**FIT2099 Assignment 1**

**Tutorial 9 Team 9**

**Prepared by:**

**Joanne Ang Soo Yin**

**Nelly Tay Yee Ting**

**Jonathan Wong Leong Shan**

**DinoActor - extends Actor**

All dinosaurs share a lot of common characteristics, such as feeding, breeding etc.

Hence, to adhere to the *Don’t repeat yourself* principle and improve maintainability, we will create an abstract class DinoActorthat inherits from Actor. In turn, all Actors which are dinosaurs, namely *S*tegosaur, Brachiosaur and Allosaur, will inherit from DinoActor*.* These three classes will inherit all methods from dinosaurs and only override the getAllowableActions() method or the playTurn method if necessary

New attributes with appropriate setters and getters will be added to this base class for simulating the dinosaur Actor’s functionality, which will be specified in the corresponding sections later for clarity.

However, there are two important methods of DinoActor that will be highlighted. As rule of thumb, the roles of the two methods will be as such:

1. getAllowableActions method

Returns Actions that mimic interactions between twoActors on adjacent squaresonly. This is because this method will only be called every turn when another actor is on adjacent squares to the other actor. In this situation, we **benefit from polymorphism** by **overriding this method in DinoActor’s child classes to return an Actions object that contains BreedingAction and / or AttackAction only (by calling getAction on BreedingBehaviour and AttackBehaviour classes respectively).**

Note that if both BreedingAction and AttackAction are added to the Actions object, it will always be in the order BreedingAction first, then AttackAction - its importance will be explained in the playTurn method.

1. playTurn method

When the playTurn method is called for the dinosaur actors, the playTurn method in their respective classes will call the base class DinoActor’s playTurn method.

The playTurn method in DinoActor has the responsibilities to:

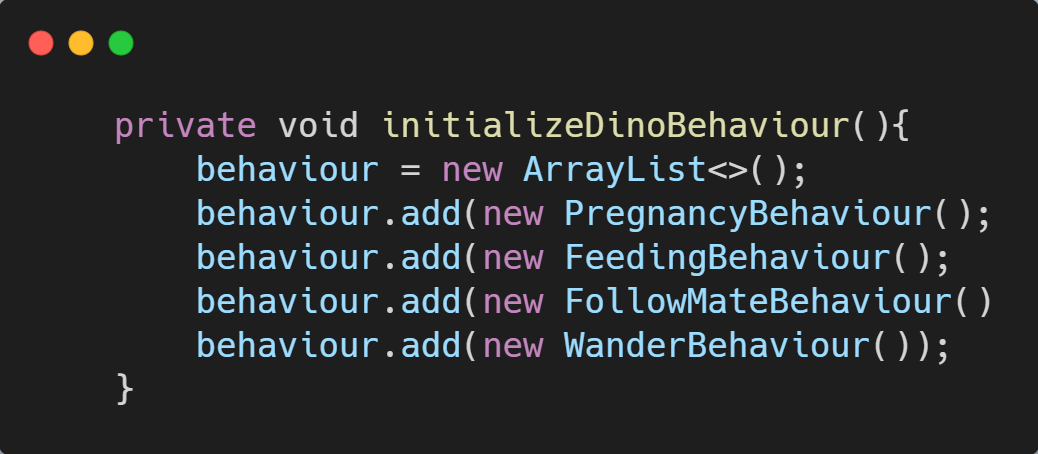
* Update the state of Actor: increment age, decrement food level, display hungry message
* Given all possible actions the Actor can take, determine and return the actual action taken by the NPC

In order to fulfill the second responsibility, i.e. returning the actual action taken by the NPC, we will create classes that implement Behaviour interface:

* **Behaviour-implementing classes** are **utilized for doing necessary processing of Actor state** and **reduces clutter in the playTurn method,** by having all the condition checking statements (of whether an Action can be done by an Actor) inside the getAction of a Behaviour. An action is returned when calling the getAction method if all conditions are satisfied, otherwise null is returned.

Implementation of the decision making on which action to return will be as such:

* Whenever creating a new instance of DinoActor, Initialize the ***behaviour***attribute with Behaviour-implementing classes in an decreasing order of priority, i.e: first one to be added is the most important. The following example is the priority we have decided on in the preliminary design:



Then, In the playTurn method of DinoActor:

1. Receives *actions* argument, which in the case of dinosaurs will only have BreedingAction and / or AttackAction. As mentioned above, we will always have BreedingAction as the first Action if there are two Action objects in the *actions* argument.
2. Loops through all Behaviour objects in the behaviour call getAction, but only stores the first non-null outcome in a local variable. Since our behaviours attribute is initialized based on a priority, it will help us to get the Action of highest priority in behaviours.

Note that it is important to loop through the *behaviour* attribute and call getAction for each and every one, since we can do some necessary processing of an Actor’s state. For example,

* + If the Actor is pregnant but not due to lay egg yet, in the getAction method it will help update the number of turns the Actor has to wait, then return null

1. Decide which is the actual Action taken
   * If the actions argument received has at least one Action, the first action is chosen. (refer 1)
   * Otherwise, the first non-null result that we obtained from looping through all behaviours is chosen. (refer 2)
   * If the *actions* argument has no Action objects and all Behaviour-implementing class in *behaviours* return null when getAction() is invoked on them, return DoNothingAction.