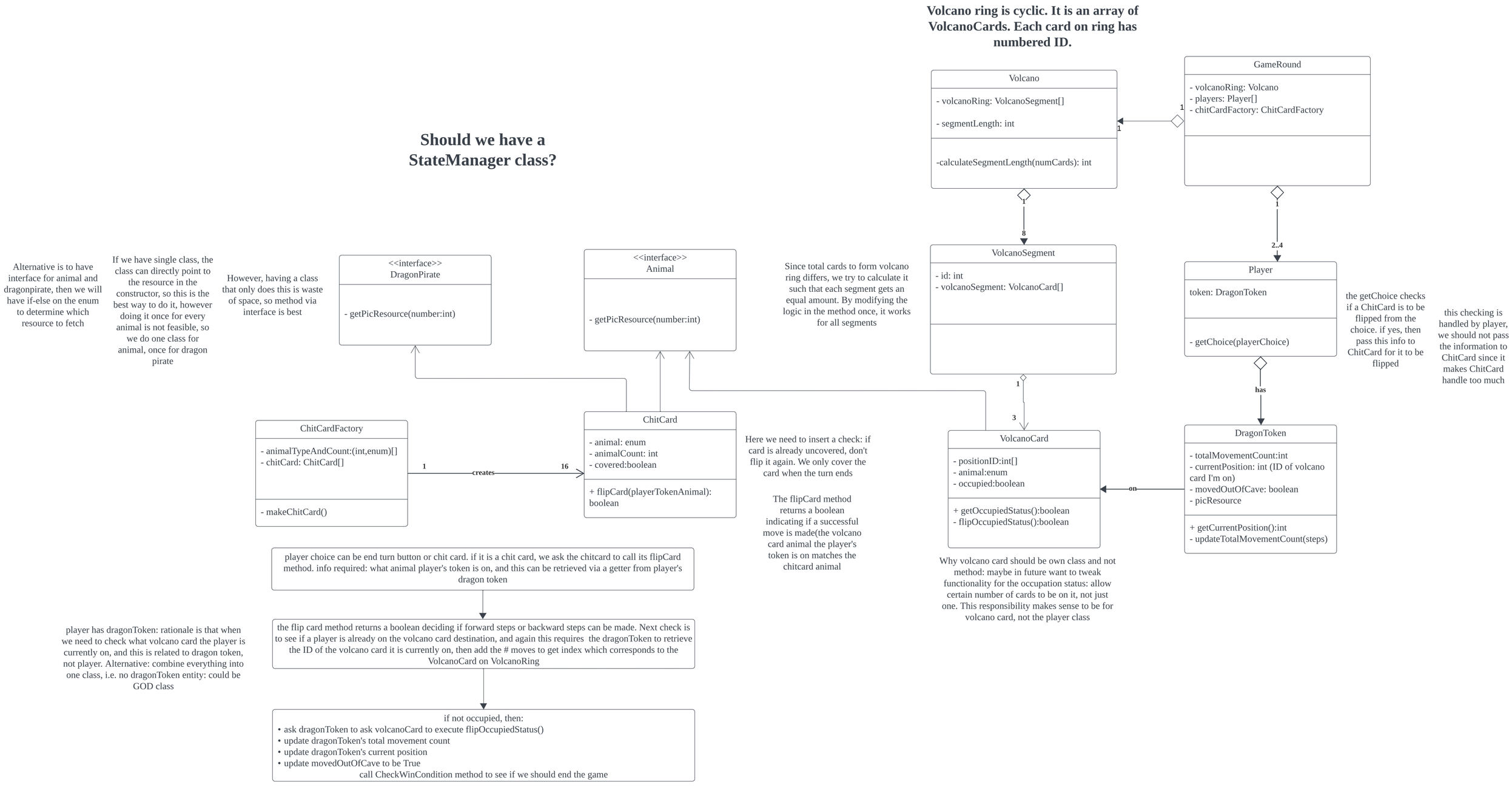
Volcano ring is cyclic. It is an array of VolcanoCards. Each card on ring has numbered ID.
Should we have a StateManager class?
Volcano
- volcanoRing: VolcanoSegment[]
- segmentLength: int
-calculateSegmentLength(numCards): int
GameRound
- volcanoRing: Volcano
- players: Player[]
- chitCardFactory: ChitCardFactory
1
1
1
8
1
2..4
VolcanoSegment
- id: int
- volcanoSegment: VolcanoCard[]
Player
token: DragonToken
- getChoice(playerChoice)
Since total cards to form volcano ring differs, we try to calculate it such that each segment gets an equal amount. By modifying the logic in the method once, it works for all segments
the getChoice checks if a ChitCard is to be flipped from the choice. if yes, then pass this info to ChitCard for it to be flipped
this checking is handled by player, we should not pass the information to ChitCard since it makes ChitCard handle too much
Alternative is to have interface for animal and dragonpirate, then we will have if-else on the enum to determine which resource to fetch
If we have single class, the class can directly point to the resource in the constructor, so this is the best way to do it, however doing it once for every animal is not feasible, so we do one class for animal, once for dragon pirate
However, having a class that only does this is waste of space, so method via interface is best
<<interface>>
DragonPirate
- getPicResource(number:int)
<<interface>>
Animal
- getPicResource(number:int)
1
has
3
ChitCardFactory
- animalTypeAndCount:(int,enum)[]
- chitCard: ChitCard[]
- makeChitCard()
1
creates
16
ChitCard
- animal: enum
- animalCount: int
- covered:boolean
+ flipCard(playerTokenAnimal): boolean
Here we need to insert a check: if card is already uncovered, don't flip it again. We only cover the card when the turn ends
The flipCard method returns a boolean indicating if a successful move is made(the volcano card animal the player's token is on matches the chitcard animal
VolcanoCard
- positionID:int[]
- animal:enum
- occupied:boolean
+ getOccupiedStatus():boolean
- flipOccupiedStatus():boolean
on
DragonToken
- totalMovementCount:int
- currentPosition: int (ID of volcano card I'm on)
- movedOutOfCave: boolean
- picResource
+ getCurrentPosition():int
- updateTotalMovementCount(steps)
Why volcano card should be own class and not method: maybe in future want to tweak functionality for the occupation status: allow certain number of cards to be on it, not just one. This responsibility makes sense to be for volcano card, not the player class
player choice can be end turn button or chit card. if it is a chit card, we ask the chitcard to call its flipCard method. info required: what animal player's token is on, and this can be retrieved via a getter from player's dragon token
the flip card method returns a boolean deciding if forward steps or backward steps can be made. Next check is to see if a player is already on the volcano card destination, and again this requires the dragonToken to retrieve the ID of the volcano card it is currently on, then add the # moves to get index which corresponds to the VolcanoCard on VolcanoRing
player has dragonToken: rationale is that when we need to check what volcano card the player is currently on, and this is related to dragon token, not player. Alternative: combine everything into one class, i.e. no dragonToken entity: could be GOD class
if not occupied, then:
• ask dragonToken to ask volcanoCard to execute flipOccupiedStatus()
• update dragonToken's total movement count
• update dragonToken's current position
• update movedOutOfCave to be True
call CheckWinCondition method to see if we should end the game

Object Oriented Design and Design Rationales

- Class Diagram(including attributes, methods, cardinalities)
- Sequence diagrams (setup initial board, all functionalities: flipping chit cards, move dragon token, change of turn to next player, winning game)
- Design rationales
    - 2 classes (why not methods)
    - 2 relationships (why aggregation not composition)
    - Inheritance why used or not used
    - 2 cardinalities
- Minimum 3 Design patterns used(or why not used)

Video demonstration (Add timestamps for references to game rules)

Talking points:

- Setup of board. All chit cards covered, indistinguishable. The volcano cards segments are random. Message indicating player's turn.
- Flipping chit card (explaining current animal versus chit card animal, dragon token movement). Encounter mismatch, so go to next player's turn (see displayed message)
- Winning the game when reach cave
- If have time
    - cannot go back further from cave, i.e. only move back further if have moved out of cave
    - if destination exceeds the initial cave position, turn ends
    - cannot use a dragon card that has already been flipped