Centre Number						Candidate Number				
Surname										
Other Names										
Candidate Signature										

General Certificate of Secondary Education
For submission in 2017

Computer Science 4512 4512/CB3

Unit 4512/1 – Practical Programming

Scenario 3 Game Application

AQA Treasure Hunt

For candidates entering for the 2017 examination
To be issued to candidates on or after Friday 15 May 2015

*This scenario is one of four available. Each of the four scenarios is available in a separate Candidate Booklet. You must complete **two** of the four scenarios.*

- You have approximately 25 hours in which to complete this scenario.
- Before starting work on the problem, read the whole of this Candidate Booklet thoroughly. You can ask your teacher to explain anything in this booklet, except Computer Science specific terms, that you do not understand.
- There are restrictions on when and where you can work on this problem. Your teacher will explain them to you. For example, you should only do work that you intend to hand in for marking when a teacher is present, so that he or she can confirm that the work is your own. The Candidate Booklet must **not** be taken outside your school/college.
- You may need to use the Internet to research certain parts of the problem. This does not have to be within the 25 hours recommended time.
- You will need to complete and sign a Candidate Record Form which your teacher will provide.

Information

You will also be marked on your use of English. It is important to:

- make sure that all your work is legible
- use correct spelling, punctuation and grammar
- use a style of writing which suits the person you are writing for
- organise your information clearly, so that you make yourself understood
- use Computer Science terms where they are needed.

Scenario 3: AQA Treasure Hunt

AQA Treasure Hunt is a game where a player moves through an eight by eight grid of squares searching for hidden treasure. In the grid are ten treasure chests and five bandits.

The player starts in the bottom left corner of the grid. The player then moves around the grid trying to find the treasure chests and avoid to the bandits. For each move, the player chooses how many squares they want to move up or down and how many squares they want to move left or right.

If the player lands on a square that contains a treasure chest then they collect 10 gold coins.

If the player lands on a square that contains a bandit then the bandit steals all the coins that the player has collected so far.

The player cannot move beyond the limits of the grid.

Each treasure chest can be visited three times. After the treasure chest has been visited three times, it is replaced by a bandit in the same square.

After each move, the grid showing the new position of the player is displayed, along with the current number of gold coins collected, the number of bandits and treasure chests in the grid.
The player wins the game if they have collected 100 gold coins.

The player loses the game if all the treasure chests have been changed into bandits and the player has not collected 100 gold coins.

Tasks

1. Develop the part of the program that displays a menu that allows the player to play the game or to quit the program.

2. Develop the part of the program in such a way that when the play game option is selected from the menu, it creates positions for ten treasure chests and five bandits within an eight by eight grid. These positions should be random squares within the grid.

   Each treasure chest and each bandit must be in its own square. A treasure chest or bandit cannot be placed in the start position for the player.

3. Develop the part of the program that displays the grid and start position of the player. The positions of the treasure chests and bandits should not be shown to the player.
   NOTE: For testing purposes the positions of the treasure chests and bandits **can** be shown, but in the finished game it is expected that they will not be seen.

4. Develop the part of the program that enables the player to enter their move. The player should be able to enter the number of squares they want to move up or down and how many squares they want to move left or right. The player should not be allowed to make a move that would place them outside the grid. **Figure 1** shows some examples of legal moves. After a legal move has been made, one should be added to the total number of moves made by the player.

5. Develop the part of the program that checks into which square the player has moved.

   a) If the player moves into a square containing a treasure chest then 10 coins should be added to the player's coin collection.
   b) If the player moves into a square containing a bandit then the player's coin collection should be set to zero.
   c) If the player moves into a square containing a treasure chest that has already been visited twice then 10 coins are added to the player's coin collection and the treasure chest is replaced by a bandit. The next time the square is visited, it contains a bandit. The number of bandits is increased by one and the number of treasure chests is decreased by one.

6. Develop the part of the program that displays the grid showing the new position of the player, the number of gold coins collected and the number of bandits and treasure chests currently in the grid.

7. Develop the part of the program that checks whether the player has won or lost the game.

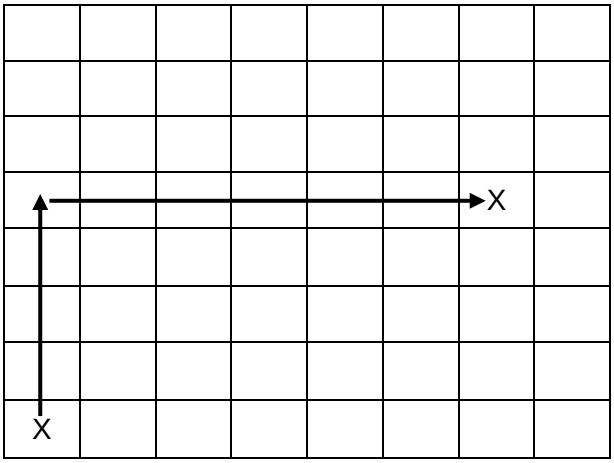
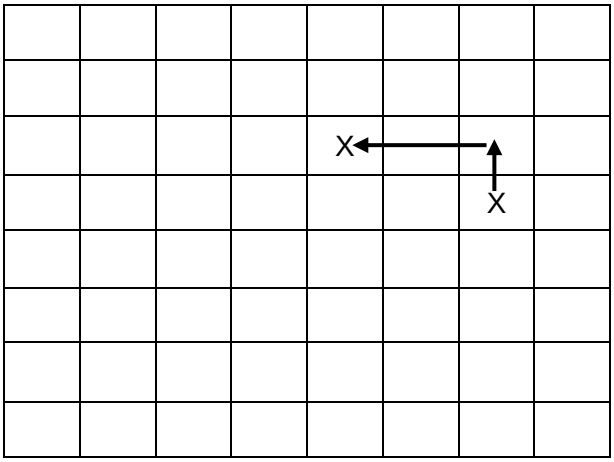
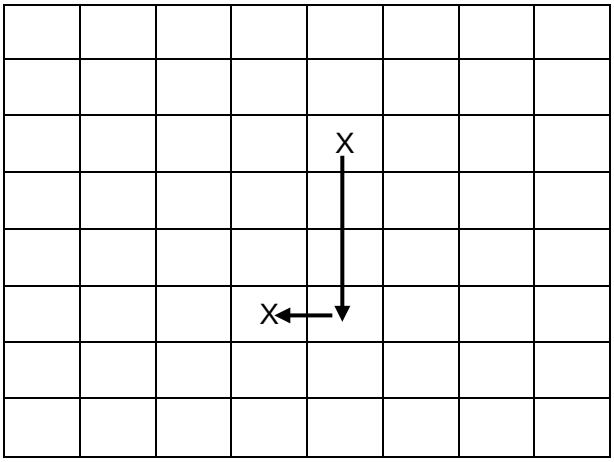
   a) The player has won the game if they collect 100 gold coins.
   b) The player has lost the game if there are no treasure chests left in the grid and the number of gold coins collected is less than 100.

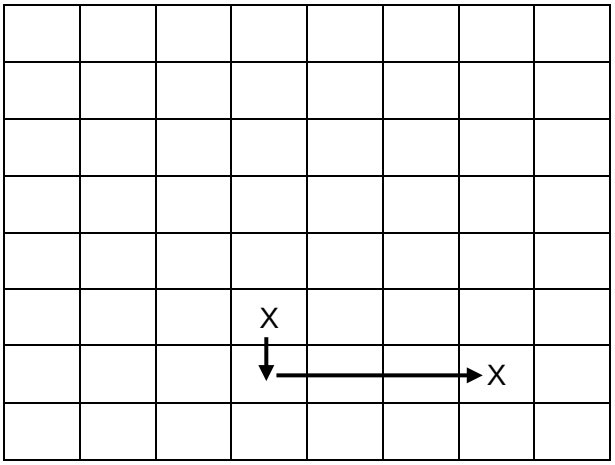
   When the player finishes the game, an appropriate message should be displayed telling them whether they have won or lost and the number of moves made in the game. The player should then be taken back to the menu developed in **Task 1**.

8. Extend the program so that the player can select the size of the grid and the starting number of treasure chests and bandits before playing the game.

   a) The player selects from a list of game layouts described in the form of a grid: for example, 10 x 10 or 12 x 12.
   b) The player enters the number of treasure chests and bandits.

Figure 1

Vertical	Horizontal	Example	Grid
UP	RIGHT	UP 4 RIGHT 6	
UP	LEFT	UP 1 LEFT 2	
DOWN	LEFT	DOWN 3 LEFT 1	

Vertical	Horizontal	Example	Grid
DOWN	RIGHT	DOWN 1 RIGHT 3	

Turn over for information on organising your portfolio

In addition

1. Your Portfolio

You are free to use whatever programming tools and techniques are available to you.

What your teacher will be looking for and how to provide that evidence for your Portfolio

In preparing you for this unit of work, your teacher will have provided you with more information about the section headings below.

Part 1 – Design of solution

Design of solution (0–9 marks available)
What you must do
• Show an understanding of what the problem involves with reference to the user's needs. • Produce an overview plan that shows how the problem is to be solved. • Produce pseudo code (or suitable alternative) showing the main blocks within the proposed solution.

Part 2 – Solution development

Solution development (0–9 marks available)
What you must do
• Show evidence of an understanding of how the final solution meets the needs of the user. • Produce annotated code that demonstrates an understanding of the programming techniques used.

Part 3 – Programming techniques used

Programming techniques used (0–36 marks)
What you must do
• Show an understanding of the programming techniques used and how the different parts of the solution work together. • Explain/justify the choice of programming techniques used to create a solution that has been coded efficiently. • Show evidence for the purpose and use of data structures. • Show the techniques used (appropriate to the language used) within the code to make the solution robust.

Part 4 – Testing and evaluation

Testing and evaluation (0–9 marks available)
What you must do
• Produce a test plan that shows the expected tests, test data and expected results. • Show that the planned tests have been carried out and provide a record of the actions taken. • Evaluate how the final solution meets the needs of the user.

2. Organising your Portfolio of work

Your Portfolio is where you keep the evidence that you have produced.

You should imagine that the Portfolio is to be used by another person who is interested in how you produced your solution. It is to help them to do something similar. It is important that you organise work for the Portfolio as shown below.

- You must keep all the work you produce in hard copy in a Portfolio (or save your work electronically which you will later copy onto a CD or DVD). Your teacher will have instructed you on what to do.

- If you are putting hard copy printouts in your Portfolio, make sure that you number each page and fasten it all together. Take your work out of any plastic sleeves before you hand it in to your teacher for marking.

- Each page should have your name, centre number and candidate number clearly shown on it.
- **Your** work must comprise (in the order shown):

    1. a *Design of solution* section
    2. a *Solution development* section
    3. a *Programming techniques used* section
    4. a *Testing and evaluation* section

It is vital for assessment and moderation purposes that the **sections are the same as those shown above** and in the same order.

When you have completed this scenario, assemble the work into a single document.

END OF CANDIDATE BOOKLET