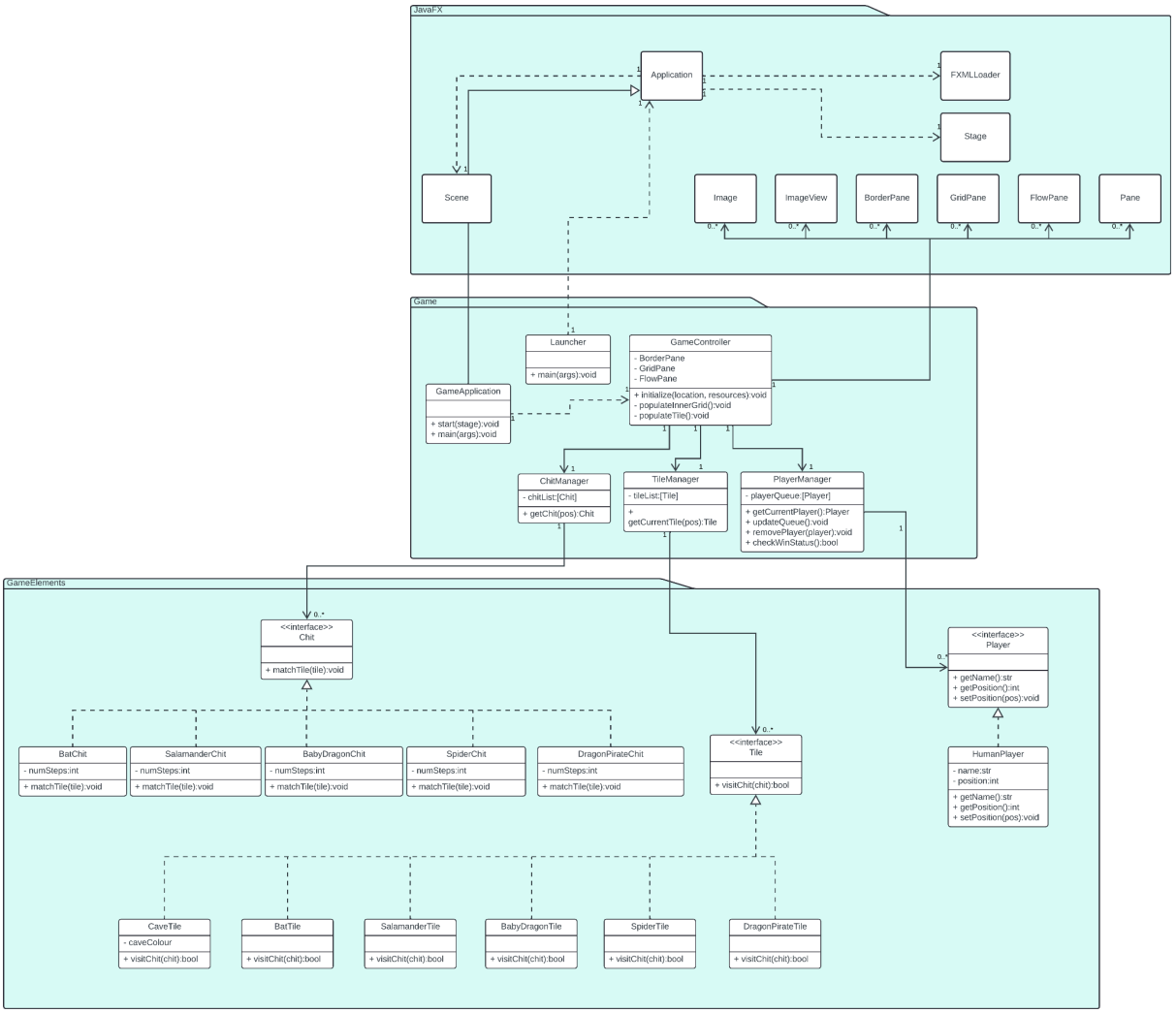
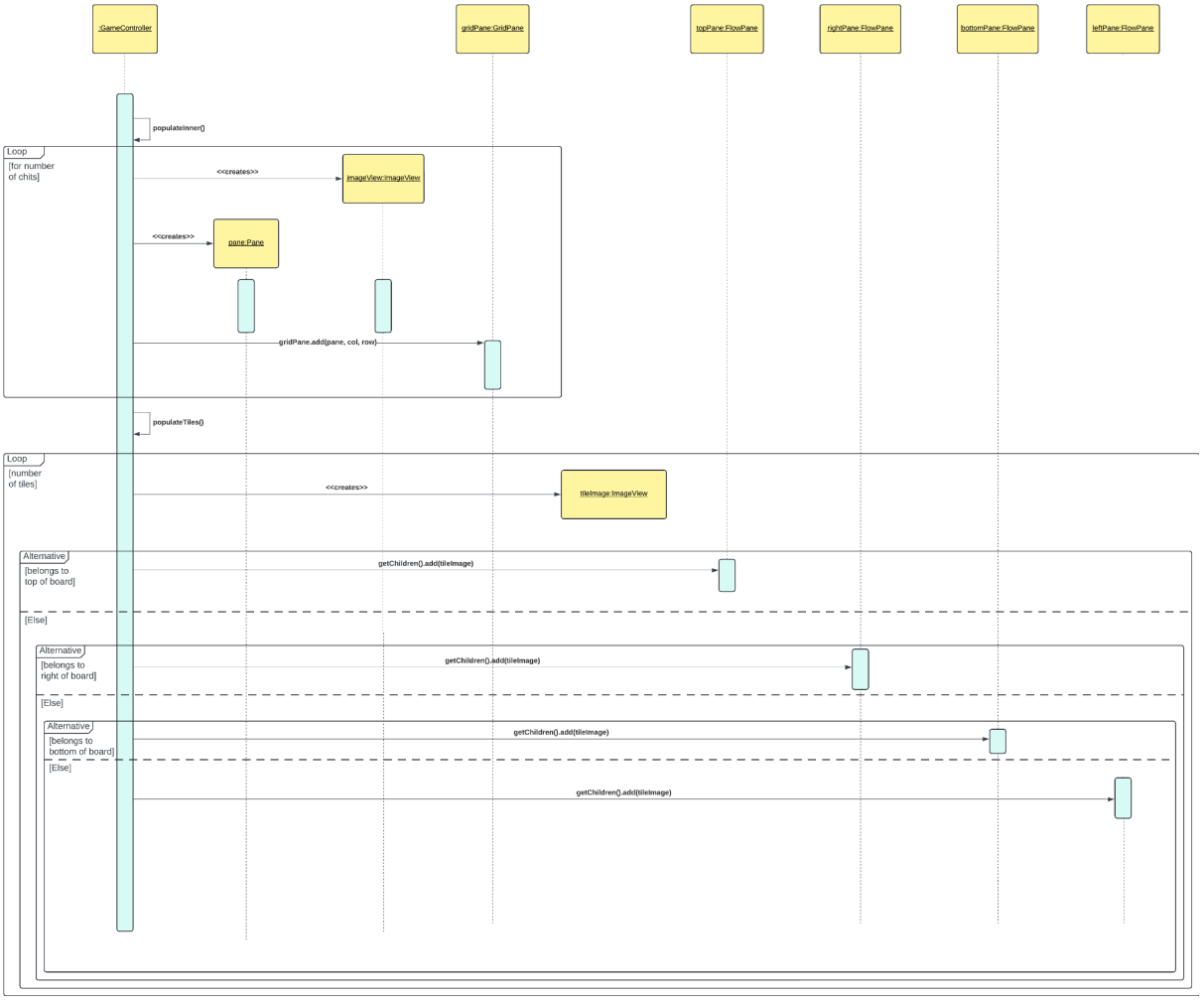
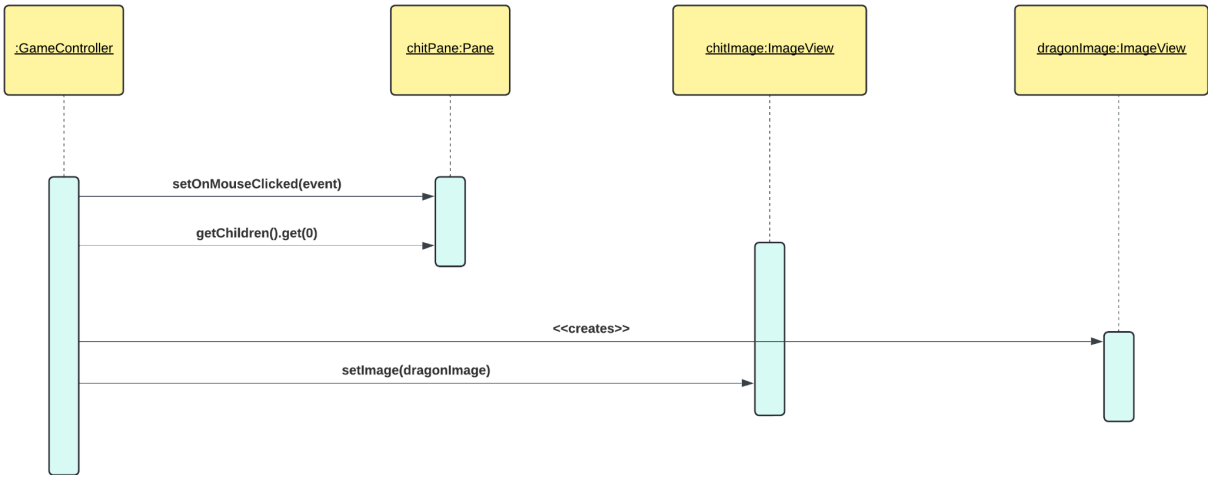
Class Diagram



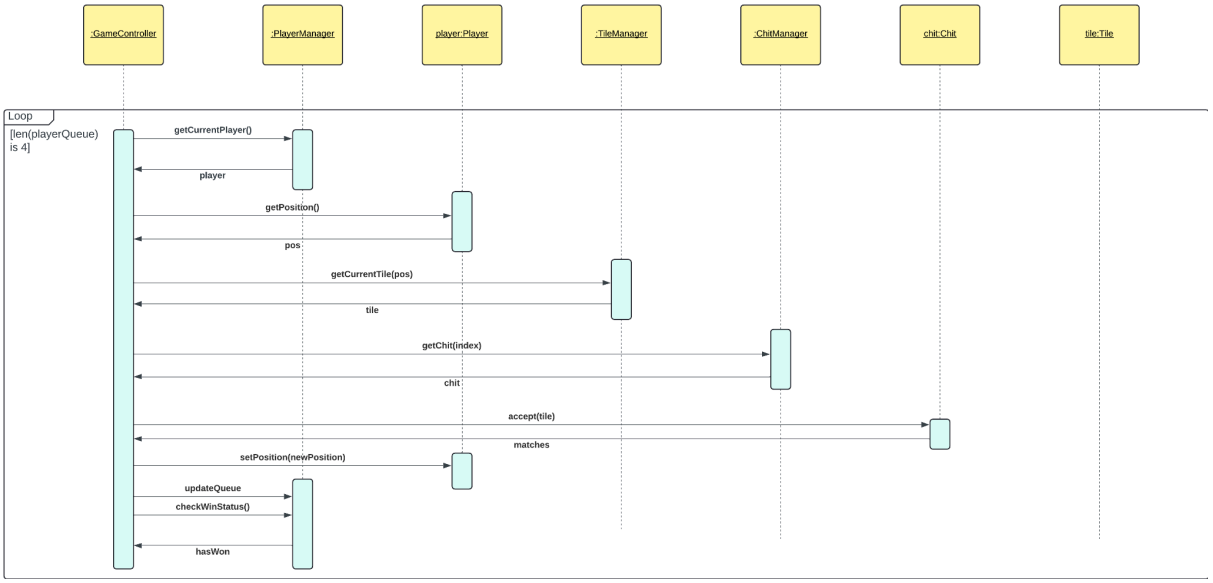
Sequence Diagrams



This diagram showcases the creation of the board. A loop for the number of chits creates a Pane and ImageView, sets the ImageView in the Pane, and adds it to the GridPane for each chit. And another loop similarly adds images to FlowPane instances for the top, right, bottom, and left of the chits. These flowpanes are actually attached to the corresponding top, right, left, and bottom borders of a BorderPane which wraps around the GridPane.



This diagram showcases how a chit is flipped. When a pane containing a chit is clicked, the current image (of an unflipped chit) is fetched from the pane. It is then replaced with the image of the corresponding chit from the resources of the java application, simulating flipping a chit.



The above diagram showcases how, in a while loop which continues as long as players are still playing, the GameController fetches the chit, player, tile, and position. It then checks if the selected chit is a valid move in the value ‘matches’, which is obtained via double dispatch and visitor method using the chit and tile. It uses that information to calculate the newPosition, and sets it for the player. The board will be redrawn depending on the new position of the player.

The above diagram also shows how, after all the actions occur in a round, the playerQueue in PlayerManager is updated (player at front of queue is moved to back). So when the loop starts again, the next player will be fetched.

The above diagram shows how, at the end of the loop, the hasWon() method is run using the PlayerManager which checks if any player has won. If the appropriate game-end conditions are met, the loop will end.

Design Rationales

HumanPlayer

This class has been added instead of having a single Player class without extending from an interface so that the game is open to extensibility in the future. Different implementations of Player such as AIPlayer, RemotePlayer, and more may be added. This design also allows each type of player to have their own implementation of methods which are likely to be different. It is not appropriate to have different

methods in the same class to handle each kind of player because the concept of these different kinds of players are better represented as a class.

BatChit

The BatChit, and every other dragon chit, have been implemented as concrete classes implementing the Chit interface instead of having a Chit class with an attribute which defines the animal. This is because each concrete dragon chit represents a specific type of animal, which promotes a clear separation of concerns. With a single Chit class and an attribute defining the animal, it would have to include conditional logic to handle the different behaviours of each animal chit. Despite each animal chit being functionally the same in the base game, they may be extended to have different effects in the future.

GameApplication - Scene

A scene is a JavaFX class used for displaying content (in this case, it will display the game board). The relationship between GameApplication and Scene is a composite relationship since the Scene object responsible for displaying the board is created within the context of GameApplication, its life cycle is dependent on it, and is directly managed by it; it is tightly coupled with the GameApplication object.

ChitManager - Chit

This is an aggregate relationship, because Chit implementations can be contained within ChitManager, but they do not require ChitManager to serve their function or exist.

Inheritance of CaveTile from Tile Interface

The CaveTile is made to inherit from Tile because, like the other tiles, the player can traverse on top of it. By inheriting from Tile, it may have some functionality similar to other tiles. On top of that, CaveTile is functionally different in some ways to other tiles; this can allow for future extension where each type of tile, including CaveTile, can have a different effect on the player, or have a different appearance. This design is driven by the need for some types of tiles to share common tile functionality while also allowing for specialised behaviour for different tile types in the game.

Cardinalities

The ChitManager to Chit cardinality is 1 to 0..*. The ChitManager acts as a controller for Chits in the game. The 1 cardinality for ChitManager is due to the fact that only one instance of the manager will exist in the system, responsible for handling all Chit instances. 0..* cardinality for Chit means the ChitManager can manage zero, one, or multiple instances of Chit.

The GameController to ChitManager cardinality is 1 to 1. The "1" cardinality on the GameController side indicates that there is only one instance of GameController in the system. This is because the GameController class acts as a central controller or manager for the entire game. The "1" cardinality on the ChitManager side means that the GameController is associated with exactly one instance of the ChitManager class. The GameController relies on or manages a single instance of the ChitManager to handle all Chit-related operations or logic within the game.

Design Patterns

Singleton Pattern

The singleton pattern is used for the ChitManager, TileManager, and PlayerManager. This is because the resources they manage, Chit, Tile, and Player instances, will be accessed and utilised by multiple parts of the system. Using the singleton pattern ensures that creation and modification goes through a single authority, preventing potential conflicts or inconsistencies that could arise from multiple instances. As a result the overall design becomes simpler and easier to maintain.

Visitor Pattern

The Chit and Tile implement visitor pattern with double dispatch. The tiles 'visit' the chits and the visit method of a tile acts according to the chit, if the chit matches the animal it handles logic concerning the player moving forward, if the chit is a pirate chit it handles going backwards, and if the chit is a different animal the player does not move. This pattern was used because it follows the open/close principle, and there is no need to modify existing behaviour if new operations are to be added to Chit classes. It promotes code reusability. Also, by separating the operations from the Chit object, the code is also more maintainable. Double dispatch also allows for more flexible behaviour depending on the combination of tile and chit.

Video Demo Link

https://youtu.be/g44KUDjH6Wg