Documentations (Assignment 3)

Requirements

Going to the Moon

Class Added: MoveToEarthAction

Roles and responsibility:

- Represents an action that can be performed by the player whenever player is standing at the assembled rocket on the moon. This action is also performed the safety system acting on the player it detects that the player does not have enough oxygen supply or does not have spacesuit on the moon. When this action is carried out, the player will be transported back to the rocket's location on Earth. This action also checks if the player's inventory contains YugoMaxx's unconscious body. If there is, player wins the game.
- Has a field of type Location to store the reference to the rocket's location on the Earth which will be initialised in the constructor.
- Has a field of type Item to store the reference to YugoMaxx's unconscious body which will be initialised in the constructor so that we can check if the player's inventory contains this item.
- Has a field of type GameWorld to store the reference of the GameWorld created in Application that all the actors is in so the World can end the game when the player that perform this action wins (if the player's inventory contains YugoMaxx's unconscious body)
- In its execution, this action will search for YugoMaxx's unconscious body in the inventory of the Player, then if it exists, the World end the game by indicating that

player has won the game. If Player does not have its body, move the Player to the rocket's location on Earth.

How the required functionality is delivered:

- When the player is standing at the assembled rocket, they must have the option to fly to moonbase. To implement this, we add MoveActorAction to the Earth's rocket's allowableActions after constructing this rocket in the GameManagement.
- The moon's map of type GameMap of size 10x15 squares will be created in the constructor of GameManagement and this map will be added into the world by using GameWorld's addMap(). This map will then be added with all the enemies that the Earth has using the map's addActor() method.
- In GameManagement, we will have a method enableTransportBetweenMaps() to allow player to move between maps. In this method, we will create a rocket for the Earth and one for the moon. Since the rocket must also appear in the moonbase map, we will then place moon's rocket on a location on the moon using the moon's additem() method. In order to allow player to move to the moonbase map, we will add MoveActorAction into earth's rocket's allowable action using getAllowableActions.add() and for the player to move from moon to the lair on Earth, we will add MoveToEarthAction into moon's rocket's allowable action. We will not use MoveActorAction when player moves from moon to Earth because MoveToEarthAction will check if the player has Yugo's unconscious body before moving the player back to the Earth's rocket location so that the player can win the game if Yugo is in the player's inventory.

Spacesuit and oxygen

Class added: OxygenDispenser

Roles and responsibility:

- Represents an oxygen dispenser that has a button that allow other actor to press, causing it to produce an oxygen tank in its location on the next turn. The button will only work when this dispenser is not producing the tank and there is no oxygen tank in the location that has not been retrieved.
- Has a static field of type OxygenTanks which store the reference to a list of non-empty oxygen tanks produced by the oxygen dispenser so that this reference can be added with the oxygen tank retrieved by the player from this oxygen dispenser.
- Has a field of type Location that stores the reference to the location of the oxygen dispenser so that when the player press the button, we can add an oxygen manager to this location. The oxygen dispenser manager will only produce an oxygen tank in its location on the next turn and it will disappear from map.
- Has a field of type boolean to indicate whether the oxygen dispenser has oxygen tank at its location. If the location has oxygen tank on it, the button of the oxygen dispenser will not work.
- Has canActorEnter() method that will return false so that it will not give actor permission to move to the oxygen dispenser's location
- Allows actor standing beside it to perform PressButtonAction and GetOxygenTankAction (if the location of oxygen dispenser has oxygen tank).

Class added: OxygenDispenserManager

Roles and responsibility:

- Represents an oxygen dispenser manager that is created when the oxygen dispenser button works after being pressed by player. It causes the dispenser to produce an oxygen tank in its location only on the next turn after player press the button. After the oxygen dispenser manager produces an oxygen tank, it will disappear from map during its turn.
- Has a field of type OxygenDispenser that stores the reference to the oxygen dispenser where the oxygen tank is produced.
- Can only perform ProduceOxygenTankAction.
- Allows other actor to press the button on oxygen dispenser, but the button will never work when oxygen dispenser manager is present because the dispenser is producing an oxygen tank.

Class added: OxygenTank

Roles and responsibility:

- Represent an oxygen tank that is produced by the oxygen dispenser after the player press the button of oxygen dispenser. Player can retrieve the oxygen tank from the oxygen dispenser once it is ready.
- Has a field of type integer that keep track of the oxygen point of a particular oxygen tank. It is initalised with 10 in the constructor as a new oxygen tank holds 10 points worth of oxygen.
- Has a method getOxygenPoint() to enable other classes to access the tank's oxygen point.
- Has a method decrementOxygenPoint which allows the safety system to decrement the oxygen tank's oxygen point by one at each turn on the moon.

Class added: OxygenTanks

Roles and responsibility:

- Represents a thin wrapper for OxygenTank to be added with oxygen tanks produced by oxygen dispenser(s). When an oxygen tank in OxygenTanks becomes empty, it will be removed from OxygenTanks.
- Has a field of type List that stores a list of oxygen tanks obtained by player from the oxygen dispenser.
- Has a method add() that add a given oxygen tank into the list of oxygen tanks.
- Has a method remove() that removes a given oxygen tank from the list of oxygen tanks

Class added: PressButtonAction

Roles and responsibility:

- Represents an action that press the button of the oxygen dispenser to cause it to summon an oxygen dispenser manager to produce an oxygen tank in its location on the next turn. The button will only work when there is no oxygen tank that has not been retrieved and also no oxygen dispenser manager that is producing the tank in the location.
- Has a field of type OxygenDispenser that stores the reference to the target to press button on. It will be initialised to the oxygen dispenser that we want to perform this action on in the constructor.

- Has a field of type Location that stores the reference to the location of the target that this action acts on. It will be initialised with the location of the oxygen dispenser in the constructor.
- In its execution, this action will check if the location of the target of this action contains any tank and also check if any actor(always be oxygen manager) is at that location. If the location of the target does not have any of those mentioned, it will create a new oxygen manager and add the actor to that location. It will then return appropriate strings accordingly.

Class added: ProduceOxygenTankAction

Roles and responsibility:

- Action that produces an oxygen tank on an oxygen dispenser, then causes the actor performing this action to disappear.
- Has a field of type OxygenDispenser that stores the reference to the target that the oxygen tank will be produced on. It will be initialised to the oxygen dispenser that we want to perform this action on in the constructor.
- In its execution, it will set the target's hasTank status to be true indicating that the target now has oxygen tank on its location. The method then removes the actor performing this action from the map and return appropriate string.

Class added: GetOxygenTankAction

Roles and responsibility:

- Represents an action that retrieves oxygen tank from oxygen dispenser.

- Has a field of type OxygenTank that stores the reference to the oxygenTank being retrieved from the oxygen dispenser.
- Has a static field of type OxygenTanks which stores a list of oxygen tanks. It is initialised to the list of oxygen tanks retrieved by player earlier.
- Has a field of type OxygenDispenser which will be initialised to the oxygen dispenser where the player get oxygen tank from.
- In its execution, it will set the oxygen dispenser that the player get oxygen tank from to be false since the oxygen dispenser now does not have any tank at its location. It then add the oxygen tank produced by the dispenser into the player's inventory as well as the field containing list of oxygen tanks and return appropriate string.

Class added: SafetySystem

Roles and responsibility:

- Class representing a safety system that keep track of player's oxygen point. When player is on the moonbase, the system will deduct one point of oxygen from player. If the player runs out of oxygen on the moon, the system transports player back to the rocket's location on Earth.
- Has a field of GameMap that is initialised to the moon's map in the constructor.
- Has a field of type MoveToEarthAction that moves player to rocket's location on earth and will check if player has won the game so that this action can be executed when safety system detects that player does not have oxygen supply or does not have spacesuit with him.
- Has a field of type item that stores the reference to the spacesuit to check if player's inventory has spacesuit.

- Has a static field of type OxygenTanks that stores a list of oxygen tanks retrieved by player from the oxygen dispenser to check if player's inventory contains any of those oxygen tanks in the list.
- Has a field called currentOxygenTank of type Oxygen Tank which represents a player's oxygen tank that is currently being decremented at each player's turn in the moon. If the current oxygen tank is empty, it will be updated to a another oxygen tank from the player's inventory (if any)
- Has retrieveOxygenTank() method that gets a non-empty oxygen tank from the list of oxygen tanks which can also be found in player's inventory.
- Has hasOxygen() method that returns true if player still has any non-empty oxygen tank in its inventory, false otherwise. It also manages OxygenTanks by updating the currentOxygenTank to be a non-empty oxygen tank found in player's inventory.
- Has a run() method containing the execution of Safety System. It first ensure that the player is on the moon map, then it proceed to check if the player still has any non-empty oxygen tank in and spacesuit in the inventory. If player's inventory does not contain either one of the objects stated, the Safety System will send player back to the Earth, else the system will decrement currentOxygenTank by 1.

How the required functionality is delivered:

- Spacesuit is an Item object which will be added to a location on the Earth lair map and player can perform PickUpItemAction to pick the spacesuit up at the location.
- OxygenDispenser extends Ground where its canActorEnter() method should be overridden to return false so that it is impassable. This is to prevent other Actor to step on it so that OxygenDispenserManager can be summoned on its location.

- To enable the Player to press the button of the oxygen dispenser on an adjacent ground by carrying out the PressButtonAction, we will override OxygenDispenser's allowableActions() so that this method returns Actions containing a PressButtonAction instead of a empty Actions. To allow player at an adjacent ground to retrieve oxygen tank from the oxygen dispenser, we will also add GetOxygenTankAction into the Actions if and only if there is an oxygen tank found at the oxygen dispenser's location.
- PressButtonAction's execute() method will also be overridden. The target in this action will be the oxygen dispenser since this action is acted on the oxygen dispenser. This method will first check if there is an oxygen tank that has not been collected or an actor by calling the oxygen dispenser's hasTank() and check if there is any actor on the oxygen dispenser's location using map.isActorAt(). In this case, oxygen dispenser manager is the only possible actor on that location of the oxygen dispenser (see later in this point). If there is an oxygen tank on the location, indicating that the oxygen tank has not been collected, or oxygen manager on that location, indicating that the oxygen dispenser is currently producing oxygen, the button does not work; otherwise, we will create a new OxygenDispenserManager and put it on the same location so that the manager will produce an oxygen tank next round. This is also why even if we have already overriden the canActorEnter() of oxygen dispenser to return false so other actors cannot enter this location, it is possible for the oxygen dispenser manager to be at the oxygen dispenser's location as we manually add it onto the location of oxygen dispenser using map.addactor().
- OxygenDispenserManager which extends Actor is created in PressButtonAction and we will override its playTurn() to return ProduceOxygenTankAction. We will also

override its getAllowableActions() to return PressButtonAction but this button will never work when oxygen dispenser manager is present because it is producing an oxygen tank.

- ProduceOxygenTankAction's execute method will be overriden where it will set the oxygen dispenser's hasTank() to be true to indicate that an oxygen tank is produced at the oxygen dispenser. It then removes oxygen dispenser manager from the map and return appropriate string.
- When the oxygen tank is successfully produced, oxygen dispenser's allowable action will then return GetOxygenTankAction where an adjacent actor can finally get oxygen tank from the dispenser. In GetOxygenTankAction, we will override execute() method so that it set oxygen dispenser's hasTank() to false and add the oxygen tank which worth 10 oxygen points of type OxygenTank produced at oxygen dispenser into player's inventory and the oxygenTanks of type OxygenTanks.
- An object of type OxygenTanks will store the references to the oxygen tanks retrieved by the player from the oxygen dispenser. The reason why we create this class is so that we can compare these references with the references to the oxygen tanks in player's inventory to check if there is still any non-empty oxygen tanks in player's inventory.
- To implement the deduction of player's oxygen point on the moon, we will have a SafetySystem that keep track of player's oxygen supply on moon. To do this, we will also create a new class called GameWorld which extends World so that the processActorTurn() will be overriden such that it runs the SafetySystem after every actor' turn is processed. Since the SafetySystem should not affect other enemies on the moonbase, in the GameWorld's processActorTurn(), we will ensure that the actor

currently being processed is player before running the SafetySystem on that actor. In SafetySystem, we will have a method run() where it first checks if the player is on the map by comparing map of the current location where the player is at with the moon's map passed in as an argument in the constructor.

  - If the player is not on the moon map, the run() method will return null to the GameWorld where this SafetySystem's run() is called, resulting in no effect to player and nothing to be printed out in the user interface.
  - If player is on the moon map, it will check whether player has any non-empty oxygen tank that can be used using the hasOxygen() method. When false is returned by hasOxygen() or player has no spacesuit, run() will send player back to the Earth using MoveToEarthAction.execute(); otherwise, deduct the oxygen point of the oxygen tank chosen, represented by currentOxygenTank, by 1.

- hasOxygen() will check whether the instance variable currentOxygenTank, which represents a oxygen tank that was used in the previous round, can continue to be used by player. If currentOxygenTank's oxygen point is 0 or not in player's inventory, this method will try to replace currentOxygenTank with a non-empty oxygen tank found in player's inventory so that it can be used this round. After that, if currentOxygenTank is null, it means that player has no more non-empty oxygen tanks in his inventory, so this method will return false.

Final boss fight

Class added: YugoMaxx

Roles and responsibility:

- Represents a Yugo Maxx that is able to wander around the map. It is also able to punch other Actor for 12 damage (same as Goon who has twice the damage of default damage).
- Has a field hasExoskeleton of type boolean which indicates whether Yugo is wearing an exoskeleton that make him invulnerable to damage.
- Has a field of type WaterPistol which stores the reference to a water pistol used by the player to squirt water onto Yugo's exoskeleton.
- Has a field of type Item that stores the reference to Yugo's unconscious body once it is defeated.
- Has getIntrinsicWeapon() to return a new IntrinsicWeapon with damage initialised with 12.
- Has destroyExoskeleton() that sets hasExoskeleton to false indicating that the exoskeleton is destroyed.
- Allows player on adjacent ground to perform AttackYugoAction if the Yugo's exoskeleton is destroyed and SquirtAction if the player's inventory contains a non-empty water pistol.
- Can perform AttackAction, SkipTurnAction and MoveActorAction(in WanderBehaviour).

Class added: Water

Roles and responsibility:

- Represents water that allows player standing beside it to perform FillPistolAction if the player's inventory contains an empty water pistol to fill the pistol with water.
- Has a field of type WaterPistol which stores a reference to the water pistol so that we can use this reference to compare with the objects in player's inventory to check if the player's inventory contains that water pistol.
- Has canActorEnter() method that will return false so that it will not give actor permission to walk on the water.

Class added: WaterPistol

Roles and responsibility:

- Represents a water pistol item, which can be filled with water and it can be used to squirt water on YugoMaxx to destroy its exoskeleton to make it vulnerable to damage.
- Has a field empty of type boolean to indicate if the water pistol is empty.
- Has a method that sets empty to false indicating that the pistol is filled with water.
- Has a method that sets empty to true indicating that the pistol is emptied.
- Has a method isEmpty() that returns true if the pistol is empty, false otherwise.

Class added: FillPistolAction

Roles and responsibility:

- Represents an action performed by player to fill a water pistol with water.
- Has a field of type water pistol which stores the reference to the water pistol that this action acts on so that we can use this reference to fill water into it.

- In its execution, it will fill the water pistol with water and return a string indicating that the water pistol is filled with water.

Class added: SquirtAction

Roles and responsibility:

- Represents an action that has 70% of squirting water onto YugoMaxx using a water pistol which can destroy its exoskeleton to make it vulnerable to damage.
- Has a field of type YugoMaxx to store the reference to the Yugo Maxx which is the subject that this action acts on.
- Has a field of type WaterPistol that represents the water pistol used by the player performing this action so that we can use this reference to empty the water of this water pistol after the execution of this action.
- In its execution, it will squirts water onto the subject at 70% chance from the full water pistol. This empties the water in the water pistol no matter it hits or misses and appropriate strings will be returned.

Class added: AttackYugoAction

Roles and responsibility:

- Represents an attack action that can be performed by the player when standing beside Yugo Maxx.
- Has a field of type YugoMaxx which represents the subject that this action acts on.
- In its execution, it will hurt the subject at 50% chance. If the subject is unconscious after being hurt, it will remove the subject from the map and replace it with its unconscious body.

How the required functionality is delivered:

- Yugo Maxx's damage can be set by overriding its getIntrinsicWeapon() to return a new IntrinsicWeapon with twice the damage of its super class. By doing so, Yugo Maxx can have the same damage as Goon, which has twice the damage of Grunt since Grunt is having the default super class's damage.
- Yugo Maxx is required to wander around the map at random, so WanderBehaviour will be added to its actionFactory in the constructor. In WanderBehaviour's getAction(), we will calculate the distance between Yugo Maxx and Player, if the Player is not on the adjacent ground to Q (distance > 1), it will return a MoveActorAction that moves Yugo Maxx to a random valid exit. If there is no exit available, null will be returned.
- The reason why we return null in getAction() when Player is on adjacent ground is because Yugo Maxx will perform AttackAction instead of wandering around in this situation. We do not need to override the playTurn() method in YugoMaxx because this is already handled in the Enemy's playTurn() (see: Assignment 2 Documentations for Enemy class).
- To add Yugo Maxx to the map to at least 10 squares away from the rocket and a water pistol item to the moon map to at least 5 squares away from the rocket, we will set their respective coordinates accordingly on the map in the GameManagement class using the map's addActor() and addItem(). Since the water pistol starts out empty, we will have a field empty in WaterPistol class and it is initialised to false in the constructor.
- To ensure that the player can't walk on water, we will override Water class's canActorEnter() to return false to indicating that it is impassable. Player can fill an

empty water pistol when standing next to water. To implement this, we will also override Water's allowableActions() to return a new collection of actions containing FillPistolAction only if the player's inventory contains an empty water pistol. FillPistolAction will then be executed calling the fillWater() method on the water pistol it is filling which will set the water pistol's field named empty to false, indicating that the water pistol is no longer empty. This field will then be used to check if the water pistol in player's inventory is empty whenever the player perform FillPistolAction or SquirtAction.

- Yugo Maxx wears an exoskeleton that makes him invulnerable to damage and its exoskeleton can only be destroyed by squirting water onto it. In order to enable the Player to squirt water on Yugo Maxx, getAllowableActions() of YugoMaxx will have to be overridden so that a new collection of actions containing SquirtAction will be returned if the adjacent player's inventory contains a non-empty water pistol. The way of executing SquirtAction is to empty the water in player's water pistol and destroy the subject's exoskeleton at 70% chance. To implement this, we will have an instance variable of type WaterPistol that stores the reference to the water pistol that will be emptied during the execution of this action, we will also have a field of type YugoMaxx in order to use this reference to destroy the Yugo Maxx's exoskeleton at 70% chance in SquirtAction's execution by calling its destroyExoskeleton() which will set the Yugo Maxx's instance variable named hasExoskeleton to false, indicating that the Yugo Maxx does not have exoskeleton now.
- In YugoMaxx getAllowableAction(), we will also return a new collection of actions containing AttackYugoAction if the Yugo Maxx does not have exoskeleton and this is indicated by the field named hasExoskeleton. This action will be prioritised over

SquirtAction by using the if-elif concept where the first if statement will return AttackYugoAction when the Yugo Maxx does not have exoskeleton. This means that as long as the Yugo Maxx does not have exoskeleton, the first if statement will always return a new collection of actions containing AttackYugoAction instead of SquirtAction even though the adjacent player has non-empty water pistol in the inventory, so the player can only squirt water onto the Yugo Maxx if and only if the Yugo Maxx has exoskeleton, otherwise it can only choose to attack the Yugo Maxx.

- The reason why we created a new class called AttackYugoAction instead of using the AttackAction given is because we want to get the reference to the unconscious body of Yugo Maxx once it is defeated so that we can implement the ending of the game where the player wins (see: Section Ending the game)..

Ending the game

Class added: GameWorld

Roles and responsibility:

- Inherits World, represents the game world, including the locations of all Actors, the player, and the playing grid. This game world can have a safety system. Player can either lose, win or quit game.
- Has a field of type SafetySystem which will be set to the safety system of this game world to the safety system given via setter
- Has two fields of type boolean to be set accordingly to indicate player win or quit, these two fields will be set to false in the constructor.
- The game world is considered to still be running if the player is still around and player has not quit and has not won the game.

Class added: QuitGameAction

Roles and responsibility:

- Represents an action that allow player to quit the game.
- Has a field of type GameWorld which stores the reference to the game world that the player is in and it is initialised in the constructor.
- During its execution, it will call the game world's quitGame() so that the game world can terminate the game.

How the required functionality is delivered:

Quit game

- A furniture item that allows player to perform QuitGameAction will be added to player's inventory. It has to be a furniture item because it should not be able to be dropped or picked up.
- When player chose to quit game, QuitGameAction is performed. This set the instance variable, named quit, in GameWorld to true.

Win game

- When player carries out MoveToEarthAction, this action will check if player has YugoMaxx's unconscious body in its inventory. If it has, then set another instance variable, named win in GameWorld to true.

Lose game

- In assignment 2, we have already ensure that the game will not crash when the game stop running. Previously, there is only one reason that ends the game, which is player loses.

End game message:

- The instance variables “quit” and “win” act as sentinels to control whether the run should continue to be run. If either of them is true or player is not in actorLocation, stillRunning() in GameWorld will return false, thus stopping the game from running. The method that return the end game message, endGameMessage() will also be overridden so that appropriate strings are returned depending on the values of quit, win. If both of them are false, this indicates that player is not in actorLocations, so it loses and a lose game message will be returned to be printed out.

Organizing Application

Previously, we set up the game in Application, but it gets harder to maintain and check for bugs as more items and actors have to be added. Therefore, a class named GameManagement is created that contains methods that set up the different parts of the game. Now, in Application, we only have to create a GameManagement instance, then call setUpGame() to get the reference to the game world and run it.