1. The layout for a 2-player board game is shown in Fig 2.1

START	1	2	3	4	5	6	7
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
31	30	29	28	27	26	25	24
32	33	34	35	36	37	38	39
47	46	45	44	43	42	41	40
48	49	50	51	52	53	54	55
END	62	61	60	59	58	57	56

Fig 2.1

The game is played by rolling two 6-sided dice and moving that number of spaces. Both players start on the START space. If a player lands on a space occupied by the other player, they move to the next available space.

The board is to be stored as a 2-dimensional array.

Each time a player moves, a series of obstacles are to be added to the board.

On their turn, each player rolls two dice. The smaller number from the two dice is taken, and that many obstacles will appear on the board in random locations.

For example, if a 3 and 6 are rolled, then 3 obstacles will appear.

A recursive function is written in pseudocode to perform this task.

© OCR 2019.

PhysicsAndMathsTutor.com

```
01 function generateObstacle(diceNumber)
02     if diceNumber == 0 then
03         return true
04     else
05         x = randomNumber(0, 7)
06         y = randomNumber(0, 7)
07         board(x, y) = new obstacle()
08         generateObstacle(diceNumber-1)
09     endif
10 endfunction
```

The code `new obstacle()` generates an instance of the object obstacle.

(i) Explain the purpose of the code in line `01` in the algorithm.

---

[2]

(ii) Identify the line of code where recursion occurs.

---

[1]

(iii) The recursive function could have been written using iteration.

Describe the benefits and drawbacks of using recursion instead of iteration.

Benefits ---

Drawbacks ---

---

[4]

(iv) Rewrite the function so it uses iteration instead of recursion.

---

[4]

(v) If a position on the board is not occupied, its value is set to a blank string ("").

The current algorithm does not check if the random space generated is currently occupied.

Write a subroutine that takes the generated position of the board, checks if it is free and returns `true` if free, or `false` if occupied.

[3]

2. Many functions can be defined using either recursion or iteration.

(i) State **one** advantage of using recursion instead of iteration.

(ii) State **one** disadvantage of using recursion instead of iteration.

---

[2]

3. Consider the following algorithm in Fig.2, expressed in pseudocode, as a function S:

```
function S(A[0..N-1], value, low, high)

     if (high < low)then
           return error_message
      endif

      mid = (low + high) / 2

     if (A[mid] > value)then
            return S(A, value, low, mid-1)
     elseif (A[mid] < value)then
           return S(A, value, mid+1, high)
     else
           return mid
    endif

endfunction
```

Fig.2

(i) Describe what is meant by recursion.

---

[2]

(ii) Identify **one** example of where recursion occurs in this algorithm.

---

[1]

4(a). A recursive function, `calculate`, is shown below:

```
01 function calculate(num1, num2)
02    if num1 == num2 then
03        return num1
04    elseif num1 < num2 then
05        return calculate(num1, (num2-num1))
06    else
07        return calculate(num2, (num1-num2))
08    endif
09 endfunction
```

Identify the lines where recursion is used.

[1]

(b). Re-write the function so it uses iteration instead of recursion.

[4]

(c). Trace the algorithm, showing the steps and result when the following line is run:

```
print(calculate(4,10))
```

[5]

5. A developer is extending a game that allows users to have multiple pets of different types.
The developer has written a class, `Pet`.

The attributes and methods in the class are described in the table:

Identifier	Attribute / Method	Description
`petName`	Attribute	Stores the pet's name
`bored`	Attribute	Stores the % bored
`hunger`	Attribute	Stores the % hunger
`intelligence`	Attribute	Stores the intelligence
`type`	Attribute	Stores the type of animal
`new`	Method	Creates a new instance of `pet`
`feed`	Method	Reduces `hunger` to 0 and outputs `hunger`
`play`	Method	Reduces `bored` to 0 and outputs `bored`
`read`	Method	Increases `intelligence` by a set value
`outputGreeting`	Method	Outputs a message to the user

Part of the class declaration is given:

```
class Pet
    private petName
    private bored
    private hunger
    private intelligence
    private type
 ...
 ...
```

(i) After a user enters the pet name, and chooses a type, the constructor method of Pet is called to create a new instance. The method needs to set `petName`, as well as `hunger`, `bored` and `intelligence` to starting values of 0.

Write, using pseudocode, the constructor method for this class.

---

[4]

(ii) Write a line of code that creates a new instance of `Pet` for a Tiger called “Springy”.

---

[2]

(iii) The method `outputGreeting` for the superclass is written as follows:

```
public procedure outputGreeting()
    print("Hello, I'm " + petName + ", I'm a " + type)
endprocedure
```

A class is needed for Tiger. The class needs to:

- inherit the methods and attributes from `pet`
- in the constructor, set `type` to Tiger, `intelligence` to 10, `hunger` to 50 and `bored` to 10
- extend the method `outputGreeting`, by outputting an additional line that says “I like to eat meat and roar”

© OCR 2019.

PhysicsAndMathsTutor.com

Write, using pseudocode, the class `Tiger`.

[5]

6(a). Kim is writing an object-oriented program for a four player board game. The board has 26 squares that players move around, as shown in Fig. 5.1.

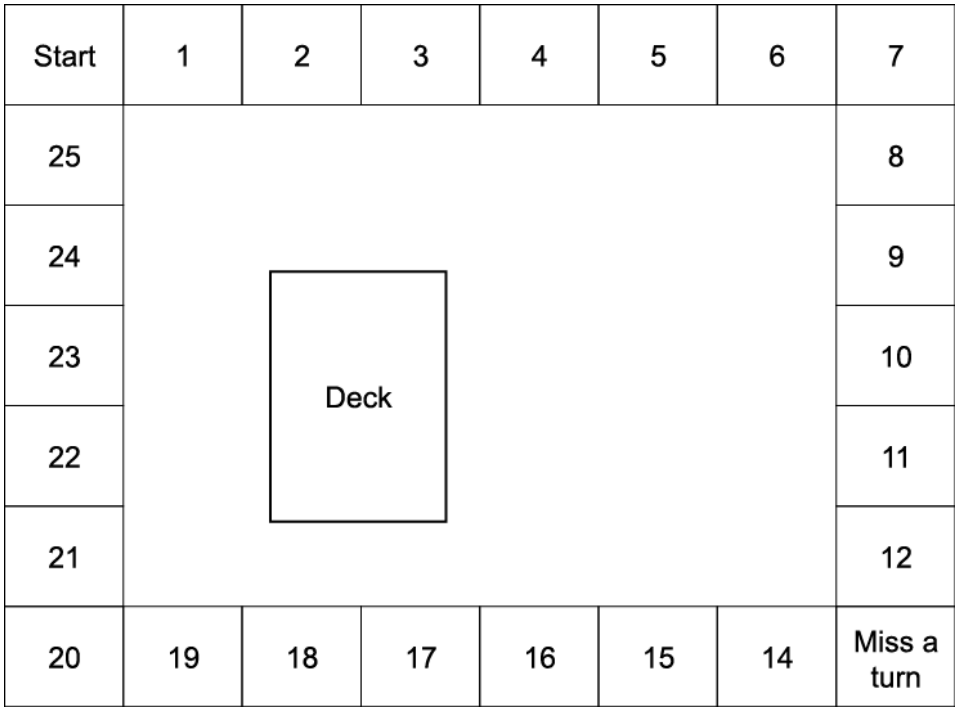

Fig. 5.1

Each player takes it in turn to roll two dice. They then move that number of spaces on the board. If they roll a double (both dice have the same value), they then take a card from the deck. The deck contains 40 cards that each include a sentence (such as “You have won the lottery”). The sentence on the card determines if money is given or taken away from the player.

© OCR 2019.

PhysicsAndMathsTutor.com

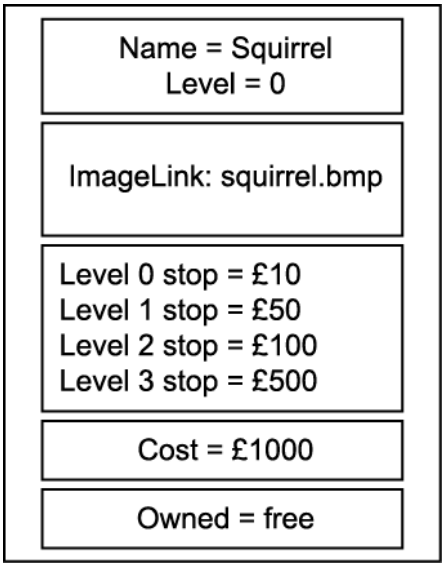

Fig. 5.2

Each square (apart from Start and Miss a turn) has an animal associated with it that the player can purchase, if it has not been purchased already, for example square 6 has a Squirrel. Fig. 5.2 shows an example of one of these animals. Once a player has purchased the animal, any opposing player which subsequently lands on the square/animal has to pay a fine.

Each animal can be upgraded, with each upgrade the game charges more each time a player stops on them. For example, with no upgrade the level 0 squirrel costs £10 when a player stops on it. If £1000 is paid to upgrade, the squirrel is then a level 1 animal and now charges £50 for a stop.

The cost to purchase and upgrade the animal is the same.

Each animal can be upgraded to a maximum of level 3.

When a player lands on, or passes the square ‘Start’ (position 0), they receive £500. If they land on ‘Miss a turn’ (position 13), they miss their next turn.

(i) A class, Player, stores the player's ID (P1, P2, P3, P4), their current board position and the amount of money they have.

Fig. 5.3 shows a class diagram for Player. A class diagram describes a class. It contains the class name, followed by the attributes, then the methods.

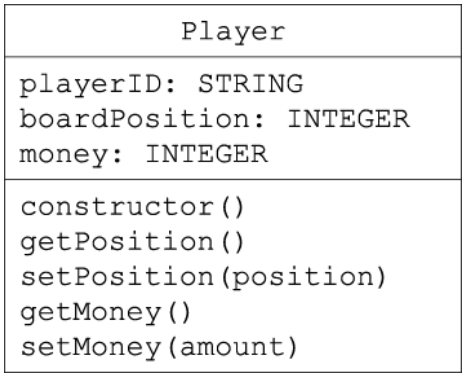

Fig. 5.3

The constructor creates a new instance of Player, taking the player's ID as a parameter. The board position is set to 0, and money to £2000.

Write, using pseudocode, the constructor method for the Player class.

--- [3]

(ii) A class, Animal, define the attributes and methods for the animals stored in each square.
Fig. 5.4 shows a class diagram for Animal.

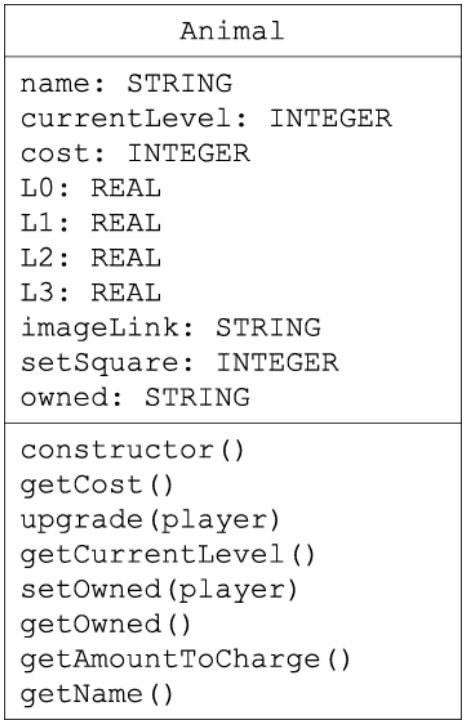

Fig. 5.4

The constructor takes the required data as parameters and then sets currentLevel to 0, and assigns the parameters as the remaining attributes for the new object.

Write, using pseudocode, the constructor method for the Animal class.

[4]

(iii) Write, using pseudocode, the code to create an instance of Animal for the Squirrel shown in Fig. 5.2, positioned on square number 6, for the constructor function you wrote in part **(a)(ii)**.

---

[2]

(b). The board is stored as a 1D array, board, of data type Animal. The spaces at 0, and 13, are left as empty elements that are checked using separate functions.

(i) Complete, using pseudocode, the function to:

- Roll both dice
- Move the player, the dice number of spaces
- If a double is rolled, calls the procedure pickDeck
- Adds £500 if they have passed or landed on Start
- Calls the procedure missAGo if they land on space 13 or
- Calls the procedure checkAnimal
- Return the new position

```
function playerMove(currentPlayer)
    dice1 = random(1,6)
    dice2 = random(1,6)
    boardPosition = ........................... + dice1 + dice2
    if ..............  == dice2 then
        pickDeck(currentPlayer)
    endif
    if position > 25 then
        currentPlayer.setMoney(currentPlayer.getMoney() +  .............)
        position = position -  ..............................
    endif
    if position == ................................ then
        missAGo(currentPlayer)
    elseif position != 0 then
        checkAnimal(currentPlayer)
    endif
    ..............................................................
endfunction
```

[6]

(ii) *The parameter currentPlayer from part **(b)(i)** can be passed by value or by reference.

Explain the difference, benefits and drawbacks between passing by value and by reference. Recommend which should be used for currentPlayer, justifying your decision.

---

[9]

(c). The deck is stored as a zero-indexed 1D array, named deck, of type Card.

The class diagram for Card is shown in Fig. 5.5.

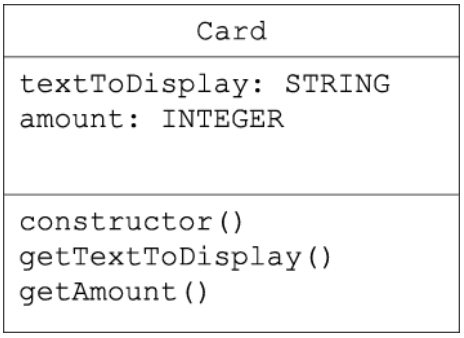

Fig. 5.5

The array, deck, is treated as a queue, with a variable, headPointer, identifying the first card in the deck. When a card has been used, the head pointer increases to move to the next position. If the end of the deck is reached, the head pointer returns to 0 and starts again.

The procedure pickDeck:

- takes the current player as a parameter
- outputs the text to be displayed from the first card in the queue
- adds or subtracts the amount to/from the current player's money
- increases the head pointer

Write, using pseudocode, the procedure pickDeck.

---

[6]

(d). The procedure checkAnimal:

- Takes the current player as a parameter
- Accesses the data for the animal at the player's position in the array board
- If the animal is free, asks the player if they would like to purchase the animal and outputs its name and cost, if they choose to buy the animal, it calls the procedure purchase() with the player and animal as parameters
- If that player owns the animal, and it is not at level 3, it asks if they would like to upgrade the animal
- If they would like to upgrade, it calls the method upgrade for that animal with the current player as a parameter
- If a different player owns the animal, it calls the method getAmountToCharge() for that animal, sending this value and the current player as parameters to the procedure chargeStay()

Write, using pseudocode, the procedure checkAnimal.

[10]

END OF QUESTION PAPER

Question			Answer/Indicative content	Marks	Guidance
1		i	1 mark per bullet to max 2 • Declares a function called `generateobstacle` (1) • Has parameter `diceNumber` (1)	2	
		ii	• 08 (1)	1	
		iii	max 3 marks for benefit, max 3 for drawback, max 4 marks overall Benefit • More natural to read (1) • Quicker to write / less lines of code. (1) As some functions are naturally recursive (1) • Suited to certain problems (1) For example those using trees (1) • Can reduce the size of a problem with each call. (1) Drawback • Can run out of stack space / memory (1) (due to too many calls (1)) causing it to crash (1) This can be avoided with tail recursion (1) • More difficult to trace / follow (1) as each frame on the stack has its own set of variables (1) • Requires more memory than the equivalent iterative algorithm. • Usually slower than iterative methods (1) due to maintainence of the stack (1)	4	

Question			Answer/Indicative content	Marks	Guidance

		iv	1 mark per bullet
			
			• Loop start and end in correct positions (1)
			• With correct number of iterations (1)
			• Returns a value (1)
			• All other code correct, in the right place (1)
			
			e.g.
			`function generateobstacle(diceNumber)`
			`  for count = 0 to diceNumber`
			`  x = randomNumber(0, 7)`
			`  y = randomNumber(0, 7)`
			`  board(x, y) = new obstacle()`
			`  next count`
			`  return true`
			`endfunction`
				4	

		v	1 mark per bullet, to max 3
			
			• Appropriate declaration of function, taking 2 parameters (1)
			• Checks position in board against “” correctly (1)
			• Returns `false` and true correctly (1)
			
			e.g.
			`function checkFree(x, y)`
			`  if board(x, y) == "" then`
			`  return true`
			`  else`
			`  return false`
			`  endif`
			`endfunction`
				3	

			Total	**14**	

Question			Answer/Indicative content	Marks	Guidance
2		i	*eg* • Code is generally shorter • (can be) closer to natural language description	1	Allow humans think recursively **Examiner's Comments** Many poor answers were given here. Many students didn't appear to have a depth of knowledge regarding recursion and its issues beyond “it calls itself”.
		ii	*eg* • Uses more memory / resources • Difficult to trace / debug	1	Difficult to understand is TV Allow difficult to follow **Examiner's Comments** Many poor answers were given here. Many students didn't appear to have a depth of knowledge regarding recursion and its issues beyond “it calls itself”.
			Total	**2**	
3		i	• The function calls itself (1) from within the function.	2	Up to 2 marks for a valid description.
		ii	• Return S(A, value, low, mid-1) (1) return S(A, value, mid+1, high) (1).	1	For 1 mark (either point). Accept if point in the algorithm is unambiguously referenced. There are two recursive calls in the program. Either is acceptable.
			Total	**3**	

Question			Answer/Indicative content	Marks	Guidance
4	a		05 and 07	1 AO2.1 (1)	

	b		1 mark per bullet to max 4
			
			• Suitable loop with correct condition
			
			• In IF: Overwriting num2 with num2 – num1
			
			• In ELSE: Overwriting num1 with num2…
			
			• … Overwriting num2 with num1–num2 correctly (using a temp variable)
			
			e.g.
			`while num1 != num2`
			`  if num1 < num2 then`
			`  num2 = num2 - num1`
			`  else`
			`  temp = num1 - num2`
			`  num1 = num2`
			`  num2 = temp`
			`  endif`
			`endwhile`
				4
				AO2.1
				(1)
				AO2.2
				(1)
				AO3.2
				(2)
					Alternatively swapping values by
					`temp = num1`
					`num1 = num2`
					`num2 = temp - num2`
					
					Examiner's Comment:
					Most candidates produced recognisable pseudocode. Weaker candidates produced logically incorrect solutions or did not understand the difference between an iterative and a recursive solution – reformulating another recursive solution. Where strong candidates produced good solutions they sometimes forget the necessity to have a temporary swap variable when swapping the values in two different variables over.

Question			Answer/Indicative content	Marks	Guidance
	c		1 mark for each highlighted element  	5 AO2.1 (5)	Allow trace table or any sensible equivalent. **Examiner's Comment:** Many candidates scored well in parts (a) and (b) and it was pleasing to see that recursion could both be identified and traced. Few candidates achieved full marks in part (b) because they did not appreciate that the function was inside a print statement so a final output of 2 would be produced after the value 2 was returned.
			Total	10	

Question			Answer/Indicative content	Marks	Guidance

5		i	1 mark per bullet to max 4
			
			• Correct declaration, appropriate name (e.g. `new`)
			
			• Taking `name` and `theType` as a parameter
			
			• Setting `petName` to parameter
			
			• Setting `bored, hunger` and `intelligence` to 0
			
			e.g.
			`public procedure new(name, theType)`
			`  petName = name`
			`  bored = 0`
			`  hunger = 0`
			`  intelligence = 0`
			`  type = theType`
			`endprocedure`
				4
				AO2.2
				(1)
				AO3.2
				(3)	

		ii	1 mark per bullet to max 2 • `myPet`/appropriate = `new pet` • `Springy` and `Tiger,` in `" "`, in same order as constructor declaration e.g. `myPet = new pet("Springy", "Tiger")`	2 AO2.1 (2)	

Question			Answer/Indicative content	Marks	Guidance

		iii	1 mark per bullet to max
			
			• Class declaration including inherit (or equivalent e.g. Tiger extends Pet, Tiger::Pet, Tiger(Pet))
			
			• Constructor procedure (new) with all attributes present
			
			  • bored = 10, hunger = 50, intelligence = 10, type = "Tiger"
			
			• outputGreeting procedure
			
			  • Outputting original and new messages correctly
			
			e.g.
			`class Tiger inherits Pet`
			`public procedure new(name)`
			`petName = name`
			`bored = 10`
			`hunger = 50`
			`intelligence = 10`
			`type = "Tiger"`
			`endprocedure`
			
			`public procedure outputGreeting()`
			`print("Hello, I'm " + petName + ", I'm a " + type)`
			`print("I like to eat meat and roar")`
			`endprocedure`
			`endclass`
				5
				AO2.2
				(2)
				AO3.2
				(3)
					Accept
					super.outputGreeting()
					In place of first print statement
					
					Examiner's Comment:
					Many candidates struggled with the application of object oriented techniques and concepts and it was clear that many of these candidates had not had practical experience of object oriented programming. Stronger candidates did perform well and understood how to create instances from classes and how to use inheritance.

			Total	11	

Question			Answer/Indicative content	Marks	Guidance

6	a	i	1 mark per bullet to max 3
			
			• Declaring the procedure and taking a player ID as parameter
			• Setting playerID to parameter
			• Setting boardPosition to 0 and money to 2000
			
			e.g.
			`public procedure new(thePlayerID)`
			`  playerID = thePlayerID`
			`  boardPosition = 0`
			`  money = 2000`
			`endprocedure`
				3	

		ii	1 mark per bullet to max 4
			
			• Declaration as public and procedure, named constructor/new and Taking all correct parameters (missing currentLevel)
			• Sets currentLevel to 0
			• Setting all the data…
			• …to the matching parameters taken
			
			e.g.
			`public procedure new(theName, theCost, theL0, theL1, theL2, theL3, theImageLink, theSetSquare, theOwned)`
			`  name = theName`
			`  currentLevel = 0`
			`  cost = theCost`
			`  L0 = theL0`
			`  L1 = theL1`
			`  L2 = theL2`
			`  L3 = theL3`
			`  imageLink = theImageLink`
			`  setSquare = theSetSquare`
			`  owned = theOwned`
			`endprocedure`
				4	

		iii	1 mark per bullet to max 2 • Creating new instance e.g. squirrel = new Animal • Parameters matching part (b)(i) e.g. `squirrel = new Animal("Squirrel", 1000, 10, 50, 100, 500, "squirrel.bmp", 6, "free")`	2	

Question			Answer/Indicative content	Marks	Guidance

	b	i	1 mark for each correctly completed space
			
			`function playerMove(currentPlayer)`
			`  dice1 = random(1,6)`
			`  dice2 = random(1,6)`
			`  position = currentPlayer.getPosition() + dice1 + dice2`
			`  if dice1 == dice2 then`
			`  pickDeck(currentPlayer)`
			`  endif`
			
			`  if position > 25 then`
			`  currentPlayer.setMoney(currentPlayer.getMoney() + 500)`
			`  position = position - 26`
			`  endif`
			`  if position == 13 then`
			`  missAGo(currentPlayer)`
			`  elseif position != 0 then`
			`  checkAnimal(currentPlayer)`
			`  endif`
			`  return position`
			`endfunction`
				6	

Question			Answer/Indicative content	Marks	Guidance
		ii	**Mark Band 3 – High level (7-9 marks)** The candidate demonstrates a thorough knowledge and understanding of passing values by reference and by value; the material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. The candidate provides a thorough discussion which is well balanced. Evaluative comments are consistently relevant and well considered *There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.* **Mark Band 2 – Mid level (4-6 marks)** The candidate demonstrates reasonable knoledge and understanding of passing values by reference and by value; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate provides a reasonable discussion, the majority of which is focused. Evaluative comments are, for the most part appropriate, although one or two opportunities for development are missed. *There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence* **Mark Band 1 – Low Level (1-3 marks)** The candidate demonstrates a basic knowledge of passing values by reference and by value with limited understanding shown;	9	**AO1: Knowledge and Understanding** **Indicative content** By Value • sends the actual value • if changes are made then only the local copy is amended By Reference • sends a pointer to the value • The actual value is not sent/received • If changed the original is also changed when the subroutine ends **AO2: Application** • Send by value • The currentPlayer value is not /does not need to be changed in the subprogram • Send by reference • The currentPlayer value is updated **AO3: Evaluation** • ByValue creates new memory space… • ByReference means existing memory space is used • Depends if original variable is local/global • If local and just referenced, send by value • If original value needs editing send by reference • If passing by reference then instead of returning position the code could just amend currentPlayer.position • If passing by value there could be inconsistencies when currentPlayer is passed to other methods, for example pickDeck

Question			Answer/Indicative content	Marks	Guidance
			the material is basic and contains some inaccuracies. The candidates makes a limited attempt to apply acquired knowledge and understanding to the context provided. The candidate provides a limited discussion which is narrow in focus. Judgements if made are weak and unsubstantiated. *The information is basic and comunicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be clear.* **0 marks** No attempt to answer the question or response is not worthy of credit.		

	c		1 mark per bullet to max 6
			
			• Procedure declaration with correct name and parameter
			• Outputting the correct text from deck at headPointer
			• Sending to currentPlayer.setMoney …
			• getMoney + deck at head pointer amount
			• Increase the head pointer
			• Set headPointer to 0 if position 40 or greater
			
			`procedure pickDeck(currentPlayer)`
			`  output(deck[headPointer].getTextToDisplay())`
			`  amount = deck[headPointer].getMoney()`
			`  currentPlayer.setMoney(currentPlayer.getMoney() + amount)`
			`  headPointer = headPointer + 1`
			`  if headPointer == 40 then`
			`  headPointer = 0`
			`  endif`
			`endprocedure`
				6	

Question			Answer/Indicative content	Marks	Guidance

	d		• Declaring the procedure with correct parameters
			• Check if the space/animal is free
			  ◦ If free, outputting name and cost
			  ◦ Checking if they want to buy
			  ▪ Calling purchase with current player and animal
			• If they own the animal, checking if they can upgrade
			  ▪ If they can, asking if they want to upgrade outputting the cost
			  ▪ If they want to, calling the upgrade method
			• If they don't own the animal
			  ◦ Calling chargeStay with the amount to charge and the current player
			
			e.g.
			`procedure checkAnimal(currentPlayer)`
			
			`  if board[currentPlayer.getPosition()].owned == "free"`
			`  answer = input("Would you like to purchase ",`
			`  board[position].getName(), "? It costs ",`
			`  board[position].getCost()`
			
			`  if answer = "yes" then`
			`  purchase(currentPlayer, board[position])`
			`  endif`
			
			`  elseif board[currentPlayer.getPosition()].getOwned() == currentPlayer`
			
			`  if board[currentPlayer.getPosition()].getCurrentLevel() != "L3"`
			`  answer = input("Upgrade? It costs ", board[position].getCost())`
			
			`  if answer == "yes" then`
			`  board[currentPlayer.getPosition()].upgrade(currentPlayer)`
			`  endif`
			
			`  endif`
			
			`  else`
			`  amount = board[position].getAmountToCharge()`
			`  chargeStay(amount, currentPlayer)`
			`  endif`
			`endprocedure`
				10	Allow follow through for incorrect accessing of methods

			Total	**40**	