**Design rationale**

**Overall UML Class Diagram**

Insert UML Class Diagram here once we are all done

**Status Enum Class**

This class is created to keep track of the dinosaurs’ statuses, such as pregnant, baby and so on. This is used by the dinosaur class, and its subclasses. Could be included inside the dinosaur class itself, or split into multiple enum classes. This class can also be used to add statuses to grounds, like TALL or SHORT for bushes and trees, reducing dependency on checking class type. This will help to reduce dependency on checking dinosaur class type.

**Gender Enum Class**

This enum class is used in determining dinosaurs’ genders. Is a separate enum class from Status as we need to lock what type of enum is it when creating dinosaurs. Using this will be much clearer than using Boolean to indicate female or male, making it easier to understand.

**Dinosaur Abstract Class Extends Actor**

The class was created to have a generalisation of what things are to be processed each turn for the dinosaur (playTurn() method), such as hunger(HP), age, pregnancy, as well as going through the behaviour ArrayList to find an action to do. When extended, the extended class should call its parent’s playTurn() method, reducing repeated code. This class should have various helper methods that will help in playTurn() such as checking how long it was unconscious, to avoid cluterring up playTurn(). Would also have many attributes, and static final attributes that help in playTurn() and other classes, such as age, pregnantAge, unConsciousTime, HUNGRY\_INT, CORPSE\_ROT\_TIME. Will also have an ArrayList of behaviour that is used in playTurn(). The constructor of this class should initialise the age and gender, and add WanderBehaviour, and more can be added to the front of the ArrayList in extended classes, to show more behaviours in a dinosaur. There should be two constructors where one is an one parameter constructor that takes in gender from the status enum class, and will add the gender enum(MALE or FEMALE) into capabilities depending on the gender, and automatically initialises age to the adult age. The other constructor will take in age and gender (or just age and randomise the gender), and initialises the dinosaur normally, with an inputted age, as well as gender.

When extending this class, many of the attributes and methods will be inherited, thus reducing repeated codes. Each dinosaur is still of type Actor, in which the engine will still process, and doesn’t increase dependency.

Any subsequent classes that wish to have its own max HP, starting HP, etc can just simply override the final static attribute, and all other code will still use the same attributes instead of a number that needs to be updated everywhere else.

**HerbivoreDinosaur Abstract Class Extends Dinosaur**

This class is created so that we can generalise the herbivore food behaviour. This class will extend its parent’s constructors and add the necessary behaviours, BreedBehaviour and HerbHungerBehaviour in the constructor, to give HerbivoreDinosaurs breeding and feeding behaviour which simulates the dinosaur’s breeding and feeding. This class is here just in case we may want to introduce more Herbivore Dinosaurs, and when we do, it will reduce repeated codes.

**CarnivoreDinosaur Abstract Class Extends Dinosaur**

This class is created to generalise Carnivore food behaviour. This class will extend its parent’s constructor and add BreedingBehaviour and CarniHungerBehaviour which simulates the dinosaurs breeding and feeding. This is created in case we may want to introduce more Carnivore Dinosaurs, and when we do, we will reduce repeated codes. Will override getAllowableActions and check if the other actor (Must be a player) has suitable food (Carnivore Kit) to feed to this dinosaur. This is done by looping through the other actor’s inventory.

**Stegosaur Extends HerbivoreDinosaur**

This class is used to represent a Stegosaur. Should have its own starting HP, max HP, adult age that is unique to Brachiosaur. All of the necessary behaviours are already in the parent’s code. The constructor should add a short\_neck enum which will be used in HungryHerbivoreBehaviour. Default constructor should call super and initialise the Stegosaur with its starting HP and age. Will initialize final static attributes inherited from Dinosaur. Will override getAllowableActions and check if the other actor (Must be a player) has suitable food (Herbivore Kit and Fruit) to feed to this dinosaur. This is done by looping through the other actor’s inventory.

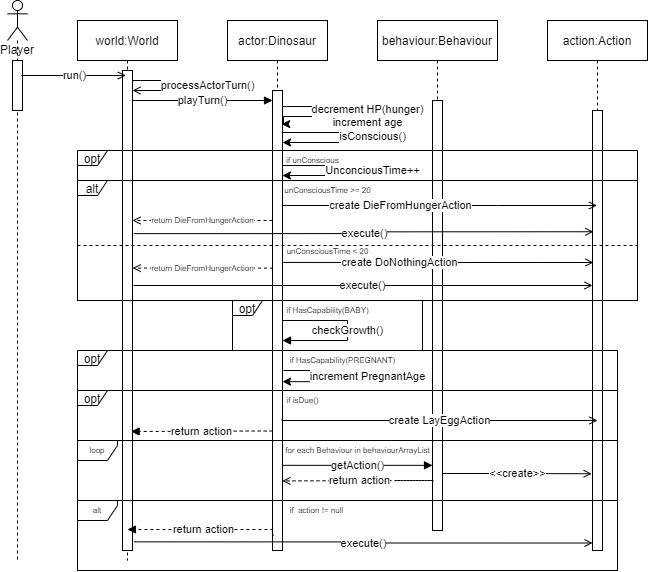
**Brachiosaur Extends HerbivoreDInosaur**

Used to represent a Brachiosaur. Should have its own starting HP, max HP, adult age that is unique to Brachiosaur. All of the necessary behaviours are already in the parent’s code. The constructor should add a long\_neck enum which will be used in HungryHerbivoreBehaviour. Default constructor should call super and initialise the Stegosaur with its starting HP and age. Will initialize final static attributes inherited from Dinosaur.

**Allosaur Extends CarnivoreDinosaur**

Used to represent an Allosaur. Constructor will add PredatorBehaviour to the front of the behaviour ArrayList to simulate the Allosaur attacking adjacent Stegosaurs. Will have a hashmap to keep track of the Stegosaur it has attacked. playTurn() is overridden, and we will loop through the hashmap and increment the turns elapsed since Stegosaur attacked in it. If it’s more than 20, we will remove it, then we call super’s playTurn(). Will initialize final static attributes inherited from Dinosaur.

**General Interaction Diagram of Dinosaurs**



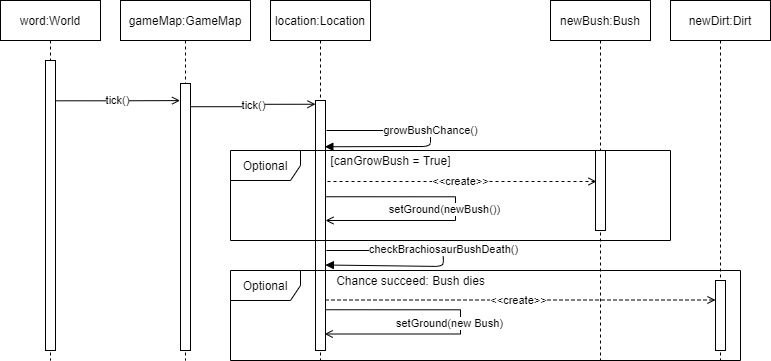
**JurassicParkLocation Extends Location**

Used to represent a location in our Jurassic Park game. This is used so we can check whether a bush can grow in the tick() method, as well as implementing the chance a bush dies if there’s a brachiosaur on it. This is the same as how the Conway demo implemented it. Due to polymorphism, other methods that expects Location will still work since JurassicParkLocation is a location, thus not increasing dependency.

**JurassicParkGameMap Extends GameMap**

Used to represent a GameMap in our Jurassic Park game. This is used so we create a JurassicParkLocation instead of a standard location. This is the same as how the Conway demo implemented it. The game engine will still expect a normal GameMap but due to polymorphism this is still accepted and will work, thus not increasing dependency.

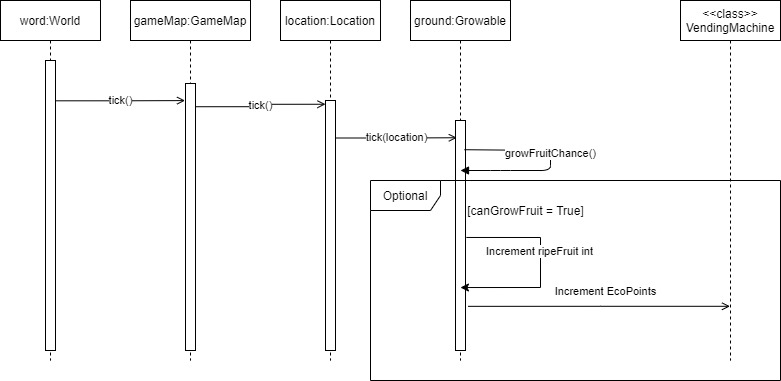
**Interaction Diagram of Location**

****

**Growable Abstract Class extends Ground**

Used to represent anything that can grow fruit. We will override tick() so we can add a check to see if we can grow fruit. This class will have a final static variable to store the chance of growing the fruit and the check will use that variable. By doing this, we can easily extend the class to add things like bushes and trees that grow fruits, where they can override the growth chance, and the chance is still passed over to the parent Growable class when using super’s tick() to check whether a fruit will grow, reducing repeated code. The class should also keep track of how many ripe fruits are there (integer). Should have helper methods for the number of ripe fruits.

**Interaction diagram of Growable**



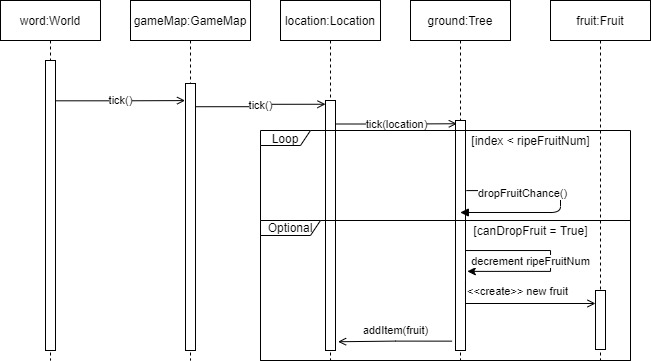
**Bush extends Growable**

Represents a bush. Would just call super’s constructor to indicate its character on the map. Everything here should be done in the parent’s code already. The only thing to override is allowableActions, in which we would add an EatFromBushAction (If actor has short neck, shouldn’t be added anyway and should be in behaviour) and PickFruitAction (If actor is the player). The grow chance should be set to something like 0.1, which when tick() is called, will be used in chance calculation. Should add an enum SHORT in capability which can be used in behaviour to check whether a dinosaur can eat from this.

**Tree extends Growable**

Represents a tree. Calls super constructor to indicate its character. This class will add a final static dropFruitChance. The tick() method is overridden and we will call super’s tick() to run the fruit growth chance, and run something like a dropFruitCheck(), which takes dropFruitChance and sees if a fruit will drop, by looping through numberOfRipeFruit. Also will have age and will increment age and change the character on the map. The allowableAction is also overridden and adds pickFruit (If is player) and eatFromTreeAction (If has long neck, shouldn’t be added anyway and should be in behaviour). The grow chance would also be set here, like 0.5, and will be used in super’s tick(). Should add an enum TALL in capability which can be used in behaviour to check whether a dinosaur can eat from this.

**Interaction Diagram for Tree**



**Design Rationale for Items and Vending Machine**