FIT3077 - Architecture and Design

Sprint Three

MA_Tuesday08am_Team123

Swift Crafters

Team Members:

Maliha Tariq

Loo Li Shen

Khor Jia Wynn

Dizhen Liang

MONASH University

Table of Contents

MONASH University

1. Assessment Evaluation Criteria

(a) Interval-based measurement scale

The following criteria are adapted from the ISO/IEC 20510 quality model to assess each team member's design and implementation.

Factor	Characteristic	Design Principle
Functional Suitability The degree to which the design/implementation provides functions that meet stated and implied game features	Functional completeness	Completely covers the key functionalities
	Functional correctness	The implementation provides accurate results when used
	Functional appropriateness	Suitability of the approach to facilitate functional completeness and correctness
Interaction Capability The degree to which the design can be easily interacted with to exchange information; and the usability of the implementation	Appropriateness recognizability	Easy to understand solution direction
	User Engagement	Aesthetically pleasing user interface
Maintainability The degree of effectiveness and efficiency with which the implementation can be modified to improve it, correct it or adapt it to changes in environment, and requirements.	Coding Standards	Quality of source code (e.g. practising modularity, not over-reliant on case analysis and down-casts)
	Documentation Quality	Appropriate documentation of the more complex software components, clarity of documentation
	Error Handling	Logging is provided for debugging errors, or errors are prevented altogether.
	Code Readability	Good naming conventions, well-structured and easy to understand without having to consult the code author
	Modifiability	Design can be efficiently modified without introducing defects or degrading existing product quality (in anticipation of extensions to the game for Sprint 4).

For each of the design principles above, we define the metric as an interval-type measurement scale of 1-5 as follows:

1 - Poor	Several Major Omissions
2 - Fair	Few Major Omissions
3 - Satisfactory	Many Minor Omissions
4 - Good	Minor Omissions Only
5 - Excellent	No Omissions detected (which does not mean there are none, but we are not aware of any)

(b) QMOOD evaluation

We adapt the Quality Model for Object Oriented Design (QMOOD) proposed by Goyal & Joshi (2014) to further evaluate each design, which incorporates absolute and ratio measurement scales.

This measurement is very subjective and likely does not take into account enough context to provide a fair assessment, however, we will use it as an additional reference.

Summary of the steps:

Step 1: Compute the design metrics that correspond to the following 11 design properties:
Note:
(i) These values can be computed with reference to a complete class diagram.
(ii) The values are normalised with reference to the design with the largest value. For example, if the values of design size of the 4 designs are 10, 6, 5 and 1, then their respective normalised values are 10/10 = 1, 6/10 = 0.6, 5/10 = 0.5 and 1/10 = 0.1

Design Property	**Design Metrics**	**Measurement**
Design size	DSC (design size in classes)	Total number of classes(not interfaces) in design
Hierarchies	NOH(number of hierarchies)	Number of class hierarchies, i.e. the number of root/non-child classes
Abstraction	ANA (average number of ancestors)	Average number of classes from which a class inherits information. Example of calculation: Scenario: Class B, C and D extend class A, and classes E and F extend class B (i) Count the number of child classes for each class. We have the child classes counted as follows: A has 3 first-generation descendants B, C,

		D, and 2 second-generation descendants E, F => 5 child classes B has 2 first generation descendants E, F => 2 child classes C, D, E and F have no descendants => each of them have 0 child classes (ii) Take the average: (5+2+0+0+0+0)/6 = 1.167
Encapsulation	DAM (data access metrics)	Average of the ratio of the number of private attributes to the total number of attributes declared in a class. Use '0' if no information on visibility modifiers is available.
Coupling	DCC (direct class coupling)	Average of the number of different classes a class is directly related to (number of association/dependency relations shown in the class diagram)
Composition	MOA (measure of aggregation)	Number of composition relationships
Inheritance	MFA (measure of functional abstraction)	For child classes, the ratio of the number of inherited methods to the total number of methods in the child class. Average is taken across all child classes
Polymorphism	NOP (number of polymorphic methods)	Number of methods that exhibit polymorphic behaviour
Complexity	NOM (number of methods)	Average of total number of methods in a class
Messaging	CIS (class interface size)	The average number of public methods in a class

Step 2:Compute the values for the quality attributes using the equations:

Quality Attribute	**Index Computation Equation**
Reusability	-0.50*coupling+0.5*messaging+0.5*design size
Flexibility	0.25*encapsulation - 0.25*coupling + 0.5*composition + 0.5 *polymorphism
Understandability	-0.33*abstraction + 0.33 *encapsulation -0.66*coupling - 0.33*polymorphism - 0.33*complexity - 0.33*design size
Functionality	0.25*polymorphism + 0.25 * messaging + 0.25*design size + 0.25 * hierarchies
Extendibility	0.5*abstraction - 0.5 * coupling + 0.5 * inheritance + 0.5 * polymorphism
Effectiveness	0.2*abstraction + 0.2 *encapsulation + 0.2*composition + 0.2*inheritance

	+ 0.2 * polymorphism

Note: We do not compute the Total Quality Index(TQI) highlighted in QMOOD due to its subjective nature. Instead, the values of each quality attribute individually can be used to provide an alternate perspective.

MONASH University

2. Review of Sprint 2 Tech-based Software Prototypes

(a) Interval based assessment

Team Member 1: Maliha Tariq
Chosen Key Functionality: (ii) Flipping of the chit cards

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	5	Flipping of the chit cards along with the game board set up was done well and able to run smoothly without errors
Functional Correctness	4	Only minor absentees where the board does not have a circular shaped line out but other than that everything that needs to be there is there
Functional Appropriateness	5	The class seems fine as there are multiple classes for each function instead of hoarding everything into one. For example the classes, “Card”, “CardFactory” and “ChitCard”
Interaction Capability		
Appropriateness Recognizability	5	The starting menu of the page is straightforward allowing users to know what they want and where to look for (ie., Single Player, Play With Friends)
User Engagement	3	The design has all the right images as well as the necessary shape based on the requirements which allows the user to visualise the actual circular board shape, however as mentioned previously there wasn’t a circle outline and there are gaps between each segment parts of the cards which doesn’t provide the full user experience required
Maintainability		
Coding Standards	4	The classes and methods that are created are being used as well as implemented properly according to their needs There are also multiple classes for different uses which does not clump up everything into a single class, (eg: GameFrame, ControlPanel and GamePanel)
Documentation Quality	4	All classes are documented however some of the methods in the class aren’t documented of their functionalities, but the titles are straightforward and explains itself, However, it lacks comments explaining the purpose and usage of the enum.

Error Handling	2	Shows a GameTests class which means that error is tried and tested for, however, the class doesn't work when tried to run it
Code Readability	5	Code seems clear and concise with good naming conventions for each method and variables used
Modifiability	3	The code does show modifiability by organising functionalities into separate classes and provides basic modularity by separating UI components. However, proper error handling mechanisms (from what I can run) are missing, which could lead to unexpected behaviour and make the code less maintainable.

Total Score: 40/50

Team Member 2: Loo Li Shen
Chosen Key Functionality: (iv) Change of turn to the next player

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	3	The gameboard shows the volcano ring, with the chit cards placed inside and the tokens, however, there are no images, and the view is generic. Change of turn is done by a button and it also rotates the view of the gameboard for the next player
Functional Correctness	5	The turn-changing button effectively switches player turns without errors or glitches.
Functional Appropriateness	3	**The Factory Method** pattern is mentioned but not directly implemented in the code.
Interaction Capability		
Appropriateness Recognizability	4	The code exhibits a clear structure for managing capabilities and characters, facilitating easy understanding of their purpose and relationships.
User Engagement	2	Due to the absence of images on the board, the game board lacks visual appeal, impacting user engagement.
Maintainability		
Coding Standards	4	The code adheres to basic coding standards. However, some improvements could be made, such as organising imports and ensuring consistent formatting throughout the codebase.
Documentation Quality	4	The provided class summaries are informative, yet some classes lack documentation, resulting in inconsistency.
Error Handling	2	Error handling in the provided code is minimal. While some exceptions are caught and printed to the console, there is limited error handling beyond that
Code Readability	3	Variable names are descriptive, and the code structure follows standard Java conventions. However, certain methods could benefit from better organisation and decomposition.
Modifiability	3	The code demonstrates modifiability to some extent by using inheritance and composition to structure the relationships between classes. However, there could be further improvements in terms of making the code more flexible and extensible to accommodate future changes or additions to the game's functionality.

Total Score: 33/50

Team Member 3: Khor Jia Wynn
Chosen Key Functionality: (v) Winning the game

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	5	Functionalities are completed according to the requirements
Functional Correctness	4	Caves are missing animals. Lack of outline for the segments.
Functional Appropriateness	5	All the functions handle everything correctly
Interaction Capability		
Appropriateness Recognizability	4	The provided class diagram and rationales provide a good indication of the solution direction, with no inconsistencies with the implementation. However, tokens should be initialised on the cave, not on the volcano card tile, which may confuse players.
User Engagement	5	Clear interaction logic with the player turns the display on the top-left corner. Excellent effect of using animation to display dragon token movement and flipping of chit cards.
Maintainability		
Coding Standards	5	Extensive and proper use of design patterns and obeying design principles, ChitCardFactory (Factory Pattern), LSP for EntityFactory in SpawnFactory. PlayerTurnManager -> Singleton Pattern
Documentation Quality	4	Minor documentation omissions. Provided documentation can be improved by explaining important methods in more detail.
Error Handling	5	No crashes are encountered when extensive play-testing is done.
Code Readability	3	The code is readable but some parts lack clarity and require further explanation from the author. Some methods are missing documentation.
Modifiability	4	The use of design patterns is suitably justified, and the responsibilities are wrapped up in the corresponding classes. (SRP, OCP in ChitCardFactory designed to extend with new entity types) The design somewhat supports the extension of accommodation of different board configurations, as this logic can be controlled via the arguments passed to the constructor of the VolcanoRingFactory class, with some minor refactoring.

Total Score: 44/50
Team Member 4: Dizhen Liang
Chosen Key Functionality: (iii) movement of dragon tokens based on their current position as well as the last flipped dragon

Assessment Criteria	Marking	Justification
Functional Suitability		
Functional Completeness	5	Setup of game board: Board shows the volcano ring, the caves, the tokens, and the chit cards. Movement of dragon token: Dragon tokens can move when chit cards are flipped.
Functional Correctness	3	Setup of game board: Minor placement issue of animal on Tile Movement of dragon token: 1) Logic error causing incorrect movement behaviour: Should be checking the animal type of the Tile it is on with the animal type of the last flipped chit card. The implementation incorrectly uses the animal type of the cave instead. 3) The player should be able to see all the animals on all the Tiles of the volcano ring at all times. However, whenever a dragon token moves to a new Tile, it covers the animal on it.
Functional Appropriateness	3	Setup of game board: Decomposing the initialisation/setup into separate functions for each of the components practises modularisation. However, assigning it the responsibilities of setting up the game board, handling user interactions, and managing state violates the Single Responsibility Principle. Movement of dragon token: 1) The automated approach to moving the dragon tokens discussed in the design is appropriate. However, players should not be allowed to drag the dragon token, as this would introduce confusion to the aforementioned mechanism. 2) Delegating the responsibility of 'checking if a valid move is made' to the token is acceptable.
Interaction Capability		
Appropriateness Recognizability	3	The provided class diagram and rationales provide a fair indication of the solution direction. However, some confusion arises because of some inconsistencies between the design and implementation. Some classes highlighted in the design are not used (Volcano), and some modularisation techniques (checkTokenMovement, calculateTokenPosition) discussed do not translate to the actual implementation. Consider splitting the sequence diagram into smaller, less complex diagrams to illustrate the interactions more clearly.

User Engagement	3	The image resources used are aesthetically pleasing. The response time is satisfactory, and the flip and movement animations are fair. The token shares the same image as the cave, and the placement of the token directly on the cave makes it difficult to distinguish between them. Consider adding distinguishing elements, or adjusting the placement of the token. As mentioned, the dragon token obscures the animal on the Tile it occupies, preventing the player from seeing it. This makes it difficult for the player to make the next move, since it forces the player to remember the animal on the Tile as well.
Maintainability		
Coding Standards	3	Misalignment between design and implementation: Volcano class was not used. Some parts of the code can be modularised. Some classes handle too many responsibilities (GOD classes): RoundGameBoard, ChitCard
Documentation Quality	3	Some documentation is provided but lacks clarity. Missing documentation for important methods and/or variables. For example, how 'setComponentZOrder()' and repainting are used to achieve the effect of moving the token.
User handling	5	No crashes are encountered when extensive play-testing is done. Logging is provided to aid with development, e.g. errors that may arise during image resource retrieval.
Code Readability	3	Code is generally readable but constrained to the documentation quality. Some parts of the code require detailed explanations from the author.
Modifiability	3	Excessive use of magic numbers affects future extensions. Some refactoring effort is required to support different board configurations. Good use of inheritance: Defining the RoundComponent parent class for the UI components of Cave and DragonToken. It can be used for implementing the Luck Card in future extensions.

Total Score: 34/50

(b) QMOOD

The following is the evaluation of the 4 submissions using QMOOD:

Design Property/metric	Measurement	Member 1	Member 2	Member 3	Member 4
Design size/DSC	Actual	11	4	17	8
	Normalised	0.647	0.235	1	0.471
Hierarchies/NOH	Actual	11	3	11	3
	Normalised	1	0.273	1	0.273
Abstraction/ANA	Actual	0	0.333	0.353	0.5
	Normalised	0	0.666	0.706	1
Encapsulation/DAM	Actual	0.818	0.875	0.882	0.913
	Normalised	0.896	0.958	0.966	1
Coupling/DCC	Actual	2.091	2.25	3.294	2.25
	Normalised	0.635	0.683	1	0.683
Composition/MOA	Actual	3	2	6	5
	Normalised	0.5	0.333	1	0.833
Inheritance/MFA	Actual	0	0.667	0.509	0.671
	Normalised	0	0.994	0.759	1
Polymorphism/NOP	Actual	0	0	1	0
	Normalised	0	0	0	0
Messaging/CIS	Actual	1.818	2.5	1.294	3.625
	Normalised	0.502	0.690	0.357	1
Complexity/NOM	Actual	1.909	2.5	1.882	3.625

	Normalised	0.527	0.690	0.519	1

Calculation of quality attributes using the equations:

Quality attribute	Member 1	Member 2	Member 3
Reusability	0.257	0.121	0.179
Flexibility	0.315	0.235	0.492
Understandability	-0.511	-0.660	-1.075
Functionality	0.537	0.300	0.589
Extendibility	-0.318	0.489	0.232
Effectiveness	0.280	0.590	0.686

Summary:

Member 1	Member 2	Member 3	Member 4
The assessment indicates that they excel in certain aspects, earning high scores for functional suitability and appropriateness recognizability, thanks to its smooth execution and straightforward user interface. Functional accuracy is almost perfect, with very small exceptions, such as the lack of a circular board outline. The code demonstrates good maintainability with clear coding standards and documentation, although there are some gaps in method	Based on the assessment, the current implementation demonstrates functional completeness to a reasonable extent, with the gameboard effectively displaying the volcano ring and chit cards, albeit lacking images, resulting in a somewhat generic visual experience. The turn-changing mechanism, facilitated by a button, functions correctly and enhances gameplay dynamics. However, the absence of images detracts from user engagement,	All functionalities are almost perfectly implemented and beyond the scope of the chosen key functionality. The minor omissions are: 1) Cave should have animals. 2) While the player's dragon token has not exited the cave, animal-type checks are done between the Cave's animal and the chit card's animal. The rationale for the chosen solution direction is consistent with the implementation, and ample justifications are	In terms of functional requirements, correctness requires the most improvement. This should be coupled with extensive testing of the various scenarios for moving the dragon token. Most of the rationales for the chosen solution direction are appropriate for the requirements in Sprint 2. The use of inheritance for abstracting the common UI features is appropriate and can be reused in the consolidated design. Given the design documents and

documentation and error-handling implementation. While the code is modifiable due to the organisation of functionalities into different classes, the lack of suitable error-handling techniques may impede future maintenance efforts, reducing overall maintainability. Overall, the code has strong functional performance and user interface capabilities, while there is potential for improvement in error management and modifiability.	impacting the overall gaming experience. While the code exhibits a clear structure for managing capabilities and characters, there are opportunities to improve error handling, enhance code readability through better organisation and decomposition of methods, and ensure consistent documentation across all classes. Additionally, implementing the Factory Method pattern, as mentioned in the design rationale, would further enhance functional appropriateness.	provided. The solution direction is mostly understandable with reference to the class diagram, sequence diagrams and rationales. However, the source code documentation should be more comprehensive to improve readability. The code is written with a logical flow, and usage of case analysis is appropriate for the specific solution direction. The aesthetics of the user interface are satisfactory.	rationales, the solution direction was understandable, with only a few inconsistencies between the design and the implementation. The written rationales are easy to understand, but simpler sequence diagrams are needed to visualise interactions within the system. To support future work, there is a need to reallocate some of the responsibilities of GOD classes to existing classes or to new dedicated classes. There is some refactoring required to support various board configurations as the extension requirement in Sprint 4. The source code requires more comprehensive documentation to clarify semantics. Case analysis is used where necessary, but alternative solutions can be explored. The aesthetics of the UI is satisfactory, though the animations have potential for improvement.

MONASH University

3. Outcome of the Assessment

Based on the above reviews, we decided to use team member 3's tech-based software prototype as a baseline for Sprint 3. The justifications are as follows:

1) It is the most functionally complete, correct and appropriate to the basic Fiery Dragons game
    a) Completed implementation for setting up game board, flipping of chit cards, change of turn, and winning the game.
    b) Extensive play-testing was done to cover all possible scenarios, and correct behaviour was displayed (with the exception of the issue mentioned above)
2) The aesthetics are acceptable and consistent. Card flip animation and dragon token movements are visually appealing.
3) The solution direction for board setup aligns with other team members and is justified
    a) Decomposition of the game board into separate components avoids a single GOD class
    b) Defining classes with different roles: Manager, Information/State Holder and Factory/Initialiser enforces separation of concerns

We decided to integrate some unique elements of the other tech-based prototypes into Sprint 3:

Sprint 2 Prototype	**Element to be integrated**
Team member 1	● Main menu of the Fiery Dragons game application. ● Defining a generic Animal class.
Team member 4	Implementation of animals on caves, as well as the logic for checking animal types when a dragon token is still in its cave.

We propose the following new elements to improve the prototype:

a) Encapsulation of related classes into packages
b) Updated aesthetics:
    i) Player ID indicator that follows the dragon token as it moves
    ii) Wooden table texture was used for the background
c) New solution direction for animals: Define specific animal instances (BabyDragon, Bat, DragonPirate, Salamander, Spider) which extend the abstract Animal class. The Animal class contains a 'factory method' getAnimal() to return a new Animal instance based on the specified AnimalType(enum) and number of animals. This way, we isolate the nested if-else to a single point (Animal), and other parts of the code can perform animal type-checking using the AnimalType enum. (which also avoids having to cast types) This is open to future extensions (where new animal types are added) since we just need to:
    i) Add a new class extending the Animal
    ii) Add a new AnimalType to the enums
    iii) Update the getAnimal() method to include the check for this new AnimalType
d) Decompose large methods into smaller, modular components.
e) Define an InitModel interface for classes that are responsible for instantiating classes (in the 'Model' package) for the first time during the initialisation stage of the game. This enforces separate initialisations of the View and the initialisation of the Model.

MONASH University

4. Object-Oriented Design

1. Class-Responsibility-Collaboration (CRC) Cards

FieryDragonsApplication	
• Creates a main menu for the game	MainMenu
• Knows and tracks whether a player has won	Player, DragonToken
• Initialises the game by requesting the Factory and Manager classes to create the view components and the state	VolcanoRingFactory PlayerTurnManager DragonTokenFactory CaveFactory ChitCardFactory ChitCardFlipManager ButtonFactory TextFactory TextDisplayManager
• Requests the manager classes to handle updates when a player's turn ends or when the game ends	ChitCardFlipManager TextDisplayManager PlayerTurnManager
• Requests the chit card view components to listen for user input(clicks), and asks the Player and Manager classes to respond to them.	Player ChitCardAdapter ChitCardFlipManager
• Allows the next player to play their turn if the game has not ended	PlayerTurnManager

Description:
It is the main entry point of the game, extending the GameApplication class provided by the FXGL framework. It provides game settings like the background, application window dimensions, and the MainMenu, which is what the Player sees when the game is first launched. It serves as the central hub through which game components are initialised and managed. The onUpdate() method that is called by FXGL every frame allows custom logic to be implemented to handle different events.

Alternative: Move the responsibility of handling specific user inputs (like card clicks) to a dedicated InputManager class. This would allow FieryDragonsApplication to focus on the game's lifecycle and state management, while InputManager translates raw input into actionable game events.
Reason for rejection: Currently there is only one form of input (chit-card clicks), so creating an additional Manager class introduces unnecessary complexity. However, it can be considered if a variety of inputs are to be implemented in the future.

PlayerTurnManager	
• Knows the current player's turn	
• Instructs DragonToken to move (valid move)	DragonToken
• Sets the dragon token's destination Volcano Tile to "occupied" after a valid move	VolcanoRingFactory
• Updates DragonToken's state after a valid move: current Volcano tile and total movement, whether it has moved out of the cave	DragonToken
• Creates and tracks the player instances	Player
• Transitions turn to the next player	

Description:
The PlayerTurnManager creates the Players during initial setup, assigning them a random starting animal (that is shown on the cave). It continues to track the players throughout the game, and tracks which player is currently playing its turn.
As the controller, whenever a successful move is made, it asks DragonToken to move (view) and updates the state of VolcanoCard(Tile) and DragonToken (model). It also manages the change of turn to the next player.

Alternative: Create a TokenMovementManager that is responsible for moving tokens. This separates the logic of moving tokens from player management, allowing PlayerTurnManager to focus on turn order and state.
Reason for rejection: Because of the composition relationship between Player and DragonToken, the PlayerTurnManager (which manages all Player instances) has indirect access to DragonToken. It would be unnecessary to expose the DragonToken's interface to a separate manager class. Additionally, encapsulating the update logic within the PlayerTurnManager class after a valid move helps to ensure all updates occur in the correct sequence (minimise effects of connascence of execution)

Player	
• Knows the animal on the random cave it starts on (assigned by PlayerTurnManager during creation)	PlayerTurnManager
• Controls its dragon token (composition relationship)	DragonToken
• Checks whether its turn should end (invalid move)	DragonToken, ChitCardAdapter, Circle, VolcanoRingFactory

• Determines the destination volcano card tile (valid move)	DragonToken, Circle, ChitCardAdapter

Description:

The Player class is created by the PlayerTurnManager during the initial setup, where it is assigned a random animal. This animal type will be passed on to the CaveFactory so that it can render the required animal in the cave.
To fulfil its responsibilities, it requires information from its :
(i) DragonToken (e.g. whether the dragon token has moved out of the cave, the animal on the tile the dragon token is currently on)
(ii) the ChitCardAdapter, which returns the ChitCard instance(model) bound to the UI shape (view), for the player to access the animal type on it. This animal type is used for checking the animal type of the chit card against the destination volcano card.

Alternative: Implement rollback/moving token backwards if it lands on a tile already occupied by another player's dragon token, instead of move validation.
Reason for rejection: There are no significant design differences, so this is likely up to personal preference. However, move validation is arguably less complex to implement since we only move the token once. If we were to move it twice, we would need to ensure that state correctness is preserved across the other game objects. (e.g. volcano card tile occupation status)

ChitCardFlipManager	
• Tracks all the view components (Circle) of the ChitCards(uncovered and covered)	Circle
• Animates the flipping of chit cards clicked by Players (set the covered side to invisible and uncovered side to visible)	RotateTransition, ChitCard
• Animates the flipping of chit cards when a player's turn ends (set the covered side to visible and uncovered side to invisible)	RotateTransition, ChitCard, ChitCardAdapter, Circle

• Toggles the click listeners of the ChitCard's uncovered view component (Circle)
  ○ Toggle off once the card has been uncovered
  ○ Toggle on when the player's turn ends
	ChitCard, Circle

Description:

The ChitCardFlipManager performs the rotation animation
(i) Whenever a player clicks on a covered chit-card shape
(ii) on all uncovered chit-card shapes whenever a player's turn ends

It maintains an array of covered and uncovered chit-card shapes to help with this. To get the chit card view components for modifying this array, it collaborates with the ChitCard model since the ChitCard model contains references to both its covered and uncovered form. The ChitCardAdapter also helps to get the covered form (Circle) of a chit card by first adapting the uncovered form (Circle) to the ChitCard, and then obtaining the reference to the covered form of the ChitCard. By providing this info, the ChitCardFlipManager can set the visibility of the covered form to 'True'.

Alternative: Animation logic could be moved to an AnimationManager class, which would handle all animations in the game, not just card flips. This would allow for centralised control over animations and potentially enable the reusing of animation code.

Reason for rejection: There is little similarity between the animation code of chit cards and the dragon token movement, so code reuse is not viable. Centralising the animation logic is not a priority in our game since there is no requirement for handling multiple animations simultaneously.

Animal	
• Knows the base image path of the resource used for rendering animals on ChitCards/VolcanoCard tiles.	Config
• Knows how many animals are used	ChitCardFactory, VolcanoRingFactory
• Helps to create and fill a shape with the ImagePattern with the specified number of animals. This is polymorphic since the Shape type is generic and can be substituted with any specific shape, e.g. Rectangle for VolcanoCard/Circle for ChitCard	Image, ImagePattern, Shape
• Contains a factory method which returns a new instance of a specific animal based on the AnimalType enum. It isolates case analysis into a single class, and is less complex than multiple dispatch	BabyDragon Bat DragonPirate Salamander Spider

Description:

The abstract Animal class helps create custom animal image patterns that can be applied as the 'fill' property of Shapes. The customisation involves the specific animal type and the number of animals to be displayed in the image.

Alternative: Multiple dispatch for checking animal type. (Here, we choose to use an AnimalType enum that is used for checking animal types, instead of using multiple dispatch)
Reason for rejection: Multiple dispatch does not scale well when the number of animals increases, and since we have 5 types (including DragonPirate), we would need to define one method in each animal class for checking with every other animal type.

VolcanoRingFactory	
• Manages a single reference to the volcano ring of VolcanoCards	
• Creates volcano segments from the configuration of segments.	VolcanoSegment VolcanoCard
• Initialises the view (Rectangle) component from randomly arranged volcano segments. It is arranged as a Group in the shape of a ring, and the randomisation is provided by the Utils class	Util, Rectangle,, Group, Animal
• Initialises the model(VolcanoCard component) from the randomised volcano segments.	VolcanoCard, VolcanoSegment

Description:
VolcanoSegment is a static helper class within VolcanoRingFactory to help with the creation of segments. The Animal class helps to fill the Rectangle with the specific animal image. Here we make use of the Liskov Substitution Principle, since the abstract fillUIShape() method will work with any inheritor of the Animal class (polymorphism). In summary, this class helps initialise the view and model for the volcano ring.

Alternative: The creation of game board elements could be abstracted into a BoardFactory class, which would be responsible for generating all board-related entities in a consistent manner. VolcanoRingFactory could then focus on the specifics of the volcano ring's unique arrangement.
Reason for rejection: This is something that is already done in the FieryDragonsApplication's initGame() method, and does not need to be abstracted into a class which provides this trivial implementation.

DragonToken	
• Executes the movement animation whenever a valid move is made.	Timeline KeyValue KeyFrame
• Knows its starting volcano card tile (assigned by DragonTokenFactory during creation) and provides this information to Player (for checking if a token can move backward beyond the cave)	DragonTokenFactory

• Knows and updates its current position on the circular path around the volcano ring it is free to move around on (measured by an angle)	
• Knows the view component (Rectangle) bound to it and the text label that travels with it	
• Knows and updates the current volcano card tile (and the AnimalType of it) it is on. It is updated when the PlayerTurnManager instructs it to do so.	PlayerTurnManager
• Knows and updates its total movement count (PlayerTurnManager instructs it perform updates)	PlayerTurnManager
• Knows and updates the status on whether it has moved out of the cave	PlayerTurnManager

Description:

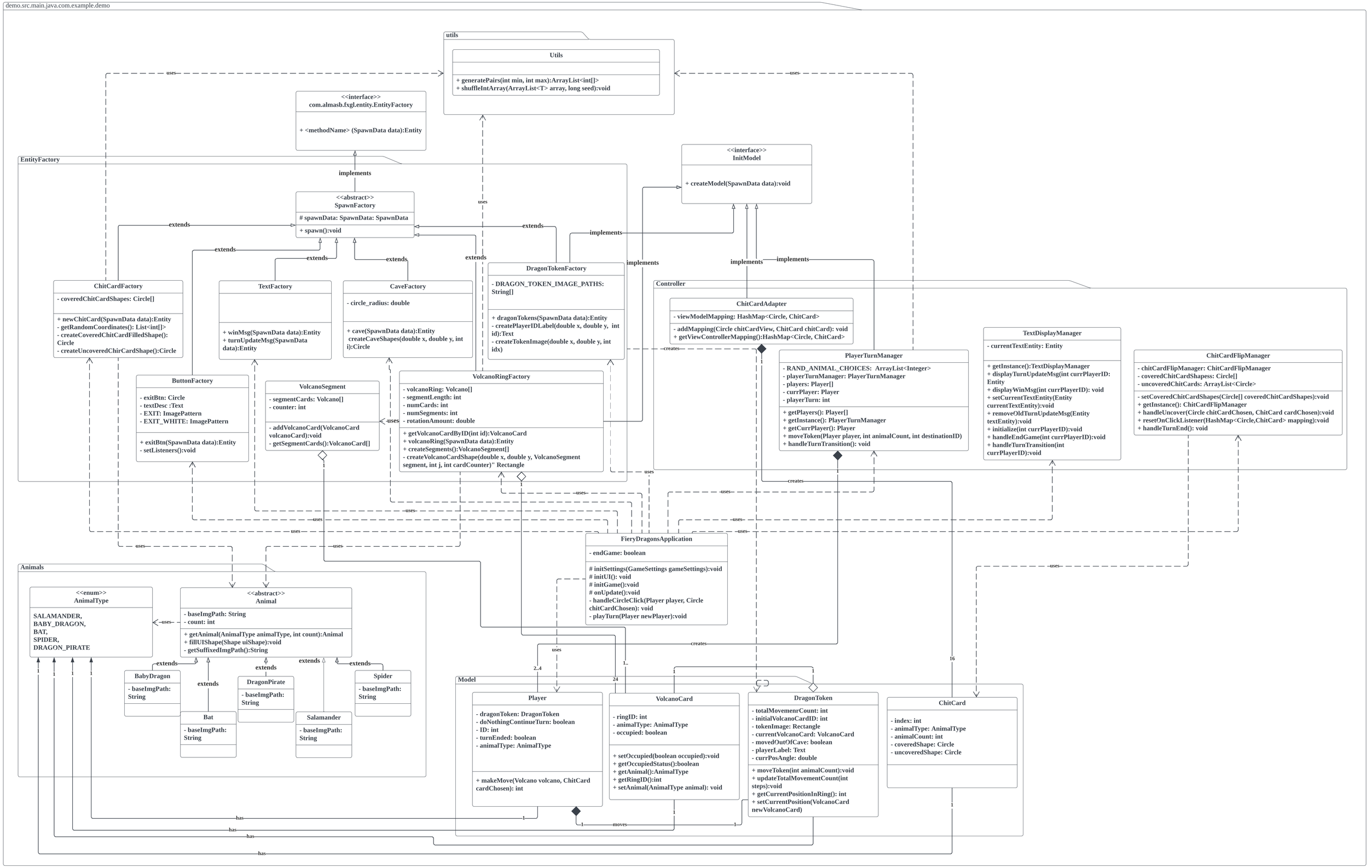
The dragon token represents the player's piece on the game board and is moved according to the game's rules. It tracks the current position and movement count, which are essential for determining the game's progress and outcome. When the total movement count reaches the number of volcano cards, a win condition is detected.
It has two main responsibilities: animating the movement (Using Timeline, KeyValue, KeyFrame) and providing a public interface for classes to read/write to its properties.
Alternative: Moving the token could be the responsibility of a GamePieceMovementController, which would handle the logic of moving any game piece, including dragon tokens. This would allow DragonToken to act more as a data holder for the token's state, with all movement logic externalised.
Reason for rejection: However, in the current game, there are no other moving pieces, so it is reasonable for it to handle this responsibility.

2. Class Diagram

Note:
1) Some classes, such as utility classes used for animations (RotateTransition) are not directly involved in the design and hence omitted from the class diagram. It mainly shows the classes involved in the design.

2) Some getters and setters are also omitted to avoid overcomplicating the diagram.

2. Class Diagram



5. Executable Instructions

Instructions:
The executables were built using Maven based on the pom.xml file. The output is a zip file and it should be located in a directory called 'target'.

You should be using Java SDK version 22.
To produce the zip file: In your IDE, run the command `mvn javafx:jlink` using Maven.
In IntelliJ, this can be done by pressing the Ctrl key twice and then searching for this command.

YouTube Video Link:

6. Sprint Contributions

Wiki GitLab

https://git.infotech.monash.edu/FIT3077/fit3077-s1-2024/MA_Tuesday08am_Team123/-/wikis/Contribution-log

Contribution Log Excel

https://docs.google.com/spreadsheets/d/1fVo7kWvmeEl-KOMaM-aRRMiX9ivhWH3TT6GkVxvbUTY/edit#gid=0

Contributor Analytics

MONASH University

7. References:

Goyal, P., Joshi & G. (2014). QMOOD metric sets to assess quality of Java program. https://ieeexplore.ieee.org/document/6781337