Recommendations for extensions to game engine

I would like to recommend EndGameAction for the game engine. In current sandbox mode, this game could not be ended by any way unless the player forces to terminate the program. With this EndGameAction added to the game engine, the player has an option to quit himself from the game. EndGameAction extends from the Action class which if the actor is an instance of a player, the player will be removed from the game map. When there is no player on the map, the game will come to an end. The option to end the game will be displayed to the user through the menu. Advantage of this change as mentioned before, player has an option to end the game instead of forcing the program to halt. This is a potential feature for rpg game type clients as most of the rpg games needs user to complete the game in order to end the game. Hence, an EndGameAction would provide the users to have an option to end the game instead to force stop the game.

Next, I recommend buyItemAction. This action extends from action class which allows players to purchase items from the game. The current engine does not have features to allow player to make any business deals in the game. For example, in this assignment, if player has enough ecoPoints to buy the desired item, item will be added to player's inventory while their ecoPoints get deducted according to the price of item. If player does not have enough ecoPoints, system will alert the player that he does not have enough of ecoPoints. buyItemAction will be an option for the player whether to spend their ecoPoints or not. Not only buy item, sellItemAction can also be created in this engine by extending from action class also. Implementation for sellItemAction is mostly same as buyItemAction except instead of deducting their ecoPoints to purchase an item, the player now will earn ecoPoints by selling an item to the game. This feature is crucial as it will reduce the redundancies of code and it is

clearer to the developer. Developer just need to simply have a buyer, a sell and this action to perform the interaction.

In this assignment, dinosaur class is created in the engine as an abstract class which extends from actor class. Stegosaur, Allosaur, Archaeopteryx and Agilisaurus share some common methods and properties. Designing a parent class will obey the DRY(Don't repeat yourself) principle and reduce code redundancies. This logic goes to food class as well. Instead of having setter and getter in each type of food, it has been implemented in the parent class (Food) and the child classes (Different type of food) will inherit from it.

There are many child classes such as buyItemAction, crossMapsAction,DoNothingAction, DropItemAction, EndGameAction, feedDinosaurAction, harvestGrassAction and laserStegosaurAction extend from the action class as each action has its own responsibility. If those actions are to put it into a class, the code will be hard to maintain. If an error occurs, it will be easily to be found out. This kind of implementation in this project obeys the Single Responsibility Principle. As stated by Liskov Substitution Principle, for a class S to be true subtype of T, then S must conform to T. A class S conforms to class T only if an object of class S can be provided in any contract where an object of class T is expected, and correctness is preserved. While obeying this principle, classes in this assignment maintain a correct inheritance hierarchy. For example, we have 4 types of different dinosaur which inherits from Dinosaur class. Each dinosaur has different properties such that some are herbivorous, carnivorous and omnivorous. If dinosaur dies, carcass is created which is then inherited by Food Class.