Assignment 2
FIT2099
Object Oriented Design And Implementation

Done by:
Sutulova Tatiana 30806151
Elaf Abdullah Saleh Alhaddad 31063977

Work Breakdown Agreement

Our team consists of two team members, which makes it easier to equally separate the workload. Most of the work will be done during the Zoom meetings to ensure that the time spent by both students to complete given tasks is equal and to be able to solve any issues and problems that the student may face, while completing their parts, is discussed and solved immediately.

General task	Small tasks	Deadline	Student
Implementation of the functionality	Dinosaur, Allosaur, Stegosaur classes	30/09/20	Tatiana
	FoodItem, PortableItem classes	30/09/20	Tatiana, Elaf
	EggItem, AllosaurEggs, StegosaurEgg classes	30/09/20	Elaf
	Mealkit, CarnivoreMealkit, VegetarianMealkit	30/09/20	Tatiana
	HayItem, FruitItem	30/09/20	Tatiana
	LaserGun	30/09/20	Elaf
	VendingMachine, BuyItemAction, BoughtItem, Player	07/10/20	Tatiana

	Tree, GrassDirt	07/10/20	Elaf
	SearchTreeAction, HarvestGrassAction, GrazingGrassAction	07/10/20	Elaf and Tatiana
	AttackAction, CorpseItem	12/10/20	Elaf
	BreedingBehaviour, BreedingAction, FollowBehaviour	14/10/20	Elaf and Tatiana
	HungryBehaviour, FollowFoodBehaviour, Feedaction, EatingAction	15/10/20	Elaf and Tatiana
Changes to existing documentation	Class Diagram	16/10/2020	Tatiana
	Sequence Diagrams	16/10/2020	Elaf
	Documentation	16/10/2020	Tatiana
Finalising	Testing	16/10/2020	Elaf
	Commenting	16/10/2020	Tatiana
Submission	Submit all the required files	16/10/2020	Elaf, Tatiana
Sutulova Tatiana 30806151	I agree with the WBA	Signature:	

Elaf Abdullah Saleh Alhaddad 31063977	I agree with the WBA	Signature:

Content

1. Description of new classes:

1.1 Vending Machine

Class Name	Description	Interaction with the existing system	Planned attributes and methods	Implemented methods and attributes
VendingMachine	Sells items to a Player, which may be used by them for different purposes. This includes: • HayItem (20 ecoPoints) • FruitItem (30 eco points) • VegetarianMealKitItem (100 ecoPoints) • CarnivoreMealKitItem (500 ecoPoints) • StegosaurEgg (200 ecoPoints) • AllosaurEgg (1000 ecoPoints)	• **VendingMachine** is a child class of the already existing class in the system called **Ground.** • It cannot be moved, removed or stepped on by the actors. This will be implemented using the existing method (**canActorEnter()**). • The **Player** can interact by paying ecoPoints and receiving a desired item in their	*Methods:* • **getRequiredPoints(item: Item)** - returns the required eco points for the player to be able to buy this item • **getItem(item: Item)** - returns the item the player buys	*Methods:* • **getRequiredPoints(item: Item)** • **canActorEnter(Actor actor)** added to ensure that user cannot step • **blocksThrownObjects()** added to ensure that user cannot place any items on the vending machine • **allowableActions(..)** is needed to allow the BuyItemAction, which is necessary

	● LaserGun (500 ecoPoints)	inventory. This process is shown in details in the **BuyItemAction**-sequence diagram. ● The **Item** class and **VendingMachine** class are in association relationships, since **VendingMachine** creates an **Item**.		for the user to buy a desired item ● **getItem(item: Item)** is replaced with **menuVendingMachine( )** which allows the player to choose the Item to buy and returns the chosen Item

1.2 Eggs

Class Name	Description	Interaction with the existing system	Planned attributes and methods	Implemented attributes and methods

<>
Egg
	● The **Egg** class cannot be instantiated because it is an abstract class.
	● This abstract class has to children classes:
	  ○ **StegosaurEgg**
	  ○ **AllosaurEgg**
		● It is a child class from the already existing **FoodItem** class.	*Attributes:*
			● **age**: which starts at 0, when the egg is laid and ends at 10, after which the egg will hatch and the
				Attributes:
				● **age**
				● **turns**
				
				Methods:
				● **tick()** methods
				● **getDropAction()**

| | | | Dinosaur object will be instantiated
● **turns**: which starts at 0 and increases by 1 every turn when it is in Dinosaur inventory (before being laid)

Methods:
● **tick()**- methods which are used to update age and turns
● **getDropAction()**- which is used by the Dinosaur to lay an Egg.
● **hatchEgg()**- which initializes a new **Stegosaur** /**Allosaur** object and deletes the **StegosaurEgg**/**Allosaur Egg** object. | ● **hatchEgg()**
● **increasePlayerPoints()**-Which is needed to increase the eco points of players when the egg hatches |

| StegosaurEgg AllosaurEgg | The egg is created (instantiated) if:
+ Dinosaurs of the same kind breed in the female dinosaur inventory
+ Bought from the **VendingMachine** by the **Player** | • Children of the **EggItem** class | • constructor only | • constructor only |

1.3 Food Items

Class Name	Description	Interaction with the existing system	Planned attributes and methods	Implemented attributes and methods
<> **BoughtItem**	• The **BoughItem** class allows to assign price in eco points to the items, which will make the interaction with the **VendingMachine** possible	• **BoughtItem** is a child class of the **PortableItem** and the superclass of **FoodItem**	The following class was not planned, but it is necessary to allow adding the price to Item, without changing the Item class, which may change the implementation of	*Attributes:* • **priceEcoPoints** price of the item in the vending machine. *Methods:* **getPriceEcoPoints()** - method which returns the price of the **BoughItem** in eco points

			Items that are not supposed to be bought from the vending machine	
<> **FoodItem**	● The **FoodItem** class allows to assign food points to the items, which will make it possible to increase the food level of dinosaurs	● It is the parent of items that are explicitly used as food for one of the dinosaurs. This includes: - **MealKitItem** - **FruitItem** - **HayItem** - **EggItem**	*Attributes:* ● **foodPoints**: the points that will increase the **foodLevel** of the dinosaur when it eats the **FoodItem** *Methods:* ● **getName()**- methods which returns the name of the food ● **getFoodPoints()** - method which returns the **foodPoints** of the **FoodItem**	*Attributes:* ● **foodPoints** *Methods:* ● **getName()** ● **getFoodPoints()**
<> **MealKitItem**	● The **MealKitItem** class cannot be instantiated because it is an abstract class.	● It is a parent of the items that are explicitly Meal kits	● constructor only	● constructor only

		for the dinosaurs. This includes:
		    ○ **Carnivore MealKitItem**
		    ○ **Vegetarian MealKitItem**		

CarnivoreMealKitItem	It is used to feed the **Allosaur,** increasing its foodLevel to maximum.	● The Superclass for it is the already existing **MealKitItem** class ● Can be bought from the **VendingMachine** that will be implemented in the system using the **player's ecoPoints.**	● constructor only	● constructor only
VegetarianMealKitItem	It is used to feed the **Stegosaur,** increasing its foodLevel to maximum.			
FruitItem	● It is used to feed **Stegosaur** and increases its **foodLevel** by 30. ● **FruitItem** can be bought from the **VendingMachine**	● It is a child class of the **FoodItem** class.	*Attributes:* ● **turns**: which starts at 0, when the Fruit is dropped on the ground and changes every	*Attributes:* ● **turns** *Methods:* ● **tick()**

	using the **Player's ecoPoints** or dropped by the **Tree** ● It can be stored in the **Player's** inventory without rotting. ● The **Player** has the ability to search for fruit on a **Tree** and has the probability of 40% to find a ripe one. This process is shown in the "searching for Fruit" sequence diagram. ● **Tree** has a possibility of 5% to drop a ripe **FruitItem** on the ground and it can be picked from the ground by the **Player**. ● This can be done by using the **getDropAction()** and **getPickUpAction()** methods inherited from the **Item** class		turn, till it reaches 20. It will then rot and be removed from the ground using the **remove(item)** method in the **Location** class *Methods:* ● **tick()** -which is used to update turns only when the **FruitItem** is on the ground. It is inherited from the **Item** class.

HayItem	• It is used to feed **Stegosaur** and increases its **foodLevel** by 20. • **HayItem** can be bought from the **VendingMachine** (association) using the **Player's ecoPoints** • **HayItem** can be harvested from the grass by the **Player**. This process is shown in the "Harvesting grass" sequence diagram	• It is a child class of the **FoodItem** class.	• constructor only	• constructor only
CorpseItem	• It appears on the ground if one of the dinosaurs dies.	• It is a child class of the **FoodItem** class.	This class was not planned, but is necessary to implement it to allow the **Allosaur** to eat the dead dinosaur.	• constructor only

1.4 Weapon

Class Name	Description	Interaction with the existing system	Planned attributes and methods	Implemented attributes and methods
LaserGun	- It is used by the **Player** to kill a **Stegosaur** with the help of the already existing class **AttackAction.** - It has a **damage** of 50, which is enough to kill the **Stegosaur** in one or two shots. **LaserGun** can be bought from the **VendingMachine** (association) using the **Player's ecoPoints**	It is a child class of **WeaponItem** and implements the **Weapon** interface	constructor only	*Attributes:* **PRICE_ECO_POINTS** - attribute, which is needed to buy the **LaserGun** from the vending machine, since it does not inherit the **BoughtItem** *Methods:* **getPriceEcoPoints()** is a getter for the price of the laser gun, needed to access the price outside of the **LaserGun** class

1.5 Dinosaur

Class Name	Description	Interaction with the existing system	Planned attributes and methods	Implemented attributes and methods
<> **Dinosaur**	- An abstract class that represents the dinosaurs that will be implemented in the system - To create a new Dinosaur (any kind): - The **Player** can buy an **Egg** from the **VendingMachine,** place it at any **Location** and wait for 10 turns for it to hatch. - Dinosaurs of opposite genders and same species may breed and after 10 turns the female Dinosaur lays eggs that will hatch after 10 rounds. The	It will be a parent class for the **Stegosaur** and **Allosaur** classes.	*Attributes:* **foodLevel**: It decreases by 1 every turn. **gender**: String that represents the gender of the dinosaurs. **age**: integer that represents the age of the dinosaur. **is_alive**: true, false **turn**: starts at 0 and will only be updated when the dinosaur is unconscious due to hunger.	*Attributes:* **foodLevel**: **gender**: **age**: **is_alive**: **turn**: **behaviours**: this is an array, which is needed to store the behaviours that are available for the dinosaur **hitPoints:** this an integer that is implementing 'life points' of the Dinosaur, it is needed to make the Attack action possible *Methods:* **isConscious()** **playTurn()** **eat(food)**

	breeding process is shown in the "BreedBehaviour" sequence diagram. ● After the **Egg** hatches, the dinosaur is born with the **age** of 10 and **foodLevel** of 10 ● Both dinosaurs can be fed by the **Player** with suitable food. ● Both dinosaurs can express **HungryBehaviour** that is explained in the "HungryBehaviour" sequence diagram for both of the dinosaur		_Methods:_ ● **isConscious()**- inherited from the Actor class. We plan to add the additional functionality to make the Dinosaur unconscious if the food level is zero. ● The **playTurn** method will be updated to make sure that if the Dinosaur stays unconscious for 20 turns it will die. This process can be done by incrementing the **turn** attribute and having an if statement to change the	● **getFoodLevel()** ● **getAge():** needed to retrieve the age of the dinosaur from outside of the class. It is created since the age has to be checked for the dinosaur to breed ● **getGender():** added to count for how many turns the dinosaur stays unconscious. Needed to make sure the dinosaur dies after 20 turns of being unconscious ● **getGender():** needed to retrieve the gender of the dinosaur from outside of the class. It is created since the gender has to be checked for the dinosaur to implement the BreedingBehaviour ● **setGender():** randomly sets the gender to the dinosaur, when it is

			is_alive to false when the **turn** = 20. ● **eat(food)** - increases the **foodLevel** when the dinosaur eats, if the dinosaur is not full ● **getFoodLevel()** - method that returns the **foodLevel** of the dinosaur ● **breed(dinosaur)**- method that allows the dinosaurs to breed.	created. Gender is needed to implement BreedingBehaviour ● **increaseAge():** added to increase the age ● **decreaseFoodLevel():** implemented to decrease the food level of the dinosaur on every turn. ● **getAllowableActions():** necessary to make the attack actions possible to implement on the following dinosaur. *Methods removed:* ● **breed(dinosaur),** since the BreedingAction is implemented as another class.

Stegosaur	• **Stegosaur** is a herbivore and implements only vegetarian behaviours: eats hay, grass, leaves and fruits, does not kill and attack. • Has **foodLevel** of 50 for a **Stegosaur** by default when the game starts • can be killed by the **Player** using the **LaserGun** with the help of an already existing class called **AttackAction**.	This class is an already existing class in the system. However, it will be changed completely due to it becoming a child class to a newly implemented class hence it is included here.	*Methods:* • **playTurn()** that decides what the dinosaur does based on certain conditions. Stegosaur has an implementation of the special **GrazingGrassAction** method.	*Methods:* • **playTurn()**
Allosaur	• **Allosaur** is a carnivore and implements only carnivore behaviours: eating dead Dinosaurs and eggs, attacks the Stegosaur if it is nearby. • **Allosaur** will attack a nearby **Stegosaur** with the help of an already existing class called **AttackAction**.	• It is a children class of the **Dinosaur** abstract class.	*Methods:* • **playTurn()** that decides what the dinosaur does based on certain conditions. Allosaur has an implementation of the special **AttackAction** method.	*Methods:* • **playTurn()**

1.6 Actions and Behaviours

Newly implemented Action classes include BuyItemAction, FeedAction, HarvestGrass and *SearchTreeAction, HarvestGrassAction, GrazingGrassAction, EatingAction, BreedingAction (cursive classes were not planned at first, reasoning of adding them is explained in the sequence diagrams)*. Newly Implemented Behaviours are : HungryBehaviour and BreedBehaviour. Due to the complicated nature of these new actions and behaviours we created different sequence diagrams to explain how they will be implemented. The sequence diagrams show in detail how they are connected to the existing system and the newly implemented system. To avoid repetition, we decided not to include these details inside this documentation.

2. Changes to existing classes

Class Name	Description	Attributes and methods that were planned to add	Attributes and methods that are actually added
Player	- Adding **ecoPoints** that are used to buy Items from the **VendingMachine**. **EcoPoints** are increased if: 1 when Grass grows/harvested: - When the "." changes to "g" - When the "g" changes to "." and the HayItem appears in the Player inventory 10 when Dinosaur is fed with hay 15 when Dinosaur is fed with Fruit 100 when StegosaurEgg hatches: - When the Egg age reaches 20 1000 when AllosaurEgg hatches - When the Egg age reaches 20	*Attributes:* • **ecoPoints:** The player can use to buy items from vending machine *Methods:* • **getEcoPoints()**- method that returns the ecoPoints of the player • **payEcoPoints(points : int)**- method that decrement the ecoPoints based on what the Player bought • **earnEcoPoints(points)** - increments the **ecoPoints** of the **Player**	*Attributes:* • **ecoPoints:** *Methods:* • **getEcoPoints()** • **payEcoPoints(points : int)** • **earnEcoPoints(points)**

- **Player** will have the ability to buy Items from the **VendingMachine** using these **ecoPoints** with help of the class **BuyItemAction**.
- **Player** will have the ability to Feed the dinosaurs with help of the class **FeedAction**. When the dinosaur is fed, the function will update the **Player's ecoPoints** based on what they fed the dinosaur.
- **Player** will have the ability to harvest grass with the help of the class **HarvestGrass**. When the grass is harvested, the function will update the **Player's ecoPoints**.
- **Player** will have the ability to search a tree for fruit with the help of the class **SearchTreeAction**.

| GrassDirt | Previously the **Dirt** class, with the implemented functionality of growing into grass. In the beginning of the game all the **GrassDirt** boxes are Dirt, which is represented with "." and has a low probability (2%) of growing into Grass, with display character "g". If there are no Grass cells nearby or there is a tree in the neighbour cell, Dirt has a 2% of growing into grass, but if there are 2 or more grass cells around, the Dirt has a probability of 10% of growing into Grass. This is implemented by a method, all the neighbouring cells of each cell and decides whether the grass grows or not with consideration to these probabilities. | _Methods:_
tick()- will be implemented on this function the same way it is implemented on Tree to update the dirt to grass based on the probability. | _Methods:_
tick() **getHarvestGrassAction()** necessary to allow to implement the HarvestGrassAction() **getGrazingGrassAction()** necessary to allow to implement the GrazingGrassAction() **increasePlayerPoints()** which is needed to increase the eco points of players when the grass grows or harvested |

Tree	● Will now have the ability to drop a ripe FruitItem. The probability for this to occur is 5% in each turn.	*Methods:* ● **tick()** - method will be updated to consider this new functionality of the tree ● **searchTreeAction()** allows to search the fruit on this tree	*Methods:* ● **tick()** ● **searchTreeAction()**

3. Reason for choosing this design

The main software development principle we were following in this project is Don't Repeat Yourself (DRY), which aims to reduce code and knowledge duplication. According to the DRY principle, every discrete piece of knowledge should have one, unambiguous, reliable representation within the system, which eliminates redundancy in process and logic. One example of implementation of the DRY principle in our system is creating a parent abstract class **Dinosaur** which contains all the methods and attributes, that are shared by both **Allosaur** and **Stegosaur** classes. Instead of duplicating the same code in both of these classes, the mechanism known as inheritance allows us to combine the code in the super class and inherit it to all the subclasses. The main advantages of following the DRY principle are reusability and maintainability, as if the problem occurs in the block of repeated code, it has to be fixed in every place where this block appears, whereas in the DRY code, the bug would only be fixed once.

Another important principle that we based our system on is reducing dependencies. Dependency is when one class uses another class or interface; when one class cannot carry out its work without the other and cannot be reused without reusing the other. The main disadvantages of dependencies are the decrement of reusability, which affects the speed, code quality and readability.