Unit testing [70 marks]

It tests if the methods you implemented are sound. The test includes testing the behavior of the overridden methods, and one of the non-overridden ones.

Input parameters

US 1000 100 0	player name, budget, tourism income, points
NY 2000 800 0	name, population, active, recovered cases
HealthMinister 1 1 1 ChiefExecutiveOfHA 1 1 1 Epidemiologist 1 1 1	Role, leadership, medicine, and experience
100 patients	Constants.MEDICATION_FACILITY_CAPACITY The capacity of medication facility

Class ChiefExecutiveOfHA [12 marks]

Hints: *Role used ChiefExecutiveOfHA*	Description	Tested Methods

Calling overridden method advances the points
 	
Adding then removing Isolation obj.
	• travelBanned → true [1]
	• Contaminants size = 1 [1]
	• Protection level =50 [1]
	• Points = 2 [1]
	`Calling overriden method`
	
	• travelBanned → false [1]
	• Contaminants size = 0 [1]
		`banTravel`
		[6 marks]
		
		`liftTravelBan`

Calling super class methods	• Medication facilities =1 [1]
	• Recovered cases=100 [1]
	• Active cases = 700 [1]
	• Medication level= 12 [1]
	`100 * 100/ 800 = 12 [.5]`
	• Points = 0 [2]
	`Calling super method`
		`developMedicationFacility`
		[6 marks]

Class HealthMinister [26 marks]

Hints: *Role used HealthMinister*	Description	Tested Methods

Calling overridden method	• Budget = 500 [2]
	• Active cases = 700 [1]
	• Medication level= 12 [1]
	`100 * 100/ 800 = 12 [.5]`
	• Points = 3 [2]
	`Calling overriden method`
		developMedicationFacility
		[6 marks]

Can't upgrade if buildMasksfactory wasn't called first
 	
buildMasksfactory, then upgradeMaskQuality must advance protection level to 50
	• Points = 0 [2]
	• Contaminants size=0 [2]
	
	• Protection level= 30 [1]
	• Points = 3 [2]
	`Calling overriden method`
	
	• Protection level= 50 [1]
	`30 + 20`
	• Contaminants size=1 [1]
	• Points = 3 [1]
	`Calling super method`
		upgradeFMaskQuality
		[10 marks]
		
		buildMasksFactory
		
		upgradeFMaskQuality

Calling super class methods yield to 0 points. *Calling develop then upgrade vaccine gives 100% vaccination level*	• Points = 0 [2] • Budget = 950 [1] • Vaccination level= 50 [1] • Points = 0 [2] • Budget = 900 [2] • Vaccination level= 100 [2]	**developVaccine** [10 marks] **upgradeVaccine**

Class Epidemiologist [16 marks]

Hints: *Role used Epidemiologist*	Description	Tested Methods
Can't upgrade if buildMasksfactory wasn't called first	• Points = 2 [1] • Contaminants size=0 [1]	`upgradeFMaskQuality` [6 marks]
	• Protection level= 30 [1] • Points = 2 [1] *Calling super method*	`buildMasksFactory`
	• Protection level= 50 [1] *30 + 20* • Contaminants size=1 [1]	`upgradeFMaskQuality`
Calling superclass methods yield to 0 points.	• Points = 2 [1] • Contaminants size=0 [1]	`upgradeVaccine` [6 marks]
	• Vaccination level= 50 [1]	`developVaccine`
Calling develop then upgrade vaccine gives 100% vaccination level	• Vaccination level= 100 [1] • Points = 6 [2] *2 + 2 + 2*	`upgradeVaccine`
decrement= 970	• Budget = 30 [1] [3] • Throw BudgetRunout exception • getMessage = *"run out of budget 30"*	`decreaseBudget` [4 marks] `developVaccine`

Class City [6 marks]

Hints/Input	Description	Tested Methods
Increment = 1300	• Throw Medical exception [1] • getMessage = *"activeCases cases 2000 reached city's population 2000"* [1]	`increaseActiveCases` [2 marks]

decrement = 2100	• Active cases= 0 [1] • Recovered cases = 2000 [2]	`decreaseActiveCases` [3 marks]
Min med facilities 0	• medicationFacilities=0 [1]	`decreaseMedicationFacility` [1 marks]

Class Player [10 marks]

Hints/Input	Description	Tested Methods
Call beginTurn() of any HAstaff must lead to true	• hasReadHAStaff() → false [1] • hasReadHAStaff() → true [1]	`hasReadyHAStaff` [2 marks]

Call increment 30 then half protection level
 	
Call increment 100, protection level must return a max of 100
	• Size = 1 [1]
	• Protection level =30 [1]
	• Protection level = 15 [1]
	• Protection level = 100 [1]
		`addContainmentTech`
		[4 marks]
		`incrementProtection_level`
		`halfProtection_level`
		`incrementProtection_level`

Call increment 50 then half vaccination level
 	
Call increment 100, vaccination level must return a max of 100
	• Size = 1 [1]
	• Vaccination level =50 [1]
	• Vaccination level = 25 [1]
	• Vaccination level = 100 [1]
		`addContainmentTech`
		`incrementVaccination_level`
		[4 marks]
		`halfVaccination_level`
		`incrementVaccination_level`

Integration testing [20 marks]

The test includes testing a combination of method calls. It verifies and tests the functionalities you implemented from the player's perspective.

Classes **Player** and **HealthAuthorityStaff** [5 marks each test case]

Hints/Input	Description	Tested Methods
Test Case1: *Population 2000 and current activeCases 800. Medical expulsion because no protoction or vaccination → spreadRate =3*	• newInfectedCases = 2400 [1 marks] • Active cases = 2000 [1 marks] • Throw Medical exception [1 marks] • getMessage = *"activeCases cases 2000 reached city's population 2000"* [2 marks]	`computeNewInfectedCases` `|_increaseActiveCases`

Test Case2:
50% protection by using only Isolation
50% vaccination by calling only developVaccine() without upgrade.
 	
increaseFactor =0.5
*SpreadRate=3*0.5*
	• NumNewCases = 600 [2 marks]
	*3*0.5* 0.5* 800*
	• Active cases = 1400 [3 marks]
	600+800
		`banTravel`
		`developVaccine`
		`computeNewInfectedCases`
		`|_increaseActiveCases`

Increments protection level by 30
 	
Fake masks in market:
Half protection level
 	
Call build and upgrade masks multiple times, protection level must be 100 at max
	• Protection level =30
	• Protection level = 15 [2 marks]
	• Protection level = 100 [3 marks]
		`buildMasksFactory`
		
		`halfProtection_level`
		
		`buildMasksFactory`
		`buildMasksFactory`
		`upgradeFMaskQuality`

Increment vaccination level by 50
 	
Half level
 	
Call develop and upgrade multiple times, vaccination level must be 100 at max
	• Vaccination level =50
	• Vaccination level = 25 [2 marks]
	• Vaccination level = 100 [3 marks]
		`developVaccine`
		
		`halfVaccination_level`
		
		`developVaccine`
		`developVaccine`
		`upgradeVaccine`

Execution testing **[10 marks]**

In this test, we verify the overall system functionality. We run each student's project to inspect the output, verify the random events, check losing or winning the game against input trace in file *player.txt*.

The code of Player->**generateUnexpectedDistasters()** and GameEngine->**AnnounceWinner()** will be inspected. They will also be tested by running the code and verifying the output.

generateUnexpectedDistasters [4 marks]
AnnounceWinner [6 marks]

In this test, we run the code multiple times, check if disasters happen and protection/vaccination are halved (by looking at the output), verify if the winner has the minimum active and new cases, and maximum points.