the buttons are responsive, with clear indications signifying	**Score: 4** ● The player that wins will be shown on the winning page, along with	**Score: 4** ● The winning page is able to show the correct player's name and

	the game switch to the next player ● The current player is clearly shown below the timer ● The next player smoothly switches without any delay and the timer resets.	that it is a button. ● Players can easily tell the difference between flipped and unflipped chit cards. ● There should be a clearer indicator for buttons that cannot be clicked for more intuitive user experience.	information about each player such as their name and the amount of steps left to complete the game ● Whenever the game is over, a restart or return to homepage button will appear	token that win the game ● Lacking of other players' information and game information ● Button to replay the game or return main screen is included ● The UI design is more interactive and nice to look at

Assessment Outcome

For the initial game board creation, we decided to use 2 of the design patterns that we have used in sprint 2: the builder design pattern and the factory method design pattern. These design pattern choices provided a clever approach to creating reusable code components, crucial for generating multiple instances of the same elements efficiently. Additionally, refactoring the game board creation segment of the code into separate classes enables seamless modifications whenever necessary, which allows better maintainability and extensibility.

One significant change related to the game board is the Square class and Cave class. In this sprint, we have taken a different approach compared to sprint 2. Upon reviewing our code, we noticed many similarities between the Square class and Cave class. Therefore, we created a parent class called Tile and made both Square and Cave inherit from it. This introduces polymorphism and helps reduce redundant code.

Additionally, classes such as Flip Card Action and Skip Action need to be implemented, as these actions can be executed by the player. Therefore, in order to follow the single responsibility principle, the tasks of flipping a chit card and skipping a turn should be separated from the player class. This approach aligns with the approach that we came up with during the previous sprint.

Since each of us have different approaches on how to store the game board, our ideas for moving the tokens also vary. After discussing the pros and cons of each approach, we decided to use the linked list data structure to store our game board. Each of the tiles have an instance of the next and previous tile, allowing direct access to the next or previous tile without worrying about its actual location. To fully align the functionality of movement with the game rules, further modifications are required to check the conditions of movement.

In addition to the main game functionality, we also integrated the timer component into our game as part of the game loop. For this, we adopted a method from one of our teammates used in the previous sprint. Instead of a standard game loop with a win condition check, we made the timer function as the game loop. This ensures the game progresses even without user input for a certain period. Implementing this feature early doesn't immediately enhance code quality, but it prevents the need to rework the entire game loop later on.

Each member of our team is not satisfied with the outcomes from sprint 2 regarding the rendering of physical game components in the user interface. Consequently, we collectively agreed to devise an improved approach in the current sprint. Our solution involves dividing the rendering code into a distinct class and employing the observer design pattern to efficiently update the necessary components. This action aligns with the single responsibility principle, as it involves segregating tasks into distinct classes, therefore enhancing maintainability in the process.

Class Responsibility Card

Game Board	
Initialise all the Chit Cards	ChitCardFactory
Initialise all the Volcano Cards	Animal, VolcanoBuilder, VolcanoDirector
Assign each player to their respective cave	Players, Token
Get all the Volcano Cards	Volcano
Get all the Chit Cards	ChitCard

Tile	
Knows other tiles (front and back tile)	Tile
Move token	Token
Get token	Token
Set token	Token
Knows its animal	Animal
Knows if it is occupied	Token

Token	
Knows its tile	Tile
Knows its total displacement	
Increment / Decrement displacement done	
Set tile	Tile

Turn Manager	
Get the current player who will make the move	Player
Prepare for the next round	ChitCard, Player
Add the chit card that is being flipped in the current round	ChitCard
Know all the chit card that will remain open in the current round	ChitCard

Chit Card	
Knows its animal	Animal
Knows its number of animal	
Knows if it is flipped	
Flip itself	

Game UI	
Initial render when the game first started	Volcano, AnimalType
Keep track of subscribers	SubscriberType, Subscriber
Keep track of timer	
Render chit cards when flipped	
Render tokens when moved	

Instructions on how to run the executable

To execute the executable deliverable, open terminal and run the following command:

java --module-path "the location of java fx sdk\javafx-sdk-22\lib" --add-modules javafx.controls,javafx.fxml -jar "location of my work downloaded \MA_Tuesday12pm_Team999\Project\Sprint 3\FieryDragon\out\artifacts\FieryDragon_jar\FieryDragon.jar"

The green highlighted is to be replaced by the path that is specified to the device that is running this.

volcano

VolcanoDirector

+ constructVolcano(VolcanoBuilder builder, Animal... animals)
+ constructCaveVolcano(VolcanoBuilder builder, Animal caveAnimal, Animal... animals)

VolcanoBuilder

- volcano: Volcano

+ addSquare(Animal animal)
+ setCave(Animal animal)
+ buildVolcanoCard()
+ resetBuilder()

Volcano

- squares: ArrayList<Square>
- cave: Cave

+ Volcano()
+ getSquares()
+ addSquare(Square square)
+ getSquaresNumber()
+ getCave()
+ setCave(Cave cave)
+ hasCave()

tile

Square

- cave: Cave

+ Square(Animal animal)
+ setCave(Cave cave)
+ getCave()

Cave

+ Cave(Animal animal)

<<abstract>>
Tile

- frontTile: Tile
- backTile: Tile
- token: Token
- animal: Animal

+ Tile(Animal animal)
+ setFrontTile(Tile tile)
+ setBackTile(Tile tile)
+ getFrontTile()
+ getBackTile()
+ getAnimal()
+ isOccupied()
+ getToken()
+ setToken(Token token)
+ moveTokenForward()
+ moveTokenBackward()

GameBoard

- chitCards: ArrayList<ChitCard>
- volcanoCards: ArrayList<Volcano>
- caveVolcanoDistance: int

+ GameBoard()
+ getVolcanoCards()
+ getChitCards()
- initialiseChitCards()
- initialiseVolcanoCards()
+ assignPlayersCave(ArrayList<Player> players)
+ createChitCard(ChitCardFactory chitCardFactory, int set, int... count)

Game

- players: ArrayList<Player>
- gameBoard: GameBoard

+ Game()
+ createPlayers(int numOfPlayers)
+ getPlayers()
+ getGameBoard()

Player

- playerID: int
- name: String
- playerToken: Token

+ Player(String name, int playerID)
+ getName()
+ getPlayerID()
+ getPlayerToken()

Token

- tile: Tile
- playerID: int
- totalDisplacement: int
- lastTotalDisplacement: int

+ Token(int playerID)
+ displacementDone(int displacement)
+ getTotalDisplacement()
+ getLastTotalDisplacement()
+ getTile()
+ setTile(Tile tile)

TurnManager

- instance: TurnManager
- turn: int
- players: ArrayList<Player>
- flippedChitCards: ArrayList<ChitCard>

- TurnManager(ArrayList<Player> players)
+ getInstance()
+ getInstance(ArrayList<Player> players)
+ getTurn()
+ nextTurn()
+ getCurrentPlayer()
+ getAllPlayers()
+ addFlippedChitCard(ChitCard chitCard)
+ getFlippedChitCards()

action

FlipChitCardAction

SkipTurnAction

<<abstract>>
Action

+ execute(Player player, ChitCard chitCard)

controller

GameController

- squareAnimalImageViews: ImageView[]
- squareTokenImageViews: ImageView[]
- chitCardImageViews: ImageView[]
- caveAnimalImageViews: ImageView[]
- caveTokenImageViews: ImageView[]
- gameTimeline: Timeline
- timer: Timeline
- time: int
- game: Game
- gameUI: GameUI
- subscribers: ArrayList<Subscriber>

+ GameController()
- subscribe(Subscriber subscriber)
- unsubscribe(Subscriber subscriber)
- notifySubscribe(ImageView objectImageView, Image objectImage)
+ run(int numOfPlayers)
+ chitCardOnClick(ImageView currentChitCard, Image chitCardFront, ChitCard chitCardObject)

GameUI

- squareAnimalImageViews: ImageView[]
- squareTokenImageViews: ImageView[]
- chitCardImageViews: ImageView[]
- caveAnimalImageViews: ImageView[]
- caveTokenImageViews: ImageView[]

+ GameUI(ImageView[] squareAnimalImageViews, ImageView[] squareTokenImageViews, ImageView[] chitCardImageViews, ImageView[] caveAnimalImageViews, ImageView[] caveTokenImageViews)
+ initialRender(ArrayList<Volcano> volcanoes, int numOfPlayers)
+ setCaveTokenVisible(ImageView caveTokenImageView)
+ setCaveTokenInvisible(ImageView caveTokenImageView)

subscriber

ChitCardSubscriber

- flippedChitCards: ArrayList<ImageView>

+ ChitCardSubscriber()

MovementSubscriber

<<interface>>
Subscriber

+ update(ImageView objectImageView, Image objectImage)

<<enumeration>>
SubscriberType

animal

BabyDragon

+ BabyDragon()

Bat

+ Bat()

DragonPirate

+ DragonPirate()

Salamander

+ Salamander()

Spider

+ Spider()

<<abstract>>
Animal

- animalType: AnimalType

+ visit(Animal animal)
+ getAnimalType()
+ setAnimalType(AnimalType animalType)

<<enumeration>>
AnimalType

chitcard

ChitCard

- animal: Animal
- amountOfAnimal: int
- flipped: boolean

+ ChitCard(Animal animal, int amountOfAnimal)
+ getFlipped()
+ setFlipped(boolean flipped)
+ getAnimal()
+ getAmountOfAnimal()

BabyDragonChitCardFactory

BatChitCardFactory

DragonPirateChitCardFactory

SalamanderChitCardFactory

SpiderChitCardFactory

<<interface>>
ChitCardFactory

createChitCard(int amount)