Xi'an Jiaotong-Liverpool University
西交利物浦大学

CPT 302 Individual Project
Coursework/Assignment Submission Form

2021/22 Semester 2
Bachelor's degree – Year 4

Module Code	Module Leader	Module Title
CPT302	**Ka Lok Man**	**Multi-Agent Systems**

Section A: Your Details

To be completed by the student (in English using BLOCK CAPITALS)

Student's Name	Tianlei Shi
Student ID	**1824152**

Section B: Assignment Details

To be completed by the student (in English using BLOCK CAPITALS)

Coursework Assignment Number	**Final Assignment**
Coursework (assignment) Title	**Final Assignment**
Method of Working	**INDIVIDUAL**
Date and time of submission	**30 May 2022, 17:00**

Assignment details can be found in the assignment description.

Section C: Statement of Academic Honesty

To be completed by the student

By submitting this coursework for assessment, you are confirming that you have read and understood the University's policy on plagiarism and collusion and that the submitted work is your own.

(i) I confirm that I have read a copy of the current University's definitions of collusion and plagiarism on coursework and academic honesty, and that I fully understand the meaning of these terms.

(ii) I confirm that the submitted coursework has been created solely by me and that I have not been assisted, nor have copied part or all of somebody else's work, either with their explicit approval or without their knowledge or consent.

(iii) I confirm that this is my own work and that use of material from other sources, including the Internet, has been properly and fully acknowledged and referenced.

(iv) I confirm that the information I have given is correct to the best of my knowledge.

Student's signature	Tianlei Shi	Date	May. 30, 2022

If this form is submitted electronically, please type your name in English (BLOCK CAPTIALS)

Q1,

Sol:

a) I think this statement is NOT true.
A hybrid architecture builds agent out of two subsystems: a deliberative one to develop plans and make decision by symbolic reasoning, and a reactive one to react to events without complex reasoning, and those subsystems are arranged into hierarchy of interacting layers. For InteRRap architecture, it satisfies the requirements above. InteRRap contains three layers: behavior-based layer, local planning layer, and cooperative plaining layer, where behavior-based layer plays a role of the reactive subsystem – it deals with reactive behavior, and local planning layer plays a role of the deliberative subsystem – it deals with planning to achieve the agent's goal. InteRRap uses a vertically layered two-pass architecture, the control may flow to higher layers when the lowest layer receives perceptual input, and flow back to the lowest layer after a decision is made at the higher layer. Moreover, each layer of InteRRap has an associated knowledge base, and which can map raw information to complex models.
Therefore, InteRRap architecture is a hybrid agent architecture.

b) The differences between a Vickrey auction and an English auction are:
   1. They have different auction form
      - In Vickrey auction, bidders submit their bid to auctioneer, and the highest bidder wins, but he/she pays the price of second-highest bid
      - In English auction, bidders raise freely his/her bid publicly, and auction ends if no bidder is willing to raise bid anymore
   2. In English auction, bidders have the incentive to speculation; but in Vickrey auction, there is no incentive to speculation
   3. In Vickrey auction, the dominant strategy of bidders is to submit valuation truthfully; but in English auction, the dominant strategy of bidders is to bid a small amount above highest current bid until the valuation of bidder is reached

c) 1. Possible percepts: Per = {dirt, clean, hole}, which means the environment is dirt/clean/hole.

Domain predicates describing internal state:

In(x, y) – agent is at (x, y)

Near(x, y) – agent can move to (x, y) by only one step

Dirt(x, y) – there is dirt at (x, y)

ObserveDirt(x, y) – there is dirt at (x, y) observed by agent

ObserveHole(x, y) – there is hole at (x, y) observed by agent

IsAvoid(x, y) – agent should not move to (x, y)

AllTravel – all positions have been traveled or observed

Possible actions: Act = { Move(x, y) – agent moves to (x, y)

Suck – suck dirt

ObserveAndExchange – agents observe all adjacent cells, and exchange information they already know, such as the location of explored area or hole

Wait – agent waits in place

Finish(x, y) – agent complete the clean-up task and move to specified location (x, y)

Avoid(x, y) – delete (x, y) from locations that agent possible moving to, i.e., make (x, y) become IsAvoid(x, y)}

Deduction rules:

In(x, y) $\wedge$ Dirt(x, y) -> Do(Suck)

In(x, y) $\wedge$ $\neg$ Dirt(x, y) -> Do(ObserveAndExchange)

In(x, y) $\wedge$ $\neg$ Dirt(x, y) $\wedge$ $\neg$ AllTravel -> Do(Wait)

Near(x, y) $\wedge$ ObserveHole(x, y) -> Do(Avoid(x, y))

In(x, y) $\wedge$ $\neg$ Dirt(x, y) $\wedge$ ObserveDirt($\neg$ x, $\neg$ y) $\wedge$ $\neg$ IsAvoid($\neg$ x, $\neg$ y) $\wedge$ $\neg$ AllTravel -> Do(Move($\neg$ x, $\neg$ y))

In(x, y) $\wedge$ $\neg$ Dirt(x, y) $\wedge$ $\neg$ IsAvoid($\neg$ x, $\neg$ y) $\wedge$ $\neg$ AllTravel -> Do(Move($\neg$ x, $\neg$ y))

In(x, y) $\wedge$ $\neg$ Dirt(x, y) $\wedge$ AllTravel -> Do(Finish($\neg$ x, $\neg$ y))

In the Vacuum World, the robot R could move to adjacent four cells (up, down, left, and right). It can also observe adjacent cells to detect whether there is dirt or hole (ObserveHole, and ObserveDirt), and after observation, the agents will exchange the information they already know, including the location of explored area, the location of hole, and the location of cleaned area. Moreover, each robot maintains a database, which record cells that have traveled or observed (AllTravel is get from this), and cells that should be avoided (IsAvoid), and they will update the database according to the exchanged information as well.

2. Clean task start

Robot agent A	Robot agent B
In(1, c) ∧ ¬ Dirt(1, c) -> Do(ObserveAndExchange)	In(2, a) ∧ ¬ Dirt(2, a) -> Do(ObserveAndExchange)
In(1, c) ∧ ¬ Dirt(1, c) ∧ ¬ IsAvoid(1, d) ∧ ¬ AllTravel -> Do(Move(1, d))	In(2, a) ∧ ¬ Dirt(2, a) ∧ ObserveDirt(2, b) ∧ ¬ IsAvoid(2, b) ∧ ¬ AllTravel -> Do(Move(2, b))
In(1, d) ∧ ¬ Dirt(1, d) -> Do(ObserveAndExchange)	In(2, b) ∧ Dirt(2, b) -> Do(Suck)
Near(1, e) ∧ ObserveHole(1, e) -> Do(Avoid(1, e))	In(2, b) ∧ ¬ Dirt(2, b) -> Do(ObserveAndExchange)
In(1, d) ∧ ¬ Dirt(1, d) ∧ ¬ IsAvoid(2, d) ∧ ¬ AllTravel -> Do(Move(2, d))	In(2, b) ∧ ¬ Dirt(2, b) ∧ ¬ IsAvoid(2, c) ∧ ¬ AllTravel -> Do(Move(2, c))
In(2, d) ∧ ¬ Dirt(2, d) -> Do(ObserveAndExchange)	In(2, c) ∧ ¬ Dirt(2, c) -> Do(ObserveAndExchange)
In(2, d) ∧ ¬ Dirt(2, d) ∧ ObserveDirt(2, e) ∧ ¬ IsAvoid(2, e) ∧ ¬ AllTravel -> Do(Move(2, e))	In(2, c) ∧ ¬ Dirt(2, c) ∧ ObserveDirt(3, c) ∧ ¬ IsAvoid(3, c) ∧ ¬ AllTravel -> Do(Move(3, c))
In(2, e) ∧ Dirt(2, e) -> Do(Suck)	In(3, c) ∧ Dirt(3, c) -> Do(Suck)
In(2, e) ∧ ¬ Dirt(2, e) -> Do(ObserveAndExchange)	In(3, c) ∧ ¬ Dirt(3, c) -> Do(ObserveAndExchange)
In(2, e) ∧ ¬ Dirt(2, e) ∧ ¬ IsAvoid(3, e) ∧ ¬ AllTravel -> Do(Move(3, e))	In(3, c) ∧ ¬ Dirt(3, c) ∧ ¬ IsAvoid(3, c) ∧ ¬ AllTravel -> Do(Move(4, c))
In(3, e) ∧ ¬ Dirt(3, e) -> Do(ObserveAndExchange)	In(4, c) ∧ ¬ Dirt(4, c) -> Do(ObserveAndExchange)
In(3, e) ∧ ¬ Dirt(3, e) ∧ ¬ IsAvoid(4, e) ∧ ¬ AllTravel -> Do(Move(4, e))	Near(4, b) ∧ ObserveHole(4, b) -> Do(Avoid(4, b))

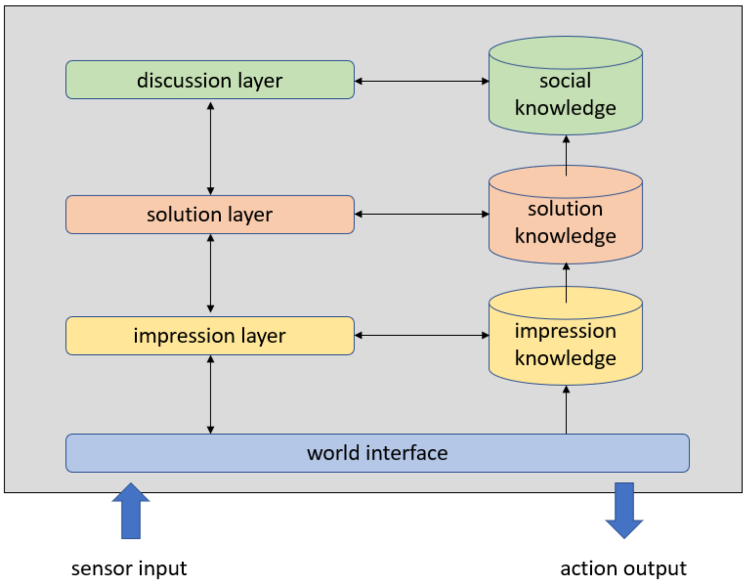
In(4, e) ∧ ¬ Dirt(4, e) -> Do(ObserveAndExchange)	In(4, c) ∧ ¬ Dirt(4, c) ∧ ¬ IsAvoid(5, c) ∧ ¬ AllTravel -> Do(Move(5, c))
Near(5, e) ∧ ObserveHole(5, e) -> Do(Avoid(5, e))	In(5, c) ∧ ¬ Dirt(5, c) -> Do(ObserveAndExchange)
In(4, e) ∧ ¬ Dirt(4, e) ∧ ¬ IsAvoid(3, e) ∧ ¬ AllTravel -> Do(Move(3, e))	In(5, c) ∧ ¬ Dirt(5, c) ∧ ¬ IsAvoid(5, b) ∧ ¬ AllTravel -> Do(Move(5, b))
In(3, e) ∧ ¬ Dirt(3, e) ∧ ¬ IsAvoid(2, e) ∧ ¬ AllTravel -> Do(Move(2, e))	In(5, b) ∧ ¬ Dirt(5, b) -> Do(ObserveAndExchange)
In(2, e) ∧ ¬ Dirt(2, e) ∧ ¬ AllTravel -> Do(Wait)	In(5, b) ∧ ¬ Dirt(5, b) ∧ ¬ IsAvoid(5, a) ∧ ¬ AllTravel -> Do(Move(5, a))
In(2, e) ∧ ¬ Dirt(2, e) ∧ ¬ AllTravel -> Do(Wait)	In(5, a) ∧ ¬ Dirt(5, a) -> Do(ObserveAndExchange)
In(2, e) ∧ ¬ Dirt(2, e) ∧ ¬ AllTravel -> Do(Wait)	In(5, a) ∧ ¬ Dirt(5, a) ∧ ObserveDirt(4, a) ∧ ¬ IsAvoid(4, a) ∧ ¬ AllTravel -> Do(Move(4, a))
In(2, e) ∧ ¬ Dirt(2, e) ∧ ¬ AllTravel -> Do(Wait)	In(4, a) ∧ Dirt(4, a) -> Do(Suck)
In(2, e) ∧ ¬ Dirt(2, e) ∧ AllTravel -> Do(Finish(2, e))	In(4, a) ∧ ¬ Dirt(4, a) ∧ AllTravel -> Do(Finish(4, a))

Clean task finish

Q2,

Sol:

a) What I designed is a question answering agent architecture, and its architecture is shown in below:



This agent architecture is a vertically layered two-pass architecture, and consists of three layers: impression layer, solution layer, and discussion layer. Impression layer deals with reactive behavior, if question is simple or it has impression, it will answer the question. Solution layer deals with complex questions by reasoning to achieve the goal of agent. Discussion layer deals with social interactions, it will discuss with other agents to solve questions. Moreover, each layer has a relevant knowledge base, which can provide necessary information. Control may flow from lower layers to higher layers if the lower layer cannot deal with the current situation, and then control flow back again after high layer make a decision. For example, the agent needs to solve a question, now control in the impression layer. If the question is simple or agent has solved it before (has an impression), then impression layer will solve it and output result, otherwise, control will flow to solution layer or even discussion layer. After higher layer solved the question by reasoning or discussing with other agents, the control will flow back to impression layer to output the solution.

b) Practical reasoning is reasoning directed towards actions, and it figures out what agent should do. Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes. Moreover, practical reasoning consists of two components: deliberation, and means-ends reasoning, and deliberation decide what state of affairs we want to achieve, and means-ends reasoning decide how to achieve these states of affairs. For instance, an agent now feels weary, so it has an initial intention – {take a rest}. It also has some beliefs – {sleep -> take a rest, and go to bedroom ∩ go to bed -> sleep}. Thus, the agent can generate desires according to intention and beliefs, the desires may be – {sleep, go to bedroom, go to bed}. Finally, the agent will get the intention – {take a rest, sleep, go to bedroom, go to bed}, and it will go to bedroom, and go to bed, then sleep to take a rest.

c) 1. The "Blind Commitment" protocol is used in this code.

2. Delete the 13 – 21 line, and the code will become "Overcommitted".

3. Change the 9th line to: "while not (empty(π) or succeeded(I, B) or believeimpossible(I, B)) do", and the code will become use "Open-minded commitment".

4. Assume the commitment protocol "Single-minded commitment" is used in the above code, the agent will stop to reconsider its intentions in the 15th line.

d) 1. A STRIPS Planning Strategy is a planner. Planner is a system that takes as input the following: i) a goal, intention or task, ii) the current state of the environment, iii) the actions available to the agent, and generates a plan as output. STRIPS consists of two basic components: a model of the world, and a set of action schemata describing preconditions and effects of all actions. What's more, STRIPS achieves the goal based on finding the difference between the current state of the world and the goal state, and then reducing this difference by applying an appropriate action.

2. Predicates for describing the robot's world:

Predicate	Meaning
In(x, y)	object x in area y
UnderLight(x, y)	object x is under light y

LightOn(x)	light x is on
LightOff(x)	light x is off
ArmEmpty(x)	robot x is not push any object
On(x, y)	object x is on top of object y
Clear(x)	there is nothing on top of object x
SamePlace(x, y)	object x and object y in the same place
CanTouch(x, y)	robot x can touch object y
Hold(x, y)	robot x is pushing object y

Stack Operations:

Operation	Specification	Meaning

Move(x, y, z)	pre {In(x, ¬ y), ArmEmpty(x)}
	del {In(x, ¬ y)}
	add {In(x, y), SamePlace(x, ¬ x)}
		robot x moves to area y through door z

Push(x, y, z, k)	pre {SamePlace(x, y), ArmEmpty(x)}
	del {ArmEmpty(x)}
	add {In(x, z), In(y, z), Hold(x, y), SamePlace(x, ¬ x)}
		robot x push object y to area z through door k

PushLight(x, y, z)	pre {SamePlace(x, z), SamePlace(x, y)}
	del {}
	add {UnderLight(y, z), CanTouch(x, z), Hold(x, y)}
		robot x push object y under light z

ClimbOn(x, y)	pre {SamePlace(x, y), Clear(y), Hold(x, y)}
	del {Clear(y)}
	add {On(x, y)}
		robot x climbs on object y

ClimbDown(x, y)	pre {On(x, y)}
	del {On(x, y)}
	add {Clear(y)}
		robot x climbs down object y

TurnOff(x, y)	pre {CanTouch(x, y), LightOn(y)}
	del {LightOn(y)}
	add {LightOff(x)}
		robot x turns off light y

PutDown(x, y)	pre {Hold(x, y)}
	del {Hold(x, y)}
	add {ArmEmpty(x)}
		robot x put down object y

The initial state of the robot's world:

{In(Robot_1, Corridor_1), In(Robot_2, Room_2), In(Box_1, Room_3), In(Box_2, Corridor_2), In(Box_3, Room_1), In(Box_4, Room_4), In(Box_5, Room_5), In(Box_6, Room_5), ArmEmpty(Robot_1), ArmEmpty(Robot_2), Clear(Box_1), Clear(Box_2), Clear(Box_3), Clear(Box_4), Clear(Box_5), Clear(Box_6), LightOn(switch_1), LightOn(switch_2), LightOn(switch_3), LightOn(switch_4), LightOn(switch_5), LightOn(switch_6)}

The goal to achieve:

{In(Box_1, Room_1), In(Box_2, Room_2), In(Box_3, Room_3), In(Box_4, Room_4), In(Box_5, Room_5), In(Box_6, Room_6), LightOff(switch_1), LightOff(switch_2), LightOff(switch_3), LightOff(switch_4), LightOff(switch_5), LightOff(switch_6)}

The plan to achieve the goal:

For robot_1: Move(robot_1, Room_5, door_5) -> PushLight(robot_1, Box_6, switch_5) -> ClimbOn(robot_1, Box_6) -> TurnOff(robot_1, switch_5) -> ClimbDown(robot_1, Box_6) -> Push(robot_1, Box_6, Corridor_1, door_5) -> Push(robot_1, Box_6, Room_6, door_6) -> PushLight(robot_1, Box_6, switch_6) -> ClimbOn(robot_1, Box_6) -> TurnOff(robot_1, switch_6) -> ClimbDown(robot_1, Box_6) -> PutDown(robot_1, Box_6) -> Move(robot_1, Corridor_1, door_6) -> Move(robot_1, Room_1, door_1) -> PushLight(robot_1, Box_3, switch_1) -> ClimbOn(robot_1, Box_3) -> TurnOff(robot_1, switch_1) -> ClimbDown(robot_1, Box_3) -> Push(robot_1, Box_3, Corridor_1, door_1) -> Push(robot_1, Box_3, Room_3, door_3) -> PushLight(robot_1, Box_3, switch_3) -> ClimbOn(robot_1, Box_3) -> TurnOff(robot_1, switch_3) -> ClimbDown(robot_1, Box_3) -> PutDown(robot_1, Box_3) -> Push(robot_1, Box_1, Corridor_1, door_3) -> Push(robot_1, Box_1, Room_1, door_1) -> PutDown(robot_1, Box_1)

For robot_2: Move(robot_2, Corridor_2, door_2) -> Push(robot_2, Box_2, Room_2, door_2) -> PushLight(robot_2, Box_2, switch_2) -> ClimbOn(robot_2, Box_2) -> TurnOff(robot_2, switch_2) -> ClimbDown(robot_2, Box_2) -> PutDown(robot_2, Box_2) -> Move(robot_2, Corridor_2, door_2) -> Move(robot_2, Room_4, door_4) -> PushLight(robot_2, Box_4, switch_4) -> ClimbOn(robot_2, Box_4) -> TurnOff(robot_2, switch_4) -> ClimbDown(robot_2, Box_4) -> PutDown(robot_2, Box_4)

Q3,

Sol:

a) I believe that "Blocks World" is NOT an example of a "subsumption architecture". Because the subsumption architecture based the following theses: i) Intelligent behavior can be generated without explicit representations, ii) Intelligent behavior can be generated without explicit abstract reasoning, iii) Intelligence is an emergent property of certain complex systems. However, the Blocks World uses set of formulas of first-order logic to explicitly represent the model of world, and it also need the means-ends reasoning to generate behavior explicitly. Therefore, "Blocks World" is NOT an example of a "subsumption architecture.

b) Individual behaviors:

b_0: if detect a hole then avoid the cell

b_1: if detect other agents then avoid the cell

b_2: if detect a dirt then move to the cell and suck the dirt

b_3: if not clean up at least one dirt then move randomly

b_4: if not clean up all dirt then move randomly

b_5: if clean up all dirt then move to specific cell

The above behaviors are arranged into the hierarchy:

$$b_0 \prec b_1 \prec b_2 \prec b_3 \prec b_4 \prec b_5$$

c) In a game, if there is no other outcome that makes one agent better off without making another agent worse off, and we say the outcome is Pareto efficient.

To show that this game has allocations can be Pareto efficient, we assume that for Mary, if she gets one apple, she will get 1 point, if she gets one orange, she gets -1 point. And for Peter, if he gets one orange, he will get 1 point, otherwise -1 point. The payoff matrix may like below:

	Mary: eat apples	Mary: eat oranges
Peter: eat apples	(-3.5, 3.5)	(-7, -7)
Peter: eat oranges	(7, 7)	(3.5, -3.5)

In the payoff matrix, the pair (7, 7) is Pareto efficient. Therefore, there is an allocation can be Pareto efficient – Peter eats all oranges, and Mary eat all apples.

d) 1. The payoff matrix is shown in below:

	David: play SD	David: play ST
Tina: play SD	(a, b)	(x, c)
Tina: play ST	(d, x)	(e, f)

2. The given property is hold in the given payoff matrix. Because this is no conflict, and we can take the Prisoner's Dilemma as an example. If strategy SD is confess, and strategy ST is not confess, the payoff matrix will satisfy the above payoff matrix, below is an example.

	Play 2 confesses	Player 2 does not confess
Player 1 confesses	(5, 5)	(**0**, 10)
Player 2 does not confess	(10, **0**)	(1, 1)

3. Assume that the payoff matrix is like below:

	David: play SD	David: play ST
Tina: play SD	(3, 3)	(5, 1)
Tina: play ST	(1, 5)	(1, 1)

In the above payoff matrix, the pair (3, 3) is Nash equilibrium, and this pair is also a Pareto efficient.
Because if agent i plays strategy s1, agent j can do no better than play strategy s2, and if agent j plays strategy s2, agent i can do no better than play strategy s1, then the outcome is Nash equilibrium. In the above payoff matrix, if Tina plays SD, Davide has two choices: play SD to get 3, and play ST to get 1, so Davide has no reason to play ST. And if David plays SD, Tina also has two choices: play SD to get 3, or play ST to get 1, Tina must choice SD as well. Therefore, (3, 3) is Nash equilibrium.
Moreover, (3, 3) is also Pareto efficient. Because if there is no other outcome that makes one agent better off without making another agent worse off, and we say the outcome is Pareto efficient. Pairs (1, 5) and (5, 1) all make one agent better and

other agent worse, and pair (1, 1) makes all agents worse. Therefore, the pair (3, 3) is Nash equilibrium, and this pair is also a Pareto efficient.

4. The above example is NOT unique. For example, the payoff matrix below shows another example:

	David: play SD	David: play ST
Tina: play SD	(2, 1)	(0, 0)
Tina: play ST	(0, 0)	(1, 2)

In the above example, we have two pairs (2, 1) and (1, 2), and they are both Nash equilibrium and Pareto efficient. Therefore, the example is NOT unique.

5. The case shown below has two Nash equilibria:

	David: play SD	David: play ST
Tina: play SD	(4, 5)	(2, 3)
Tina: play ST	(1, 2)	(5, 4)

There are two Nash equilibria in the above case, they are pair (4, 5) and pair (5, 4).

Q4,

Sol:

a) 1. The core of a coalitional game is the set of feasible distributions of payoff to members of a coalition that no sub-coalition can reasonably object to.

   2. The question of "is the grand coalition stable" imply another question "is the core non-empty". Because if the core is non-empty, which means there is nobody can benefit from defection, so sub-coalition has no reason to defection, and the grand coalition is stable.

   3. If the core is empty, which means that the coalition will not distribute payoff to its sub-coalition, and this coalition will bring no benefit to sub-coalition, and those sub-coalitions may get benefit from defection. Thus, the coalition is not stable and has a danger of breaking up.

b) μ ({a, b, c}) = 2 + 1 + 2 = 5,
   μ ({a, b, d}) = 2 + 1 + 3 = 6,
   μ ({a, c, d}) = 2 + 1 = 3,
   μ ({b, c, d}) = 1 + 3 = 4,
   μ ({a, b, c, d}) = 1 + 2 + 3 + 2 + 1 = 9.

c) The characteristic function v is defined by the following:

   - $v(\emptyset) = 0$
   - $v(\{i\}) = 0, where\ i\ = \{a, b, c\}$
   - $v(\{a, b\}) = \frac{1}{2}$
   - $v(\{a, c\}) = \frac{1}{6}$
   - $v(\{b, c\}) = \frac{5}{6}$
   - $v(\{a, b, c\}) = 1$

   According to the characteristic function:

Probability	Order	a	b	c
$\frac{1}{6}$	a -> b -> c	$v(\{a\}) = 0$	$v(\{a,b\}) - v(\{a\}) = \frac{1}{2} - 0 = \frac{1}{2}$	$v(\{a,b,c\}) - v(\{a,b\}) = 1 - \frac{1}{2} = \frac{1}{2}$
$\frac{1}{6}$	a -> c -> b	$v(\{a\}) = 0$	$v(\{a,b,c\}) - v(\{a,c\}) = 1 - \frac{1}{6} = \frac{5}{6}$	$v(\{a,c\}) - v(\{a\}) = \frac{1}{6} - 0 = \frac{1}{6}$
$\frac{1}{6}$	b -> a -> c	$v(\{a,b\}) - v(\{b\}) = \frac{1}{2} - 0 = \frac{1}{2}$	$v(\{b\}) = 0$	$v(\{a,b,c\}) - v(\{a,b\}) = 1 - \frac{1}{2} = \frac{1}{2}$
$\frac{1}{6}$	b -> c -> a	$v(\{a,b,c\}) - v(\{b,c\}) = 1 - \frac{5}{6} = \frac{1}{6}$	$v(\{b\}) = 0$	$v(\{b,c\}) - v(\{b\}) = \frac{5}{6} - 0 = \frac{5}{6}$
$\frac{1}{6}$	c -> a -> b	$v(\{a,c\}) - v(\{c\}) = \frac{1}{6} - 0 = \frac{1}{6}$	$v(\{a,b,c\}) - v(\{a,c\}) = 1 - \frac{1}{6} = \frac{5}{6}$	$v(\{c\}) = 0$
$\frac{1}{6}$	c -> b -> a	$v(\{a,b,c\}) - v(\{b,c\}) = 1 - \frac{5}{6} = \frac{1}{6}$	$v(\{b,c\}) - v(\{c\}) = \frac{5}{6} - 0 = \frac{5}{6}$	$v(\{c\}) = 0$

So, we can get the Shapley value for each agent:

$\mathrm{Sh}\ (S, a)/\varphi_a = \frac{1}{6}\ (0 + 0 + \frac{1}{2} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6})\ = \frac{1}{6}$,

$\mathrm{Sh}\ (S, b)/\varphi_b = \frac{1}{6}\ (\frac{1}{2} + \frac{5}{6} + 0 + 0 + \frac{5}{6} + \frac{5}{6})\ = \frac{1}{2}$,

$\mathrm{Sh}\ (S, c)/\varphi_c = \frac{1}{6}\ (\frac{1}{2} + \frac{1}{6} + \frac{1}{2} + \frac{5}{6} + 0 + 0)\ = \frac{1}{3}$.

Therefore, the Shapley value for agent a, b, and c are $\frac{1}{6}$, $\frac{1}{2}$, and $\frac{1}{3}$ respectively.

Q5,

Sol:

a) I believe that Plurality Voting is not a suitable method to select among four candidates.
There is a counter example: in Plurality Voting, it is possible that two candidates with the most votes have the same number of votes, and this will result in the failure to find the final winner. Therefore, Plurality Voting is not a suitable method to select among four candidates.

b) 1. If Alice will win the election using a plurality vote, the preference should like below:

A: 20%	B: 35%	C: 45%
Lucy	Jim	Alice
Alice	Alice	Lucy
Jim	Lucy	Jim

2. The meant by "sequential majority elections with Lucy, Jim and Alice" is that three candidates decide the winner by using sequential majority elections, and vote will be done in several steps, and in each step, candidates face each other, and the winner progresses to the next election.
According to the preference orders above:
For Lucy and Jim: votes of Lucy beats Jim = 20% + 45% = 65%; votes of Jim beats Lucy = 35%; so, Lucy wins.
For Lucy and Alice: votes of Lucy beats Alice = 20%; votes of Alice beats Lucy = 35% + 45% = 80%; so, Alice wins.
For Jim and Alice: votes of Jim beats Alice = 35%; votes of Alice beats Jim = 20% + 45% = 65%; so, Alice wins.
Therefore, Alice is the winner.

3. If Jim will be the final winner of the election using a Borda count, the preference orders should like below:

W: 10%	X: 20%	Y: 30%	Z: 40%
Alice	Bob	Lucy	Jim
Bob	Alice	Jim	Lucy
Lucy	Jim	Bob	Alice
Jim	Lucy	Alice	Bob

For Lucy, the votes are: 2×10% + 1×20% + 4×30% + 3×40% = 280%;

For Jim, the votes are: 1×10% + 2×20% + 3×30% + 4×40% = 300%;

For Alice, the votes are: 4×10% + 3×20% + 1×30% + 2×40% = 210%;

For Bob, the votes are: 3×10% + 4×20% + 2×30% + 1×40% = 210%;

Therefore, Jim is the final winner of above election using a Borda count.

4. According to the preference orders above:

For Lucy and Jim: votes of Lucy beats Jim = 10% + 30% = 40%; votes of Jim beats Lucy = 20% + 40% = 60%; so, Jim wins.

For Lucy and Alice: votes of Lucy beats Alice = 30% + 40% = 70%; votes of Alice beats Lucy = 10% + 20% = 30%; so, Lucy wins.

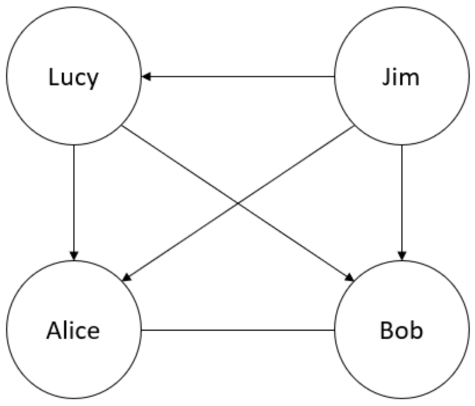
For Lucy and Bob: votes of Lucy beats Bob = 30% + 40% = 70%; votes of Bob beats Lucy = 10% + 20% = 30%; so, Lucy wins.

For Jim and Alice: votes of Jim beats Alice = 30% + 40% = 70%; votes of Alice beats Jim = 10% + 20% = 30%; so, Jim wins.

For Jim and Bob: votes of Jim beats Bob = 30% + 40% = 70%; votes of Bob beats Jim = 10% + 20% = 30%; so, Jim wins.

For Alice and Bob: votes of Alice beats Bob = 10% + 40% = 50%; votes of Bob beats Alice = 20% + 30% = 50%; so, Alice and Bob tie.

Thus, the Majority graph is shown in below:



Therefore, Jim is the Condorcet winner.

c) Cooperative Distributed Problem Solving (CDPS) focus on how to solve a problem by modular problems and assign tasks, and control component agents to let them work together by efficient protocol, and finally construct a loose coupling network.

For example, agents now face to a problem, they are required to design and build an airplane. This task requires a lot of expertise (such as kinematics, dynamics, and strength of materials) and many resources that are beyond the capabilities of a single agent. Thus, cooperation is necessary because no single agent has sufficient expertise resources and information to solve the problem, and different agents might have expertise for solving different parts of the problem.
Therefore, the manager agent will decompose problem, and then broadcast announcements to invite bids, and announcements include description of task, constraints, and meta-task information. Next, agents that receive the announcement decide for themselves whether they wish to bid for the task, and if they choose to bid, then they submit a tender, and manager agent screens them to determine who to "award the contract" to. Finally, the successful contractor then expedites the task (become component agent), or conduct further sub-contracting (become manager agent). At the end, agents will form manager agents and contractor agents. Manager agents are responsible for management and control, and contractor agents are responsible for using their abilities and resources to solve problems. For example, the task of building airplane may be divided into sub-tasks such as material, engine, model, and the agent who is good at the sub-task will be responsible for the task and communicate the progress with its manager.

The main problems that need to be addressed in CDPS include the following:

1. How can a problem be divided into smaller tasks for distribution among agents?
2. How can a problem solution be effectively synthesized from subproblem results?
3. How can the overall problem-solving activities of the agents be optimized so as to produce a solution that maximizes the coherence metric?
4. What techniques can be used to coordinate the activity of the agents, thus avoiding destructive (and therefore unhelpful) interactions, and maximizing effectiveness (by exploiting any positive interactions)?