FIT 3077 SPRINT 2 DELIVERABLES

Team Name: **U-knownZero**
Member Name: Yeoh Ming Wei (32205449)
Tutorial: 01

Table of Contents

1

Object-Oriented Design and Design Rationales

Class Diagram

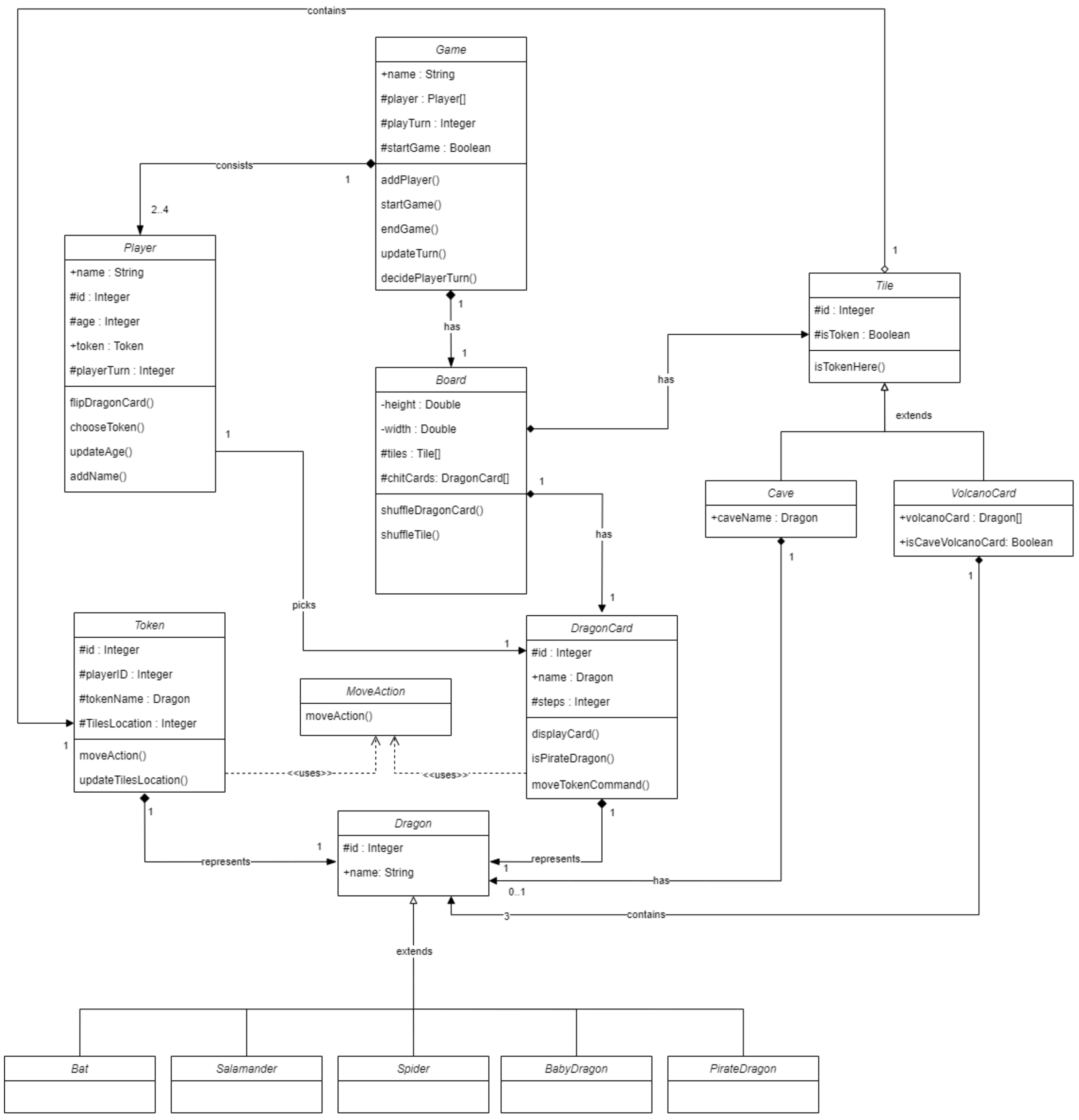
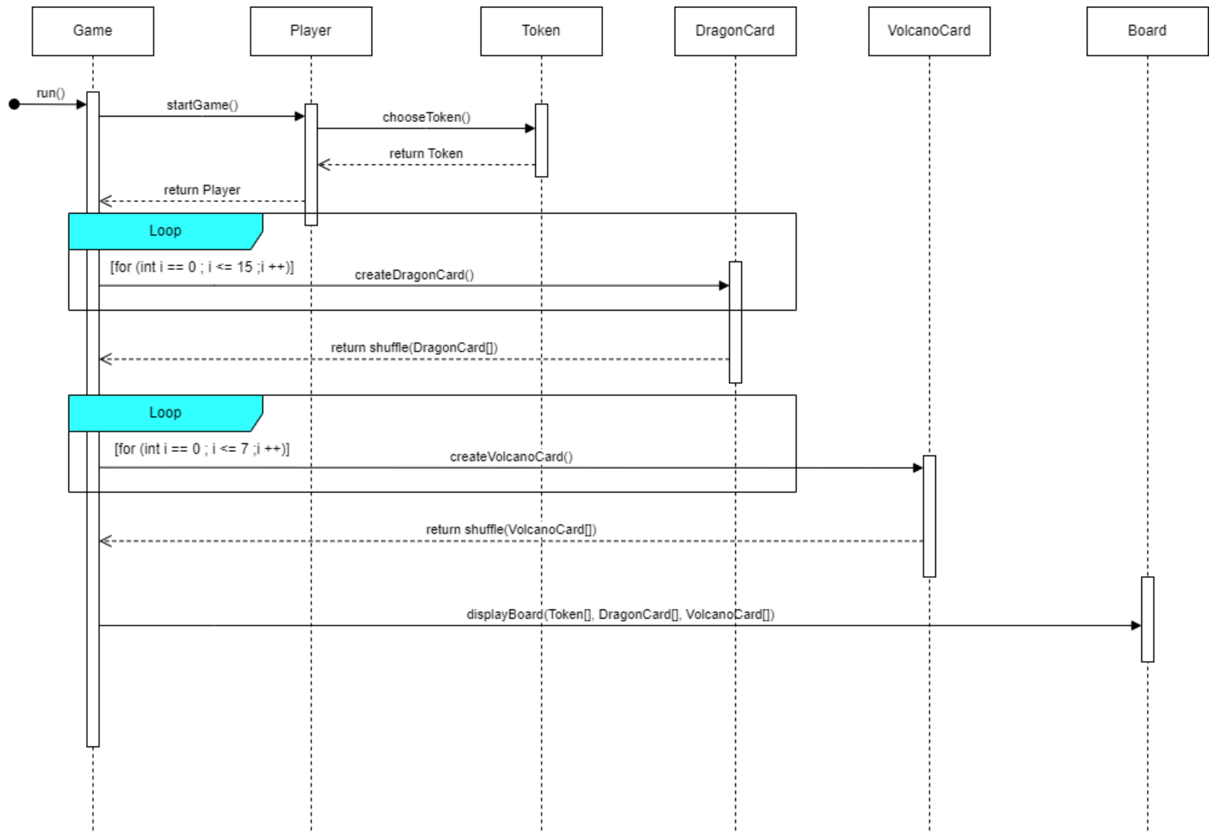


Figure 1: Class Diagram for Fiery Dragon

Sequence Diagram

Setting up the game board



Game
Player
Token
DragonCard
VolcanoCard
Board
run()
startGame()
chooseToken()
return Token
return Player
Loop
[for (int i == 0 ; i <= 15 ;i ++)]
createDragonCard()
return shuffle(DragonCard[])
Loop
[for (int i == 0 ; i <= 7 ;i ++)]
createVolcanoCard()
return shuffle(VolcanoCard[])
displayBoard(Token[], DragonCard[], VolcanoCard[])

Figure 2: Sequence Diagram when setting up the board

Flipping the dragon card (or chit card)

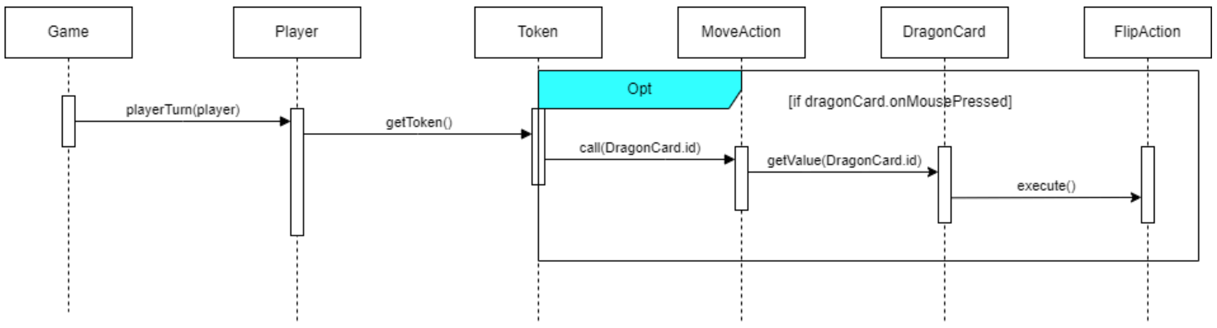


Figure 3: Sequence Diagram when flipping a dragon card

Movement of dragon token

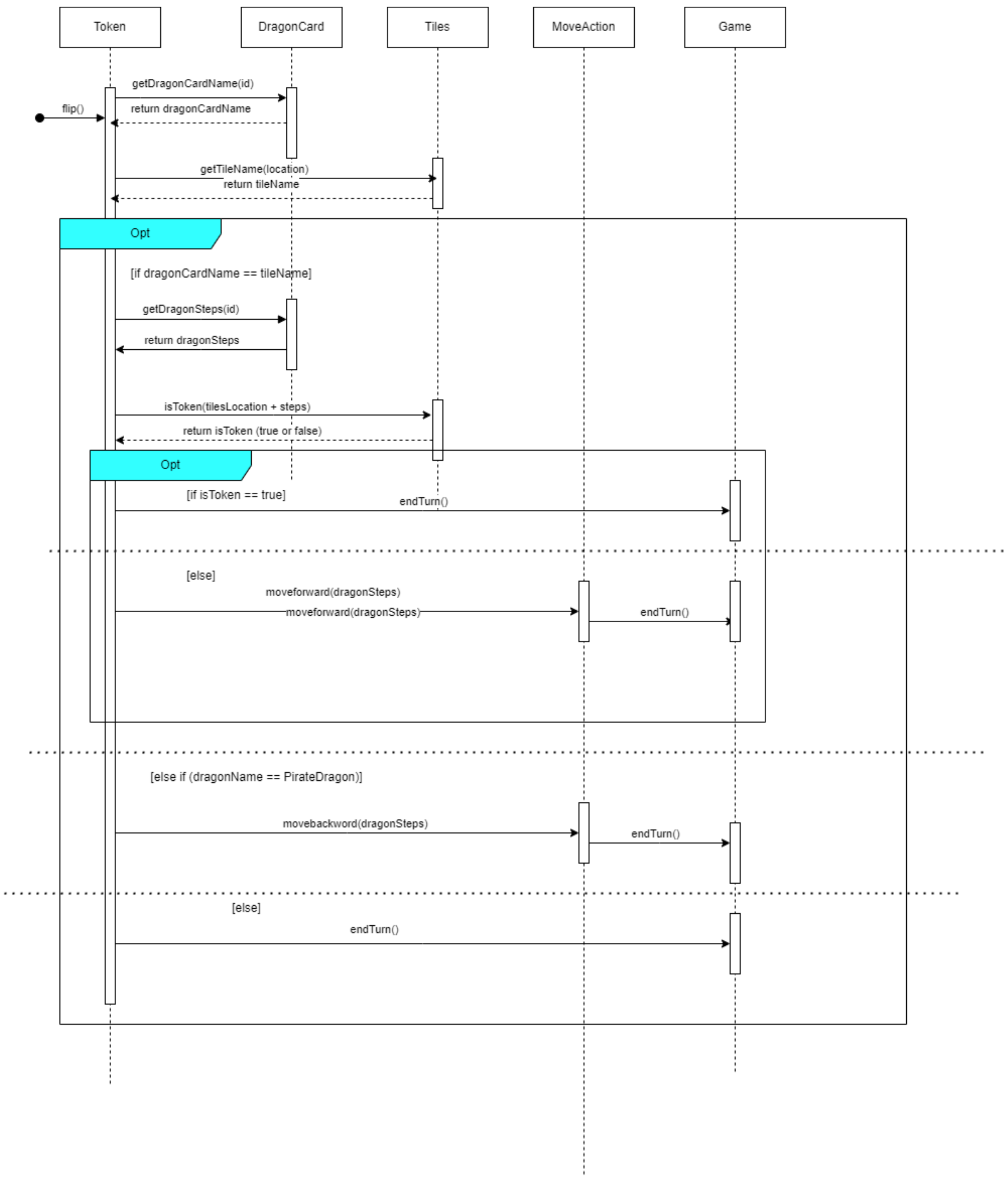


Figure 4: Sequence Diagram when a token moves

Change of turn

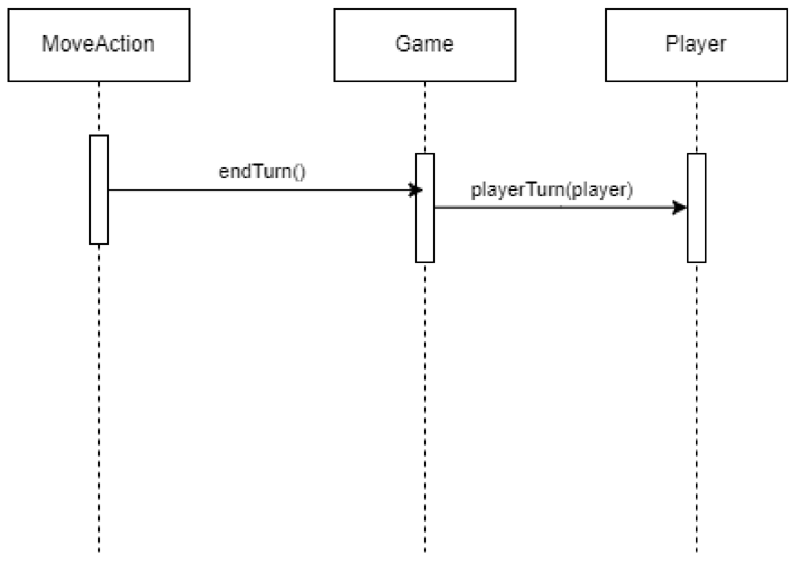


Figure 4: Sequence Diagram when a token moves

Winning The Game

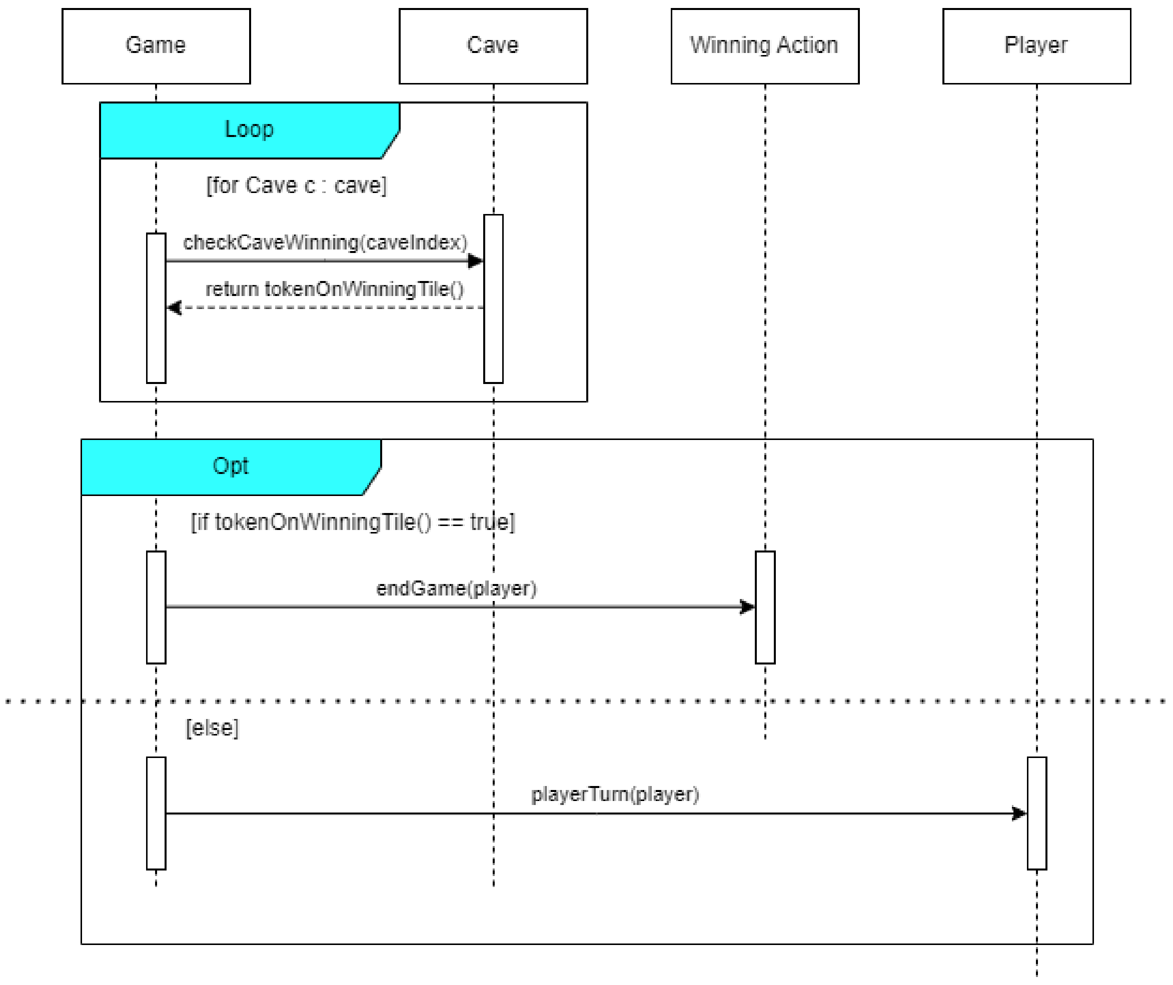


Figure 6: Sequence Diagram when the player wins

Design Rationale

Explain two key classes (not interfaces) that you have included in your design and provide your reasons why you decided to create these classes. Why was it not appropriate for them to be methods?

One of the two key classes that I had included in my application design is the game class and the board class that are the most important classes in the application. The game class is of the main design that works behind the scene. The functionality of this class includes, assigning the name of the game, acts as a tick to ensure the runtime of the game such as start and end game, manage the player and the turn of the player. The reason why we have a game class is to ensure that another class which is the board class does not become the god class. The game class is more suitable for managing anything that cannot be seen on the board, such as the player turn or control the start and end time of the game. A game class is not suitable to become a method for the application because it is a generalised term which represents the game of the application. However, the game has many functions which can exist inside the game class.

Besides that, I had also included a board class which is mentioned above to ensure that the board class does not become a god class. The function of a board class includes attributes such as height, weight, volcano cards and dragon cards. These are the characteristics of the board and also objects that are contained within the board. The board also includes some functionality such as shuffling the dragon card and volcano cards. Creating a separate class for the board ensures that all the functions that are related to the board will be encapsulated so that it is more readable and easier to modify. Same as above, it is not suitable to become a method as it is a term with broad functionality. Hence, class is more suitable for the board.

Explain two key relationships in your class diagram, for example, why is something an aggregation not a composition?

The first key relationship that I would like to explain is the relationship between the tile class and the token class. Based on the real game, the board has 28 tiles which represent the volcano cards. However there are only 4 maximum tokens. As the token moves, there may be times where the tiles have a token and some don't. Therefore, the relationship between them is aggregation as tiles work independently between tokens. The tiles can still exist without the token above the tiles.

As for the next relationship we have the key relationship between the game class and board class. To have a scenario, the game must have a board in order to be played by the user. At the same time, the board must also have so that the game would function as well . So, the relationship between the game class and board class is composition as the game cannot exist without the existence of the board, it will not function without it.

Explain your decisions around inheritance, why did you decide to use (or not use) it? Why is your decision justified in your design?

Since there are animals that have the same attributes and methods in this game, we can create a new parent class which is an animal class. Inheritance encourages reuse of code without code redundancy. In this case, we have 5 child animals that will inherit the dragon parent class so that it shares the same attributes and method. It also enables us to extend the functionality through the child class so that you do not need to modify code from the parent class. However, for this game, the dragons do not serve any purpose of providing any special methods. So, there is only a name attribute without methods.

Besides that we also have cave and volcano cards that serve a similar purpose for the board, which is a representation of tiles. Therefore, we can create a tile class as the parent class and the cave and volcano cards will extend from the parent class. In this particular class, we can add different methods in the child classes such as assigning one dragon to cave and 3 dragons to volcano cards. This shows that while sharing the same attributes through parent class, we can perform modifications through child class if there are different attributes and methods.

Explain how you arrived at two sets of cardinalities, for example, why 0..1 and why not 1…2?

As there are quite a few cardinalities in my domain model, I will pick only two sets that is more important as an example.

The first cardinalities that I encountered for the game is between the game and the player. As the rule of the game has a minimum player of 2 and a maximum player of 4. Therefore the cardinality will be 1 to 2..4, which means that one game can have 2 to 4 players.

The other cardinality is between the cave cards and the dragon cards. In this case, we have a cardinality of 1 to 0..1 which means that a cave must either stay a dragon or the dragon has left the cave and located at the volcano cards.