

# PAPER 1 EXAM RESOURCE PACK 2024

for A Level AQA Computer Science

**JAVA EDITION**

## - DIGITAL RESOURCE -

This pack includes paper versions of the electronic files.

Go to [zzed.uk/ProductSupport](https://zzed.uk/ProductSupport) to download the electronic files.



**POD  
12194**

[zigzageducation.co.uk](https://zigzageducation.co.uk)

Publish your own work... Write to a brief...  
Register at [publishmenow.co.uk](https://publishmenow.co.uk)

Follow us on X (Twitter) @ZigZagComputing

# Contents

<b>Product Support from ZigZag Education .....</b>	<b>ii</b>
<b>Terms and Conditions of Use.....</b>	<b>iii</b>
<b>Teacher's Introduction .....</b>	<b>iv</b>

## **Printouts of electronic resources (for reference)**

- Code Breakdown (7 pages)
- Puzzle File Breakdown (3 pages)
- Advanced Techniques (3 pages)
- UML Class Diagram – Complete (1 page)\*
- UML Class Diagram – Activity (1 page)\*
- Theory Questions: Write-on Version (10 pages)
- Theory Questions: Non-write-on Version (5 pages)
- Coding Tasks (21 pages)
- Additional Tasks (Extension) (3 pages)
- Theory Questions: Mark Scheme (4 pages)
- Programming Tasks: Mark Scheme (58 pages)
- Electronic Answer Document (3 pages)

*\* Note there are also electronic copies of the UML Diagrams ('Complete' & 'Activity' versions) which can be printed in A3, making them much more usable (especially when used as activities)*

## Teacher's Introduction

**Symbol Puzzle** is a single-player puzzle game where the user needs to place symbols into a grid, building up patterns. Each time the user places a new symbol into the grid, the application compares cells in the grid, looking for pattern matches. The objective of the program is to get the highest score possible by adding valid patterns into the grid.

The application can either be played using an 8 × 8 standard puzzle grid, or load in a 5 × 5 external file with a partially complete puzzle.

The user can place Q, T or X symbols into empty cells in the grid, attempting to make larger patterns of these letters. Each valid matched pattern awards the user 10 points. The application has a fixed number of symbols available – either 64 for the standard puzzle or varying amounts, dependent on which external puzzle file the user loads. A user can place each symbol as a Q, a T or an X into the grid, subject to there being space and subject to the placement rules.

Patterns are matched in a 3 × 3 section of the grid – a pattern is nine cells in total in a specific order. Once a valid match for a specific symbol has been identified, the cells in that 3 × 3 section are modified so that the same symbol cannot be placed into any of the cells in that 3 × 3 section of the grid in a future move. This prevents overlapping patterns of the same symbol type being placed into the grid. Symbols of a different type can be placed into those cells, however, allowing for patterns of a different symbol type to overlap. The grid can also contain blocked cells – denoted by an '@' symbol. The user cannot place a symbol into these cells.

When the user has exhausted all their symbols, the puzzle is complete.

This resource aims to help you get to grips with and prepare for the A Level Paper 1 examination for summer 2024, which is partly based on the **Symbol Puzzle** pre-release material.

### DIGITAL RESOURCE

Once you have downloaded the files for this resource via ([zzed.uk/ProductSupport](https://zzed.uk/ProductSupport)) you will have access to the following:



SymbolPuzzle	this folder contains all of the content (PDF/DOCX) accessible via a HTML interface
Passwords.txt	for teacher use – this file contains all of the passwords for the protected PDFs (also listed below)

\* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

**Installation:** Extract the files from the downloaded ZIP file and move the entire SymbolPuzzle folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

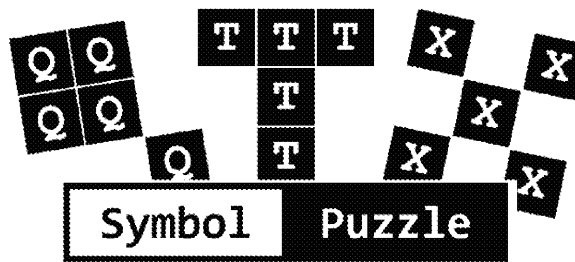
**Passwords:** All of the PDFs accessible via the *Solutions* web page are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

- j02a-UML-Diagram-Complete.pdf
- j06-TheoryQuestions-MS.pdf
- j07-CodingTasks-MS.pdf

The resource pack consists of the following sections:

- **Code breakdown:** a detailed technical overview of the skeleton program, describing in detail each class and method in turn – including their purpose/function, parameters and return values. Note that this is intended as a helpful reference document only, and not as a substitute for exploring the code in a practical manner. In addition to the code breakdown, there is also a breakdown of the puzzle files, errors in the code, and a set of 'advanced techniques' for exploring the code further.
- **UML class diagram activity:** requires you to study the program and fill in the gaps with the missing class/method names, data types, associations and access levels. There are 10 in total.
- **Video:** a quick overview of the **Symbol Puzzle** game mechanics – intended as a visual aid to accompany the notes in the official AQA pre-release material.
- **Theory questions:** designed to test your understanding of the skeleton program. These questions require access to the program, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on versions are available.
- **Coding tasks:** there are 18 modification tasks to test your programming skills – as well as an additional 12 modification ideas that you may also want to try as extension tasks.

This resource is intended to supplement your teaching only. Please read full disclaimer (p. iii) before using it.



## Skeleton Code Breakdown

### Static Methods

Identifier / Data		Description
Main		
Parameters	default	<p>This is the main entrance point to determine whether the application will run a standard, randomly created puzzle or load an external puzzle text file.</p> <p>It initialises a string variable of value 'y' and an integer variable.</p> <p>The method then enters into a loop held in that loop by the value of 'y' and operates using the following steps:</p> <ul style="list-style-type: none"> <li>Prompt the user if they want to create a new puzzle or load an external puzzle.</li> <li>Instantiate a new puzzle object.</li> <li>If the user has entered a file name, call the constructor in the puzzle class as a parameter. This will load the puzzle from an external text file.</li> <li>If the user does not enter a file name, the gridsize is calculated by the length of the file name. The gridsize for a standard puzzle is 8 columns wide by 8 rows high. The gridsize is calculated by squaring the file name length by 0.6 and then rounding down.</li> <li>The method then calls the myPuzzle which starts the puzzle when a puzzle is complete. It also sets the score variable which is the number of moves made.</li> <li>The method then prompts the user if they want to do another puzzle, setting the value of 'y' appropriately to either start another puzzle, or allow it to end the program.</li> </ul>
Return values	n/a	

INSPECTION COPY

COPYRIGHT  
PROTECTED




## Class: Puzzle

Identifier / Data		Description
<<constructor>>		
Parameters	filename : String	Initialises the following private attributes: <ul style="list-style-type: none"><li>• grid as List of Cells</li><li>• allowedPatterns as List of Patterns</li><li>• allowedSymbols as List of Symbols</li></ul> <p>These attributes are subsequently used in the loadPuzzle() method.</p> <p>The method then calls the loadPuzzle() method with the filename parameter.</p> <p><b>This constructor is called when the puzzle is loaded.</b></p>
Return values	n/a	
<<constructor>>		
Parameters	size : Int startSymbols : Int	Initialises the following private attributes: <ul style="list-style-type: none"><li>• score to 0</li><li>• symbolsLeft from parameter startSymbols</li><li>• gridsize from parameter size</li><li>• grid as List of Cells</li></ul> <p>The method performs a count of the square of the gridsize. It then creates and adds all the cells to the grid. Using a random number generator, each cell has a 90% chance of being a valid cell and a 10% chance of being a blocked cell.</p> <p>The method then initialises all the allowed patterns and allowed symbols. It then generates the default patterns and symbols, adding them to the allowedPatterns and allowedSymbols lists.</p> <p><b>This constructor is called when the puzzle is generated.</b></p>
Return values	n/a	

INSPECTION COPY

COPYRIGHT  
PROTECTED



attemptPuzzle (public)		
Parameters	n/a	This method is the main loop of time. It is held in the loop until finished.
Return values	score : Int	
		<p>The method firstly displays the calling the displayPuzzle() method to show the current user score.</p> <p>The method then asks the user for the row and column of where they want to place the symbol in the puzzle. The method uses try-catch to ensure if the user has entered integers. If not, it gives an error message and asks for valid input within the bounds of the puzzle. It does not give any error messages if the input is valid.</p> <p>The method then calls the getSymbol() method to get a symbol to place into the puzzle. It also decrements the number of symbols available in the puzzle.</p> <p>The method then creates a local variable for the row and column given by the user and these can be used in the checkSymbolAllowed() method. The symbol entered by the user as a parameter is used in that cell, the symbol in the cell is changed using the changeSymbolInCell() method.</p> <p>The method then checks if the symbol matches by calling the checkForMatch() method, passing row and column entered by the user as parameters. The result of this is an integer variable amountToAward. If it is greater than 0, it is added to the user's score.</p> <p>The method then checks if all symbols have been used, and if so, exits the loop.</p> <p>If the main program has exited, the user score is displayed by calling the displayScore() method and the user score is returned.</p>

INSPECTION COPY

COPYRIGHT  
PROTECTED





checkForMatchWithPattern (public)		
Parameters	row : Int column : Int	<p>Uses a nested loop to iterate through the grid, building together a string variable called <code>patternString</code> from nine cells in a <math>3 \times 3</math> section of the grid. A nested loop is used to iterate through all combinations of pattern that can be found at different locations in a <math>3 \times 3</math> section of the grid, with a try...catch structure to protect against a cell location outside of the board.</p> <p>Once the <code>patternString</code> has been built, the method uses a foreach loop to check if the <code>patternString</code> is in the <code>allowedPatterns</code> list by calling the <code>contains()</code> method, passing the <code>patternString</code> as a parameter. If checked as parameters.</p> <p>If a match is found, the method calls <code>addToNotAllowedSymbols()</code> with the matching <math>3 \times 3</math> section of the grid as a parameter, preventing the symbol from being placed into the grid.</p> <p>If a match has been found, the method returns <code>true</code>, otherwise it returns the <code>symbol</code>.</p>
Return values	Int	
createHorizontalLine (private)		
Parameters	n/a	<p>Uses iteration to concatenate characters which is the correct line in the current puzzle.</p>
Return values	symbol : String	
displayPuzzle (public)		
Parameters	n/a	<p>Used to print out the grid onto the console. It works by using the following steps:</p> <ul style="list-style-type: none"><li>• Print a blank line.</li><li>• If the <code>gridSize</code> is less than the screen, then iterate through the grid, printing a space followed by the character in the current column.</li><li>• Print a blank line.</li><li>• Print a horizontal line by calling <code>createHorizontalLine()</code> with the <code>gridSize</code> as a parameter.</li><li>• The method then iterates through the grid, printing out the row number, the column number, and the symbol in the current cell. Iteration uses the MOD function to determine if a row before printing a horizontal line underneath.</li><li>• This process is repeated until the entire grid is printed to the screen.</li></ul>
Return values	n/a	
getCell (private)		
Parameters	row : Int column : Int	<p>Uses the row and column parameters to return the correct Cell element in the one-dimensional array. It is then returned.</p>
Return values	Cell	



getRandomInt (private)		
Parameters	min : Int max : Int	Returns a random integer value in the range.
Return values	Int	This method is only included in the final release.
getSymbolFromUser (private)		
Parameters	n/a	Used for getting a symbol from the user. The method uses a loop to keep asking the user to enter the symbol they want to use until they enter a valid symbol which is part of the allowedSymbols list.
Return values	symbol : String	
loadPuzzle (private)		
Parameters	filename : String	The method loads an external puzzle file using the filename parameter.
Return values		
		See 'Puzzle File Breakdown' for details on what the method does in an external puzzle file.
		The method sequences through the tasks from the filename parameter in the following order:
		Assigns noOfSymbols from the puzzle file and uses this value to iterate through the symbols from those lines in the allowedSymbols list.
		Assigns noOfPatterns from the puzzle file and uses this value to iterate through the patterns in each pattern. A pattern line in the puzzle file is a list – the first element is the symbol, the second element is the pattern, and the third element is a new pattern object to be added to the allowedPatterns list.
		Assigns gridSize from the puzzle file and uses the square of this value to iterate through the cells in each cell. A cell line in the puzzle file is a list – the first element is the symbol, the second element is the pattern, and the third element is a sub-list of symbols for the symbolsNotAllowed list. If the first element is an '@' symbol, the method appends a BlockedCell to the symbolsNotAllowed list. If the application instantiates a Cell object, the subsequent symbols not allowed are added to the symbolsNotAllowed list.
		Once all the cells have been processed, the method assigns the next line in the puzzle file to the symbolsNotAllowed list and the final line to the symbolsNotAllowed list.
		The method uses a try...catch block to handle any errors. If an error occurs, an error message is displayed and the method does not give the user the option to reload the file.

**Class: Pattern**

Identifier / Data		Description
<<constructor>>		
Parameters	symbolToUse : String patternString : String	Initialises the following private attributes: <ul style="list-style-type: none"><li>• symbol from parameter symbolToUse</li><li>• patternSequence from parameter patternString</li></ul>
Return values	n/a	
getPatternSequence (public)		
Parameters	n/a	Returns the value of the private attribute patternSequence
Return values	patternSequence : String	
matchesPattern (public)		
Parameters	patternString : String symbolPlaced : String	This is used to confirm that a pattern matches the passed parameter patternString through a 3 × 3 sequence of characters. The method iterates through the patternString attribute in a parameter patternSequence and compares it to the passed parameter symbolPlaced.
Return values	Boolean	
		<p>If the passed parameter symbolPlaced does not match the symbol for the pattern, the method returns false.</p> <p>If it does match, the method iterates through the patternSequence attribute and compares it to the passed parameter symbolPlaced at the same position in the parameter patternSequence. If the patternSequence contains symbolPlaced for the pattern, and the '*' character is not present to match in the pattern. The iteration continues to the next position in the patternSequence and parameter symbolPlaced. If the iteration reaches the end of the patternSequence and parameter symbolPlaced, the pattern match returns true. If the iteration reaches the end of the patternSequence and parameter symbolPlaced, the pattern match returns false. This iteration uses a try...catch to handle any out-of-range errors. If an out-of-range error occurs, the method displays an error message to the user.</p> <p>If the iteration completes, all the characters in the patternSequence have been matched and, therefore, the method returns true.</p>

COPYRIGHT  
PROTECTED

## Class: Cell

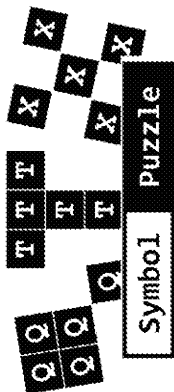
Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the protected attribute symbolsNotAllowed with an empty string list.
Return values	n/a	
addToNotAllowedSymbols (public)		
Parameters	symbolToAdd : String	Appends the parameter symbol to the private attribute symbolsNotAllowed.
Return values	n/a	
changeSymbolInCell (public)		
Parameters	newSymbol : String	Assigns the newSymbol parameter to the private attribute symbol.
Return values	n/a	
checkSymbolAllowed (public) <<virtual>>		
Parameters	symbolToCheck : String	Iterates through the private list symbolsNotAllowed. If the parameter symbolToCheck is in the list, the method returns false, otherwise it returns true.
Return values	Boolean	
getSymbol (public)		
Parameters	n/a	The method uses the isEmpty() method to check if the private attribute symbol is empty. If it is empty, the method returns the value of the protected attribute symbolsNotAllowed.
Return values	symbol : String	
isEmpty (public)		
Parameters	n/a	If the private attribute symbolsNotAllowed is empty, the method returns true, otherwise it returns false.
Return values	Boolean	
updateCell (public)		
Parameters	n/a	This method is not used in the puzzle. It has been included into the class as an option for a question which creates a new cell class and overrides its base class.
Return values	n/a	

## Class: BlockedCell (inherits from Cell)

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the parent attribute symbolsNotAllowed with an empty string list.
Return values	n/a	
checkSymbolAllowed (public) <<override>>		
Parameters	symbolToCheck : String	Overrides the checkSymbolAllowed method of the Cell class. While technically this could allow a blocked class object to bypass the symbolsNotAllowed list attribute, the method simply returns false regardless of the value of symbolToCheck.
Return values	Boolean	

COPYRIGHT  
PROTECTED





## Puzzle File Breakdown of Puzzle1

Line Number	Data	Description
1	3	This is the number of symbols in the game. It is used to iterate through the next three lines.
2	Q	FIRST SYMBOL. This is added to the AllowedSymbols list.
3	T	SECOND SYMBOL. This is added to the AllowedSymbols list.
4	X	THIRD SYMBOL. This is added to the AllowedSymbols list.
5	3	This is the number of patterns in the game. It is used to iterate through the next three lines.
6	Q,QQ**Q**QQ	FIRST PATTERN. This line is imported as a single string then split to lists using the comma as a delimiter. The first element is the Symbol for the pattern, and the second element is the PatternSequence. The PatternSequence represents the pattern as a helix of cells in a 3 x 3 section of the Grid.
7	X,X*X*X*X*X	SECOND PATTERN
8	T,TTT**T**T	THIRD PATTERN
9	5	This is the GridSize. A single value is used to denote the width and the height, i.e. the grid is square. This is used to calculate how many lines in the text files the code then needs to iterate through by multiplying it by itself, in this case 25.
10	Q,Q	The next 25 lines (because this game has a 5 x 5 grid) are the symbols in the grid. Each line has two elements. The first element is used to set the symbol in the cell (therefore, one symbol per cell).
11	Q,Q	The second element is what symbol can't be in the cell. This is used to stop two patterns of the same type overlapping. When this is imported,

COPYRIGHT  
PROTECTED



INSPECTION COPY

Line Number	Data	Description
17	,Q	This is a blank cell. In this example, however, the cell is within the 3 × 3 section of a matched pattern in the puzzle because it contains a SymbolsNotAllowed list.
18	,	
19	,	
20	,	
21	X,Q	This cell has an X symbol in it; however, the cell is within the 3 × 3 section of a matched pattern in the puzzle because it contains a SymbolsNotAllowed list. This is because the Q pattern was created before the X pattern; therefore, Q was used as the 'SymbolsNotAllowed' symbol.
22	Q,Q	
23	X,	
24	,	
25	,	
26	,	
27	X,	This is showing a cell in the grid during a game. It has an X in it which the user has put into it, but the pattern hasn't yet been complete; therefore, there are currently no symbols banned from this cell.
28	,	
29	,	
30	,	
31	X	
32	,	
33	,	

COPYRIGHT  
PROTECTED



INSPECTION COPY

## 'Errors' in the puzzle files

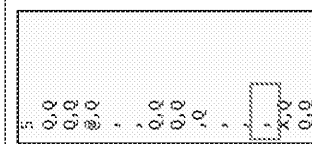
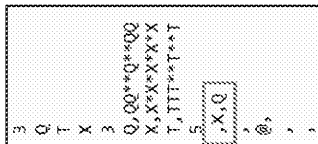
There are two logical errors that we have found in the puzzle files. These 'errors' have been checked with AQA, who have confirmed that the files operate as they expect them to or as required for the purpose of the exam. While they don't impact the main functionality of the application, the 'errors' generated unexpected behaviour at runtime.



This 'error' is demonstrated in puzzle 4. Line 10 of the file shows an empty cell being populated 'SymbolsNotAllowed' list with an X and a Q. The 'Symbols' 'Allowed' list is populated when a user matches a pattern. Under normal operation of the application, it should be logically impossible for a single cell to have populated 'SymbolsNotAllowed' list. The impact of this on the application is that when using puzzle 4, the user cannot place an X or a Q symbol into the cell at location 5.1.



INSPECTION COPY

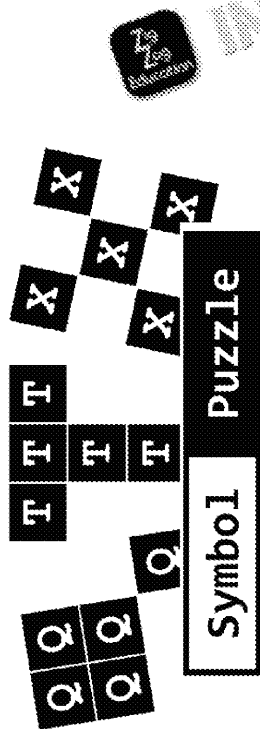


This 'error' is demonstrated in puzzles 1, 2 and 3. Line 20 of the file shows an empty cell but also an empty 'SymbolsNotAllowed' list. The puzzle files demonstrate that at a Q pattern has been matched with the top-left cell at location 5.<sup>4</sup> This should put a Q into the 'SymbolsNotAllowed' list at locations 5,1 to 3,3. Line 20 represents location 3,1 in the grid. Under normal operation of the application, it should be logically impossible for this cell to have a blank 'SymbolsNotAllowed' list after a pattern match. The impact of this on the application is that a user can place a Q symbol into location 3,1 when the application should not let them.

**COPYRIGHT  
PROTECTED**



INSPECTION COPY



## Advanced Techniques

These techniques are helper pieces of code to extend the Skeleton Program and functionality.

Note that the techniques demonstrated in this document are **BEYOND** the AQA 7517 specification. They **DO NOT** represent a mark scheme or an expected way of solving challenges presented for the public release material. The objective of this document is to extend students' knowledge and explore alternative techniques they could use as they experiment with the code. **Use of these techniques and ideas should be at the teacher's discretion.** The code shown below still demonstrates the 'wrap around' false positive matches identified by the standard skeleton code.

### Extension Technique 1 – Regular expressions

The Skeleton Program provided by AQA does not include the regular expression (regex) library and, therefore, it is unlikely that the QA would include a Section D question which uses regex. (*A regex question in Section C is perfectly possible.*) Although the library has not been included, students can import the library themselves and use it if they are confident with regex and feel that its use would aid their solutions.

To use this code you will not need to import additional regex libraries as Strings have four built-in methods for regular expressions:

- `* matches()`
- `* split()`
- `* replaceFirst()`
- `* replaceAll()`

COPYRIGHT  
PROTECTED



INSPECTION COPY

This expression can be extended further by defining duplicates of symbols:

```

public boolean checkForMatchWithPatternRegexBeyondStandard(String patternStringToCheck) {
    //These expressions comply with the techniques in the AQA specification
    //The hyphen must be last otherwise it is interpreted as a range
    String tCheck = "[3][Q|X|T|@|-]{2}T[Q|X|T|@|-]{2}T";
    String xCheck = "[Q|X|T|@|-]{4}X";
    String qCheck = "[Q]{2}[Q|X|T|@|-]{2}Q[Q|X|T|@|-]{2}Q";
    return patternStringToCheck.matches(tCheck) ||
        patternStringToCheck.matches(xCheck) ||
        patternStringToCheck.matches(qCheck);
}

```

### Extension Technique 3 – Reverse referencing the grid

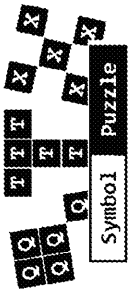
The skeleton code includes a method called `getCell` which takes the parameters of row and column and returns the Cell at the associated location in the grid data structure. This method reverses that process, allowing the user to pass the reference of a location in the grid data structure, and it returns an integer array containing the row and column for that cell.

```
private int[] getCellFromIndex(int index) {  
    if (index < 0 || index >= grid.size()) {  
        return null;  
    }  
    int result = index / gridSize;  
    int row = (gridSize - 1) - result;  
    int column = index % gridSize;  
    return new int[] {row, column+1};  
}
```

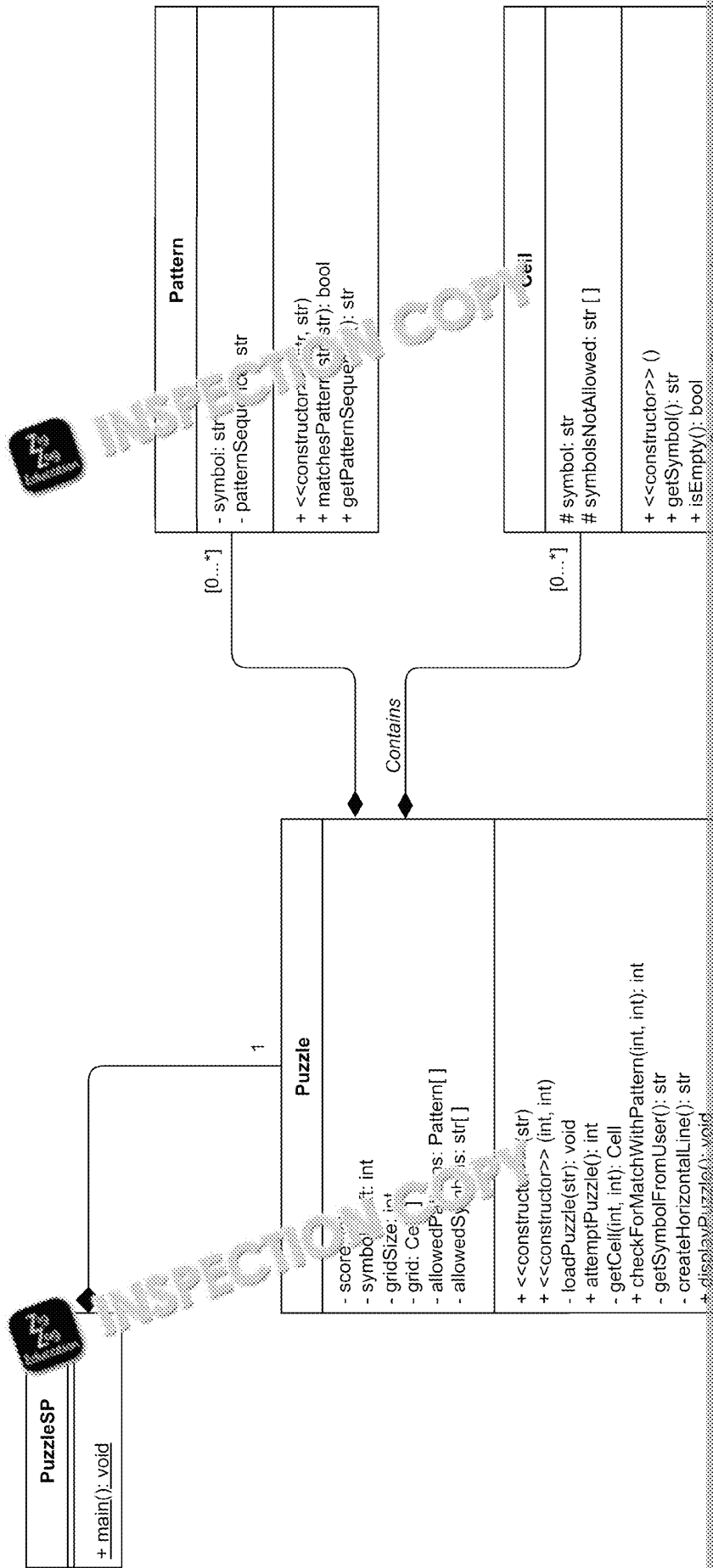
COPYRIGHT  
PROTECTED



INSPECTION COPY



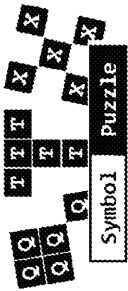
# UML Class Diagram



COPYRIGHT  
PROTECTED

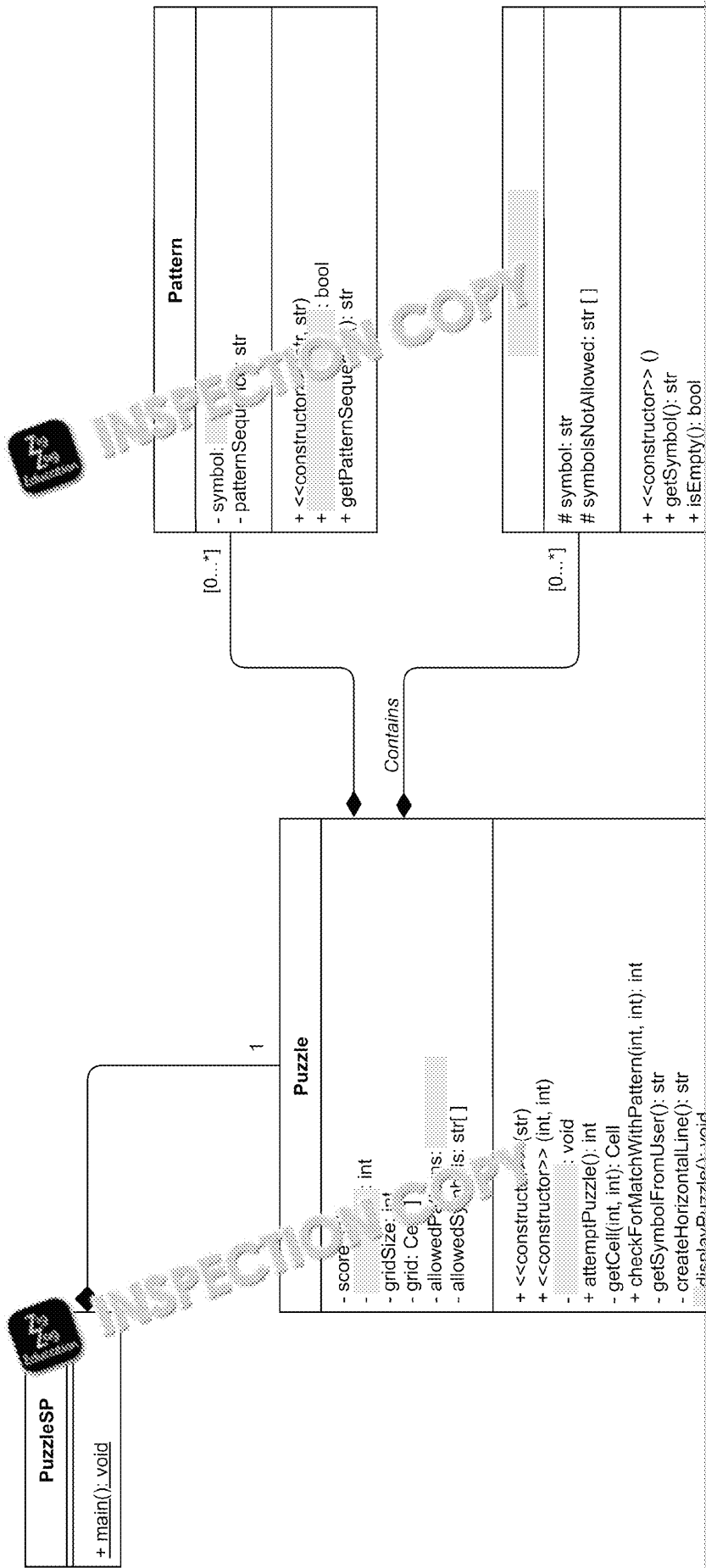


INSPECTION COPY



# UML Class Diagram

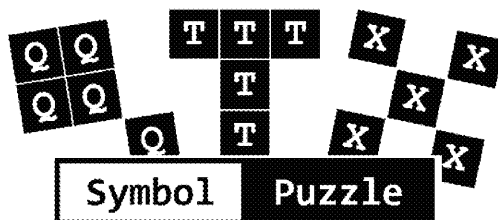
## Activity



COPYRIGHT  
PROTECTED



INSPECTION COPY



## Theory Questions

These questions are designed to test your understanding of the skeleton code and to the kinds of question you can expect to see in Section C of the Paper 1. These questions are more than 2 marks and are rarely seen in this section – these are here to challenge your understanding of the code.



These questions refer to the **Preliminary Material** and the **Skeleton Code** but **do not** require any additional programming.

**TOTAL MARKS: 75**

1 This question refers to the main method in the PuzzleSP class and the code below.

- (a) Describe the purpose of the selection statement below, including the parameters for the Puzzle constructor:

```
if (filename.length() > 0) {
    myPuzzle = new Puzzle(filename + ".txt");
} else {
    myPuzzle = new Puzzle(8, (int)(8 * 8 * 8));
}
```

---

---

---

---

---

---

---

---

---

---

- (b) Many languages such as C# and Java, allow methods of the same name to be defined in the same class.

State the name of this OOP technique.

---

**COPYRIGHT  
PROTECTED**



- 2 This question refers to the entire code. Throughout the code, several hard-coded (see two examples below).

Example 1:

```
myPuzzle = new Puzzle(8, (int)(8 * 8 * 0.6))
```

Example 2:

```
if (getRandomInt(1, 101) < 90) {
```

- (a) State two reasons why constants would be more appropriate.

.....

.....

.....

- (b) The code is written using the object-oriented paradigm.

Discuss two advantages of this over the traditional structured approach.

.....

.....

.....

.....

.....

.....

- 3 This question refers to the `checkForMatchWithPattern` method of the `Puzzle` class. The pattern string used to match the Q pattern in the puzzle is: `"QQ**Q**"`.

- (a) Explain how a successful match is determined.

.....

.....

.....

- (b) Explain how an unsuccessful match is determined.

.....

.....

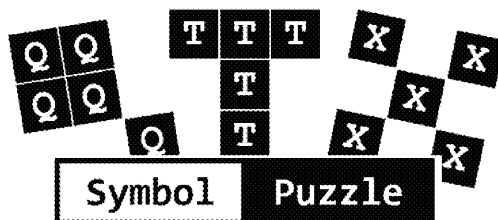
- (c) Explain how the program is prevented from placing any more letter Q's on the board once a successful match has been determined.

.....

.....

**COPYRIGHT  
PROTECTED**





## Theory Questions

These questions are designed to test your understanding of the skeleton code and to the kinds of question you can expect to see in Section C of the Paper 1. These questions are more than 2 marks and are seen in this section – these are here to challenge your understanding of the code.



These questions refer to the **Preliminary Material** and the **Skeleton Code** but **do not** require any additional programming.

**TOTAL MARKS: 75**

1 This question refers to the main method in the PuzzleSP class and the code below.

- (a) Describe the purpose of the selection statement below, including the parameters for the Puzzle constructor:

```
if (filename.length() > 0) {
    myPuzzle = new Puzzle(filename + ".txt");
} else {
    myPuzzle = new Puzzle(8, (int)(8 * 8 * 0.6));
}
```

- (b) Many languages, such as C# and Java, allow methods of the same signatures to be defined in the same class.

State the name of this OOP technique.

2 This question refers to the entire code. Throughout the code, several hard-coded (see two examples below).

Example 1:

```
myPuzzle = new Puzzle(8, (int)(8 * 8 * 0.6));
```

Example 2:

```
if (getRandomInt(0, 100) < 90) {
```

- (a) State two reasons why constants would be more appropriate.  
(b) The code is written using the object-oriented paradigm.

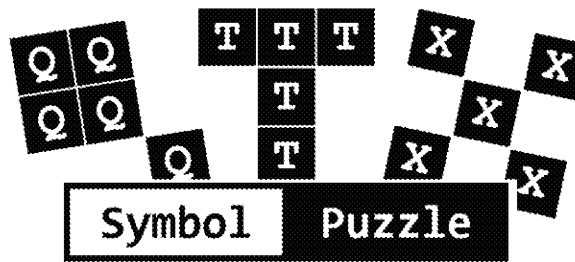
Describe two advantages of this over the traditional structured approach.

3 This question refers to the checkForMatchWithPattern method of the Puzzle class. The pattern string used to match the Q pattern in the puzzle is: "QQ\*\*Q\*\*"

- (a) Explain how a successful match is determined.  
(b) Explain how an unsuccessful match is determined.  
(c) Explain how players are prevented from placing any more letter Q once a successful match has been determined.

**COPYRIGHT  
PROTECTED**





## Programming Tasks

These questions require you to learn the **Skeleton Program** and to make

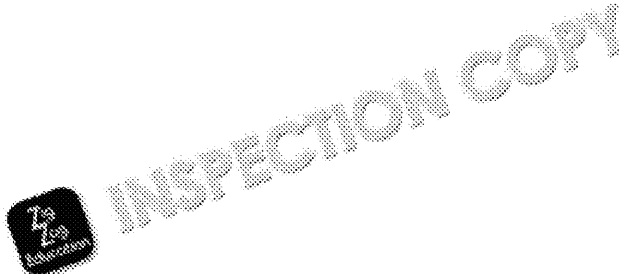
*Note that any alternative or additional code changes that are deemed appropriate should be clearly marked, so that it is clear where in the Skeleton Program those changes are made.*



AQA has highlighted in the pre-release material that there are errors in the implementation of the Skeleton Program, meaning that it does not work as described under all circumstances. Therefore, any anomalous output as a result of these errors will also be present in this resource.

The objective of this resource is to provide you with a selection of different questions. Questions which may have a similar theme may use different techniques or options on how to solve problems. Some questions may contain a wider range of code than is realistic in an exam situation to stretch students – in particular Questions 1 and 2 are more prescriptive than others in how the task should be completed in order to achieve the required solutions presented in this resource only use techniques which are included in the pre-release material.

Students are recommended to start with a clean copy of the pre-release code for each question in this resource. This will prevent modifications made for one question from affecting the code for a different question.



**COPYRIGHT  
PROTECTED**



## Task 1

This question refers to the `Puzzle` class.

Introduce a new pattern option into the constructor of the `Puzzle` class for

### What you need to do

#### Task 1.1

Add a new pattern option into the puzzle to allow the pattern shown in Figure 1 to be used in a standard puzzle.

*(This new pattern option is not expected to work with the default puzzle file supplied by AQA.)*



#### Task 1.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to start a standard puzzle.
- Input the new pattern into an available space on the grid.
- Show the program displaying the new pattern in a standard puzzle and 10 points.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the introduction of a new pattern into the constructor of the `Puzzle` class.
- SCREEN CAPTURE(S) showing the required test.



INSPECTION COPY

COPYRIGHT  
PROTECTED



## Task 2

If symbols are placed into the grid in the correct order, it is possible to get patterns which use the same symbol to overlap as shown in the example in Figure 1. In this example, the symbols are entered from the top row down, thereby matching the top T pattern and awarding the user 10 points. When the final T symbol is entered at location 3,5 the lower T pattern is matched, using some of the upper T pattern cells, gaining another 10 points. The legality of this type of move in the pre-release material is not specified and it technically contravenes the rule that when the user has successfully created an allowed pattern, they can no longer place the symbol used in the pattern in any of the other cells in that  $3 \times 3$  section.

	1
8	
7	
6	
5	
4	
3	
2	
1	

This question modifies this logical error in the puzzle so that overlapping patterns created in this way cannot be reused for multiple patterns.

The program should still allow the user to place symbols into the grid in the overlap effect, but should only award points for a single matched pattern.

### What you need to do

#### Task 2.1

Modify the `checkForMatchWithPattern` method in the `Puzzle` class to identify a helix, which is not blocked, has already been used in a pattern, and if so, return false.

#### Task 2.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to create a standard puzzle. (This may need repeating until you have a space in the grid to enter the overlapping T patterns shown in Figure 1.)
- Input a T at grid locations 6, 4    6, 5    6, 6    5, 4    5, 5  
(These grid locations may need adjusting to suit the space available in the grid.)
- Show the program displaying the puzzle with overlapping T patterns as shown in Figure 1.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the amended method `checkForMatchWithPattern` in the `Puzzle` class.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**





## Task 4

This question extends the Skeleton Program to place limits on the number of patterns which the user can place in each puzzle. Currently a user can place any pattern, subject to having enough symbols and appropriate space in the grid.

Modify the program to select a value at random, up to 3, for the amount of each pattern type that a user can place into the grid. Each pattern limit may be different. The limits should be set by the number of symbols. This functionality should only be applicable to standard puzzles. The user can still continue to place patterns until the puzzle is drawn onto the screen of how many of each pattern type.

When a program matches a correctly placed pattern in the grid, the number of patterns available for that type of pattern should be decremented. The user can still continue to place patterns of that type (subject to having enough symbols left), but the program should ensure that the limit is not exceeded.

What you need to do:

### Task 4.1

Modify the constructor in the `Puzzle` class to pass an additional parameter (representing the number of each pattern type inclusive) for each pattern when it is instantiated. This is the number of each pattern type as described.

### Task 4.2

Modify the `Pattern` class to use this additional parameter to limit the number of each pattern type placed into the grid. Store this value in a new property called `patternCount`. The program should enforce restrictions on a standard puzzle. `patternCount` for the associated pattern type. The number of each valid pattern is placed.

### Task 4.3

Create a new method `outputPatternCount` in the `Pattern` class which displays the number of each pattern type onto the screen stating the limit for each pattern type.

### Task 4.4

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to create a standard puzzle.
- Show the program displaying the number of each pattern type available.
- Input a T pattern at a suitable location.
- Show the program displaying the reduction in the number of T patterns available.
- Repeat the above to use all the T patterns available.
- Show the program giving a suitable error message when the user attempts to place a T pattern when no T patterns are available.

### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the amended constructor in the `Puzzle` class, the new method `outputPatternCount` and any other methods you have modified or created when answering this question.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 5

This question extends the Skeleton Program to allow the user to remove a symbol and increase the number of symbols remaining.

A symbol can only be moved if it is not already part of a pattern or is a block. If the user attempts to move a symbol which is already part of a pattern or is blocked, the program should display a suitable error message.

The program should display the number of `symbolsLeft` after the current symbol is removed. The program should ask the user if they would like to remove a symbol. If they agree, the program should prompt the user for the location of the symbol they want to remove. If the user enters a valid location, remove that cell from the grid and decrement the number of `symbolsLeft`. If the user enters a location which is already part of a pattern or is blocked or contains a block, the program should display a suitable error message.



### What you need to do

#### Task 5.1

Modify the `attemptPuzzle` method in the `Puzzle` class to display the number of symbols remaining and prompt the user to remove a symbol in the way described.

#### Task 5.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle3`.
- Confirm that you want to remove a symbol when prompted to do so.
- Remove the X symbol from grid location 2, 3.
- Show the program displaying the symbol removed from the grid and the number of symbols remaining.
- Attempt to remove the Q symbol from grid location 5, 1.
- Show the program displaying a suitable error message.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the amended `attemptPuzzle` method in the `Puzzle` class and any other methods you have modified or created within this question.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 6

This question extends the Skeleton Program to allow the user to save a puzzle.

The program should give the user the option to save the current state of the puzzle after each turn. On confirmation, the program should ask the user for a filename and extension, which the program should append automatically. The program should ensure the filename and location is valid and is a new file. The program should collect together the moves for a puzzle in the correct order and save a valid puzzle file.

### What you need to do

#### Task 6.1

Create a new method called `savePuzzle` in the `Puzzle` class which saves the puzzle to a file per the layout of puzzle files supplied by AQA.

#### Task 6.2

Modify the `attemptPuzzle` method in the `Puzzle` class to prompt the user if they like to save the current puzzle so that if the user confirms, the new method `savePuzzle` is called and the puzzle is saved correctly.

#### Task 6.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle3`.
- Input an X at grid location 1, 4.
- Enter the filename `MySavedPuzzle`.
- Show a screenshot of the exported text file.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the new method `savePuzzle` and any other methods you have modified or created while answering this question.
- SCREEN CAPTURE(S) showing the required test.

INSPECTION COPY

COPYRIGHT  
PROTECTED



## Task 7

This question extends the Skeleton Program by allowing the user to undo previous moves. The user should be able to undo as many previous moves as they have played. A move is a previous move where points were awarded for a pattern match does not need to be in the score.

Introduce new functionality to store moves made by the user in an appropriate data structure so that they can be undone. Before each turn, the user should be offered the option to undo previous moves. If no previous moves are available, the program should operate as normal. If the user chooses to undo a move, the program should display to the user how many moves they can undo. If the user selects to undo, the program should undo the previous move. Score changes should be taken into consideration.

**What you need to do**

### Task 7.1



Create a new class `PreviousMove` which inherits `Cell`. A `PreviousMove` should have the same properties together with accessor and mutator methods to store the location of the move. All appropriate information about a cell needs to be stored, such as `symbolsNotAllowed` list.

### Task 7.2

Create a new method in the `Puzzle` class called `undoPreviousMove` which does what is described.

### Task 7.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle3`.
- Input a `T` at grid location `4, 4`.
- Show the program displaying the updated grid with the new symbol placed.
- Confirm to undo previous move when prompted at next turn.
- Show the program displaying the updated grid with the symbol removed.

#### Evidence that you need to provide

- Your **PROGRAM SOURCE CODE** for the new class `PreviousMove`.
- Your **PROGRAM SOURCE CODE** for the new method `undoPreviousMove` and any other methods you have amended method `attemptPuzzle` and any other methods you have created when answering this question.
- **SCREEN CAPTURE(S)** showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 8

This question extends the Skeleton Program by allowing the user to move new random locations during the puzzle.

New functionality should be introduced which offers the chance to reshuffle after a pattern has been successfully matched on the grid. On selecting this move all the blocked cells to new random locations in the grid. A blocked cell where there was one originally and can only be placed in an empty cell.

### What you need to do

#### Task 8.1

Create a new method in the `Puzzle` class called `reShuffleBlockedCells` with the signature described.

#### Task 8.2

Modify the `attemptPuzzle` method in the `Puzzle` class to call the new `reShuffleBlockedCells` method when a pattern has been successfully matched in the grid.

#### Task 8.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle4`.
- Input a `T` at grid locations `4,2` `4,3` `4,4` `3,3` `2,3`.
- Show the program displaying the updated grid with the completed pattern.
- Select the option to move each `BlockedCell` to a new random location.
- Show the program displaying the updated grid with the random `BlockedCells`.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the new method `reShuffleBlockedCells` and the modified method `attemptPuzzle`.
- SCREEN CAPTURE(S) showing the required test.

INSPECTION COPY

COPYRIGHT  
PROTECTED



## Task 9

This question extends the Skeleton Program by introducing a double points functionality should only be available when a standard puzzle is generated. It should be added to a grid at random empty locations. A double points cell is shown below. If a pattern is matched and any of the symbols within that pattern are on a double points cell, the score is awarded 20 points rather than 10.

### What you need to do

#### Task 9.1

Create a new class `DoublePointCell` which should inherit from the `Cell` class. It should have the symbol 'D' and its `isDouble` method should have an overridden method called `isDouble`. Include the required `isDouble` method in the `Cell` class to allow this to operate.

#### Task 9.2

Modify the appropriate constructor for the `Puzzle` class for a standard puzzle to add double points cells to the grid at empty locations.

#### Task 9.3

Modify the `checkForMatchWithPatterns` method in the `Puzzle` class so that it checks for patterns on a double points cell in the way described.

#### Task 9.4

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to create a standard puzzle.
- Input symbols to create a pattern, either 'Q', 'T' or 'X', ensuring that at least one symbol is on a `DoublePointCell` cell.
- Show the program displaying the score of 20 when the pattern is matched.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the new class `DoublePointCell`, the constructor in the `Puzzle` class and the amended `checkForMatchWithPatterns` method.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 10

This question extends the Skeleton Program to include error handling. The that the Skeleton Program should allow a user to place a different symbol in empty cells within the  $3 \times 3$  pattern section. The implementation of the program should allow a user to place a different symbol into any cell within the  $3 \times 3$  section of a matched pattern. The program should also handle a blocked cell. Additionally, the program gives an unhandled exception if the user attempts to place a symbol outside the bounds of the grid. If the user attempts to enter a symbol at grid location  $\text{GridSize} + 1$ ,  $\text{GridSize} + 1$ , the program places the symbol at location  $\text{GridSize}$ ,  $\text{GridSize}$ . The standard puzzle grid is location 8, 1 and with one of the test puzzle files is

The program should give suitable error messages under the following conditions:

- 1. If the user enters a character that is not a digit, decrement the number of symbols left and instead just give the user a prompt.

- The user attempts to place the same symbol into **any** cell within the 3 × 3 grid.
- The user attempts to place a different symbol into any **non-empty** cell.
- The user attempts to enter a grid location which is outside the bounds of the 3 × 3 grid.

## What you need to do

### Task 10.1

Modify the `Cell` class to include appropriate accessor and mutator methods for each of the attributes in the following pattern.

### Task 10.2

Modify the `attemptPuzzle` method in the `Puzzle` class to advise the user he is out of left and introduce suitable error handling so that the user cannot place a symbol as described.

### Task 10.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle3`.
- Attempt to input a Q at grid location 4, 3.
- Show the program giving an error message, prompting the user to try a different symbol, with symbols not changing.
- Attempt to input an X at grid location 5, 1.
- Show the program giving an error message, prompting the user to try a different symbol, with symbols not changing.
- Attempt to input an X at grid location 9, 9.
- Show the program giving an error message, prompting the user to try again.

**Evidence that you need to provide:**

- Your PROGRAM SOURCE CODE for the amended Cell class and the attemptPuzzle in the Puzzle class and any other methods you have answering this question.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 11

This question refers to the `attemptPuzzle` method.

Introduce new functionality to award the user a lever. A lever can remove a blocked cell from the grid, allowing the user to then place a normal symbol into that location. A lever is awarded to the user after they have successfully placed their first valid pattern. The lever can be used to use the lever once in a puzzle. On selecting to use the lever, the user enters the grid location of the blocked cell which they want to remove. A valid location is one that contains a blocked cell. Assuming it is a valid location, replace the blocked cell with a lever, allowing the user to place a symbol at that location in their next turn.

### What you need to do

#### Task 11.1

Modify the `checkForMatchWithPattern` method to award the user with a lever when a valid pattern is found.

#### Task 11.2

Modify the `attemptPuzzle` method so that it operates in the way described below.

#### Task 11.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle3`.
- Input an `X` at grid location 1, 4.
- Select to use the lever when prompted.
- Enter grid location 5, 3.
- Show the program displaying the grid with the `BlockedCell` removed.
- Input an `X` at grid location 5, 3.
- Show the program displaying the symbol `X` at the new location.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the modified `attemptPuzzle`, `checkForMatchWithPattern` methods and any other methods you have created when answering this question.
- SCREEN CAPTURE(S) showing the required test.

INSPECTION COPY

COPYRIGHT  
PROTECTED



## Task 12

This question refers to the `attemptPuzzle` method in the `Puzzle` class and the `BlockedCell` class to create a swamp of blocked cells. A swamp is an individual cell with the symbol '!'. Multiple swamps can be placed into the grid to make placement more difficult.

Introduce new functionality to fill some empty space on the grid with swamps at random locations in the grid. The swamp can only be placed into a cell which currently contains the symbol '!' to identify the cells as being a swamp rather than a normal blocked cell. A random number of empty cells (between 1 and 4 inclusive). There should be a 50% chance of being triggered in each turn. The user must be given a choice between 2 and 4 (choice of a swamp event). Swamps can only be triggered once in a puzzle.

### What you need to do

#### Task 12.1

Modify the `BlockedCell` class to allow the symbol to be updated to '!' by overriding the `update` method.

#### Task 12.2

Create a new method in the `Puzzle` class called `swampThisCell` which inserts a swamp into the grid in the way described.

#### Task 12.3

Modify the `attemptPuzzle` method to operate in the way described.

#### Task 12.4

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle4`.
- Place symbols into the grid until a swamp is triggered.
- Show the program advising the user that a swamp has been triggered, the swamp happens and how many cells will be filled as swamps.
- Continue to place symbols into the grid until the swamp happens.
- Show the program displaying the correct number of randomly placed swamps.

#### Evidence that you need to provide:

- Your **PROGRAM SOURCE CODE** for the new method `swampThisCell` and the modified method `attemptPuzzle`.
- **SCREEN CAPTURE(S)** showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 13

This question extends the Skeleton Program by checking if there are enough symbols left to place another pattern into the grid. If there are not enough symbols left to place a pattern into the grid, the number of symbolsLeft should be deducted from the puzzle finishes. This question does not correct the error in the Skeleton Program where symbols to be placed on top of currently completed patterns or cells with the same symbol in multiple pattern matches, but it does need to take into account the normal rule that the system cannot attempt to place a symbol into a blocked cell.

After each turn the program should calculate how many patterns there are which ones match allowed patterns. For patterns which are only partially matched, the test to see how many symbols are currently in the right place and whether it is possible to complete the pattern.

If there are not enough symbols left to complete a pattern in the grid, the puzzle should finish on the user's next turn. If there are not enough symbols, the number of symbolsLeft should be deducted from the score and the puzzle should finish. The program should tell the user what the final score is.

### What you need to do

#### Task 13.1

Create a new method in the Puzzle class called `getSpaceLeftOnGrid` that returns the number of enough symbols left to add a new pattern to the grid in the way described.

#### Task 13.2

Modify the `attemptPuzzle` method in the Puzzle class which uses the new `getSpaceLeftOnGrid` to check if the user can enter another pattern as described.

#### Task 13.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle2`.
- Input an X at grid location 1, 4.
- Show the program advising the user that there are not enough symbols left to complete patterns in this grid and 2 points will be deducted from their final score.
- Show the program displaying a finished puzzle with a final score of 18.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the new method `getSpaceLeftOnGrid` and the modified method `attemptPuzzle` in the Puzzle class and any other methods you use when answering this question.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**




## Task 14

This question modifies the Skeleton Program so that it identifies errors in the files. The files in these files generate logic errors when trying to place symbols into the grid.

The files contain two types of error:

- Error 1: A cell contains a symbolsNotAllowed list but does not appear in the file.
- Error 2: A cell does not contain a symbolsNotAllowed list when it appears in the file.

Introduce new functionality which, after loading a puzzle file, checks the data for errors. The program should advise the user if the file does contain an error. It does not need to be able to fix the error.

Puzzle Files	Error	
Puzzle4.txt 	<pre> 3 Q T X 3 Q,QQ**Q**QQ X,X*X*X*X*X T,TTT**T**T 5 ,X,Q , @, , ,           </pre> <p><b>Error 1</b></p>	The symbolsNotAllowed list for the cell (5,1) contains a possible symbol that is not in the file. The program should generate an error message to the user.
Puzzle1.txt Puzzle2.txt Puzzle3.txt	<pre> 5 Q,Q Q,Q @,Q , , Q,Q Q,Q ,Q , , ,X,Q ,Q           </pre> <p><b>Error 2</b></p>	The symbolsNotAllowed list for the cell (5,1) contains a possible symbol that is not in the file. The program should generate an error message to the user.

INSPECTION COPY

COPYRIGHT  
PROTECTED



## What you need to do

### Task 14.1

Create two new methods called `testForError1` and `testForError2` that identify cells which contain the import errors highlighted on the previous page.

Error 1 is demonstrated if cells:

- contain a populated `symbolsNotAllowedList` and are not a blocked cell
- the game score is zero

Error 2 is demonstrated if:

- the number of cells containing a `symbolsNotAllowedList` is less than the given score

### Task 14.2

Modify the `loadPuzzle` method in the `Puzzle` class to call the `testForError1` and `testForError2` methods once all the puzzle data has been loaded, and display the results in the window.

### Task 14.3

Create a new method `containsSymbolsNotAllowedList` in the `Cell` class that returns `true` if the cell is a blocked cell or if the `symbolsNotAllowedList` contains at least one element or an empty string.

### Task 14.4

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle4`.
- Show the program indicating that the puzzle demonstrates error 1.
- Run the Skeleton Program again.
- Enter `puzzle1`.
- Show the program indicating that the puzzle demonstrates error 2.

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the new methods `testForError1` and `testForError2` in the `Puzzle` class and the amended `loadPuzzle` method together with the `containsSymbolsNotAllowedList` method in the `Cell` class.
- SCREEN CAPTURE(S) showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 15

This question extends the Skeleton Program to allow rotated patterns to be entered. In the pre-release material, the puzzle will only match a pattern which has been entered in a normal configuration; for example, the Q pattern shown in Figure 1.

Introduce new functionality into the program to allow the puzzle to additionally match patterns rotated at 90, 180 and 270 degrees; for example, the Q patterns shown in figures 2, 3 and 4. If the user correctly places a rotated pattern, they should be awarded 15 points rather than just 10. *Due to the wrap-around error in the original rotations may be identified as false positives. This question does not need to be changed.*

Figure 2

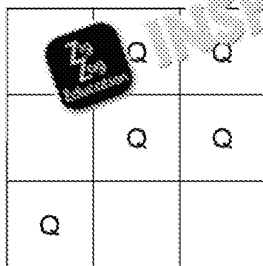
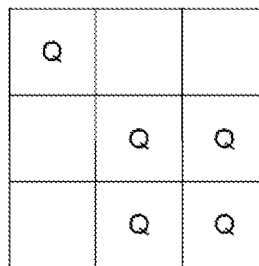


Figure 3



Character positions for a normal Q pattern:

1	2	3	4	5	6	7
Q	Q	*	*	Q	*	

A 90 degree rotation uses the normal pattern sequence in the following order: **chars 7 and 8 followed by the first 6 chars followed by the final char, e.g.:**

A 180 degree rotation uses the normal pattern sequence in the following order: **chars 5 to 8 followed by the first 4 chars followed by the final char, e.g.:**

A 270 degree rotation uses the normal pattern sequence in the following order: **chars 3 to 9 followed by the first 2 chars, e.g.:**

### What you need to do

#### Task 15.1

Create a new method called `matchesPatternRotated` in the `Puzzle` class to check for a pattern match as described above.

#### Task 15.2

Modify the `checkForMatchWithPattern` method in the `Puzzle` class to add support for rotated patterns by calling the `matchesPatternRotated` method for each pattern of the puzzle. If a rotated pattern is found, award the user 15 points.

INSPECTION COPY

COPYRIGHT  
PROTECTED



### Task 15.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle4`.
- Input a T pattern rotated through 90 degrees at a suitable location.
- Show the program awarding a score of 15 when the rotated pattern is

#### Evidence that you need to provide:

- Your PROGRAM SOURCE CODE for the `checkForMatch` method matches `Pattern` amended method `checkForMatch` with the pattern and any other methods created when answering this question.
- SCREEN CAPTURE(S) showing the required test.

INSPECTION COPY

COPYRIGHT  
PROTECTED



## Task 16

This question modifies the Skeleton Program to adjust how scoring is handled. Symbols are placed into a non-empty cell which has already been matched and scored. A pre-release material indicates that this should not be possible; however, this is now implemented. In the current implementation, if the user places a different symbol which is part of a pattern, the cell symbol is replaced and the user does not lose the score for that pattern, even though the pattern is no longer correct.

Introduce new functionality which removes points awarded to a player if they place a symbol into a non-empty cell which is part of a pattern, thereby making the pattern invalid.

### What you need to do

#### Task 16.1

Modify the `Cell` class to include appropriate mutator and accessor methods `unset` and `set` to allow a symbol to be part of a pattern when a new pattern is matched. Modify the `mutator` method to remove a symbol from the `symbolsNotAllowedList`.

#### Task 16.2

Modify the `checkForMatchWithPattern` method to pass in an additional parameter to indicate if the method should operate as normal or to remove symbols from the `symbolsNotAllowedList` and decrease the score. Modify the `checkForMatch` method to use this flag and operate as described.

#### Task 16.3

Create a new method in the `Puzzle` class called `removeSymbol` which should be called from the `attemptPuzzle` method and change the symbol and update score as described.

#### Task 16.4

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to create a standard puzzle.
- Input a valid `T` pattern into the grid at an available location.
- Show the program displaying the updated score of 10.
- Input a `Q` symbol into the grid into a cell containing one of the `T` symbols.
- Show the program displaying the updated score of 0.

#### Evidence that you need to provide.

- Your `PROGRAM SOURCE CODE` for the modifications in the `Cell` class.
- Your `PROGRAM SOURCE CODE` for the new method `removeSymbol`, the `attemptPuzzle` method and any other methods you have modified or added for this question.
- `SCREEN CAPTURE(S)` showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 17

This question extends the Skeleton Program by correcting the error in the symbols of different types to be placed into non-empty cells which have already scored as part of a pattern. This error, however, should be corrected to allow a cell. When the user attempts to place a different symbol onto a non-empty cell which has already been matched and scored as part of a pattern, the program should instead place a wild card 'W' and can represent any symbol in the allowedSymbols set. If a cell has already been matched which includes a wild card it should gain 15 points.

A wild card cannot be placed onto a blocked cell. A user can have only two wild cards. The program should tell the user how many wild cards they have left at each attempt. If both wild cards are placed, give a suitable error message if they attempt to place a symbol into a non-empty cell which has already been matched and scored as part of a pattern.

**What you need to do**

### Task 17.1

Create a new `WildcardCell` class which inherits `Cell`. It should have the same methods as `Cell` but should have an appropriate accessor method to identify it as a wild card.

### Task 17.2

Modify the `attemptPuzzle` method in the `Puzzle` class and any other methods that handle attempts to place a different symbol into a non-empty cell which has already been matched as part of a pattern, advise the user that the cell is part of a pattern and place a wild card 'W' at that location instead which should operate in the way described.

### Task 17.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `puzzle3`.
- Input an `X` at grid location `1, 4`.
- Input a `T` at grid locations `4, 3` `4, 4` `4, 5` `3, 4` `2, 4`.
- Show the program displaying the puzzle with a `T` pattern overlapping the `X` at grid location `3, 4` and a score of 35.

#### Evidence that you need to provide:

- Your **PROGRAM SOURCE CODE** for the new class `WildcardCell` and any methods you have modified when answering this question.
- Your **PROGRAM SOURCE CODE** for the amended method `attemptPuzzle` and any other methods you have modified or created when answering this question.
- **SCREEN CAPTURE(S)** showing the required test.

**COPYRIGHT  
PROTECTED**



## Task 18

*Warning! This question requires the student to generate a large amount of realistic in an exam scenario.*

This question extends the program by showing the user a possible solution score possible from placing legal patterns into the grid. This should operate for an empty grid. This question is **not** designed to auto complete a partially

A legally placed pattern cannot overlap with another pattern of the same symbol types can overlap, but not if the overlap is on non-empty or matched and scored as part of a pattern.

The program should calculate possible solutions for achieving the highest score can assume an unlimited number of patterns. Multiple solutions may achieve The program only needs to show one of them to the user together with the

After loading an empty puzzle file (puzzle file 4), the program should ask the user to auto complete the puzzle. The program should calculate possible solutions and display one of the highest scoring possible solutions to the user together with the program should then finish.

### What you need to do

#### Task 18.1

Create a new `TempPuzzle` class which should contain all the required properties for a puzzle together with the associated accessor and mutator methods.

#### Task 18.2

Create a new method in the `Puzzle` class called `checkAllCombinations` which will check all combinations of 'Q', 'X' and 'T' patterns in the grid to work in the way described in the specification. It should return a suitable data structure which contains all the possible combinations together with the score of the highest scoring possible solutions to the user together with the score.

#### Task 18.3

Test that the changes you have made work:

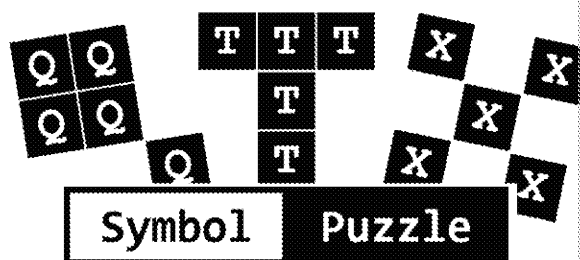
- Run the Skeleton Program.
- Enter `puzzle4`.
- Confirm that you would like the program to auto complete the puzzle with the highest scoring possible solution.
- Show the program displaying the completed puzzle with an overlapping score of 20.

#### Evidence that you need to provide:

- Your **PROGRAM SOURCE CODE** for the new class `TempPuzzle`.
- Your **PROGRAM SOURCE CODE** for the amended method `checkAllCombinations` in the `Puzzle` class and any other methods modified or created when answering this question.
- **SCREEN CAPTURE(S)** showing the required test.

**COPYRIGHT  
PROTECTED**

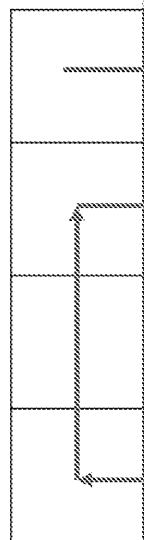




## Programming Tasks (Extension)

### Extension 1

The current helix pattern sequence is a representation of a  $3 \times 3$  section of the grid. This limits the range of letters which can be represented with the space available for symbols. Introduce new functionality to allow the application to use a  $4 \times 4$  section of the grid to represent a pattern sequence. Use this sequence to draw the helix pattern:



### Extension 2

The application currently is a single-player puzzle. Introduce new functionality to allow the application to be played by two players. A two-player puzzle should only be playable in a standard mode. Players should take turns to place symbols into the grid. Although both players can place symbols, only player 1 places upper-case symbols. Player 2 places lower-case symbols. Players gain points by placing either a complete pattern (for player 1) or a complete, valid lower-case pattern (for player 2). A pattern and lower-case symbols is not valid. The game ends when the symbols have been placed. The winner is the player with the highest score.

COPYRIGHT  
PROTECTED

### Extension 3

Currently the user must decide where they would like to place a symbol to complete a pattern. Introduce new functionality for the computer to 'suggest a move'. This costs the user 5 points. Offer the user the option to 'view a move suggestion'; the computer should suggest a move where a completed pattern (any of the symbols) can be placed into the grid. The user can accept or reject the suggestion (option here for making the suggested option a different colour). After a suggestion, the console is cleared, the user score is reduced by 5 and the game state is updated, allowing the user to then place symbols into the grid.



## **Preview of Questions Ends Here**

---

This is a limited inspection copy. Sample of questions ends here to avoid students previewing questions before they are set. See contents page for details of the rest of the resource.

Question	Answer	Mark	Guidance
10 a	A list is a dynamic data structure while an array is static // the size of a list can be changed during execution but the size of an array cannot [1]. A list can contain elements of various data types whereas an array has elements all of same data type [1].	1	MAX 1
10 b	An array is a static data structure whereas a list is dynamic so it will be stored as a single contiguous memory [1], needing more time efficient [1]. Using two dimensional arrays is more intuitive as it matches the structure shown to the user [1] whereas a one-dimensional list requires a conversion from the row and column input to give the cell [1]. The two-dimensional array will pick up indexing errors correctly [1] whereas the one-dimensional list will not always as it will frequently just return a different element [1].	4	MAX 4
11 a	All possible 3 x 3 sequences around the square upon which the player has placed a symbol are checked (using a nested for loop). // For each 3 x 3 grid, a string is collected in a hex shape to generate a string to be used for comparison [1]. If a match is found then the function/method returns 10 which stops any further matches from being detected [1].	2	
11 b	(3,3) (3,5) (1,1) (1,5) OR (3,3) (3,5) (1,5) (1,1) OR (3,3) (1,1) (3,5) (1,5)		Use whichever solution allows most matches from the start. Award 1 mark for two correct in order (stop when wrong match) and 2 marks for all four correct.
12 a	$O(n!)$	1	
12 b	This is a problem in which all combinations need to be attempted [1]. This means factorial time complexity as when a new possibility is added it must be tried with all existing possibilities [1]. It is the same problem/concept/idea as the travelling salesman problem [1].	2	MAX 2
12 c	A tractable problem has a solution [1] in polynomial time or better [1]	2	
13	The second iterative statement loops through every cell in the grid [1]. For each cell: <ul style="list-style-type: none"> <li>• If it's the start of the line [1] and the gridSize is less than 10 [1] then write out the row number and a space [1]</li> <li>• Write out a vertical bar and the symbol in the current Cell [1]</li> <li>• If it's the end of the line [1] then it writes a vertical bar and then a horizontal line on the line below and</li> </ul>	6	

COPYRIGHT  
PROTECTED



INSPECTION COPY

## Task 17

(11 marks)

### Coding

- Creation of a WildCardCell class. [1 mark]
- Set the wild card symbol to 'W' and include suitable accessor methods to identify it as being a wild card. [1 mark]
- Required virtual methods for the Cell class to make the WildCardCell class operate correctly. [1 mark]
- Suitable variable to count the number of wild cards available. [1 mark]
- Advise the user of how many wild cards they have available to use. [1 mark]
- Test to confirm if a wild card needs to be placed when inputting a new symbol into the grid. [1 mark]
- Insert a wild card into a valid location in the grid and update the number of wild cards available. [1 mark]
- When a pattern helix is created, check if any of the cells within the pattern helix are a wild card. [1 mark]
- Suitable string handling to replace any wild card symbols in the pattern helix with the appropriate pattern symbol to test against. [1 mark]
- If a match is made when a wild card is present, return 15 points. [1 mark]

### Example Solution

Creation of new WildCard class:

```
//CHANGE
class WildCardCell extends Cell {
    public WildCardCell () {
        super();
        symbol = "W";
    }
    public boolean isWild() {
        return true;
    }
} //END CHANGE
```

Modification of the Cell class to make the WildCard class operate correctly:

COPYRIGHT  
PROTECTED



INSPECTION COPY

Modification to the checkForMatchWithPattern method to additionally allow wild cards to be placed:

```
public int checkForMatchWithPattern(int row, int column) {
    for (int startRow = row + 2; startRow >= row; startRow--) {
        for (int startColumn = column - 2; startColumn <= column; startColumn++) {
            try {
                // E
                Pattern p : allowedPatterns) {
                    String patternString = "";
                    List<Cell> patternCells = new ArrayList();
                    patternCells.add(getCell(startRow, startColumn));
                    patternCells.add(getCell(startRow, startColumn + 1));
                    patternCells.add(getCell(startRow, startColumn + 2));
                    patternCells.add(getCell(startRow - 1, startColumn + 2));
                    patternCells.add(getCell(startRow - 2, startColumn + 2));
                    patternCells.add(getCell(startRow - 2, startColumn + 1));
                    patternCells.add(getCell(startRow - 2, startColumn));
                    patternCells.add(getCell(startRow - 1, startColumn));
                    patternCells.add(getCell(startRow - 1, startColumn + 1));
                    boolean wildCardInPattern = false;
                    for (Cell c : patternCells) {
                        if(c.isWild()) {
                            wildCardInPattern = true;
                        }
                        patternString += c.getSymbol();
                    }
                }
                if(wildCardInPattern) {
                    patternString = patternString.replace("W", p.getPatternSymbol());
                }
                String currentSymbol = getCell(row, column).getSymbol();
                if(p.matchesPattern(patternString, currentSymbol)) {
                    getCell(startRow, startColumn).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow, startColumn + 1).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow, startColumn + 2).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow - 1, startColumn + 2).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow - 2, startColumn + 2).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow - 2, startColumn + 1).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow - 2, startColumn).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow - 1, startColumn).addToNotAllowedSymbols(currentSymbol);
                    getCell(startRow - 1, startColumn + 1).addToNotAllowedSymbols(currentSymbol);
                }
            } catch (Exception e) {
                // ignore
            }
        }
    }
}
```

COPYRIGHT  
PROTECTED



INSPECTION COPY

```

    }
    } catch (Exception e) {
    }
}
return 0;
}

```

Suitable variable to count the number of wild cards available:

```

class Puzzle {
    private int score;
    private int symbolsLeft;
    private int gridSize;
    private List<Cell> grid;
    private List<Pattern> allowedPatterns;
    private List<String> allowedSymbols;
    private static Random rng = new Random();
    //CHANGE
    private int wildCardsLeft = 2;
    //END CHANGE
}

```

Modification to the attemptPuzzle method:

```

public int attemptPuzzle() {
    boolean finished = false;
    while (!finished) {
        displayPuzzle();
        Console.WriteLine("Current score: " + score);
        //CHANGE
        Console.WriteLine("Wild cards will allow two symbols to be placed on top of each other and still be matched for each pattern.");
        Console.WriteLine("You currently have " + wildCardsLeft + " wild cards left.");
    }
}

```

**COPYRIGHT  
PROTECTED**



**INSPECTION COPY**

```

        continue;
    }
    } else {
        currentCell.changeSymbolInCell(symbol);
    }
    //END CHECK
    int amountToAddToScore = checkForMatchWithPattern(row, column);
    if (amountToAddToScore > 0) {
        score += amountToAddToScore;
    }
    }
    if (symbolsLeft != 0) {

```

### Testing

- Displaying an overlapping T pattern using a wild card. {1 mark}

You currently have 2 wild cards left.

Enter row number: 3

Enter column number: 4

Enter symbol: T

1	2	3	4	5
5	Q Q @ ~ ~			
4	Q Q T T T			
3	~ X Q W ~			
2	~ ~ X ~ ~			
1	Q X ~ X T			

1	2	3	4	5
5	Q Q @ ~ ~			
4	Q Q T T T			
3	~ X Q W ~			
2	~ ~ X ~ ~			
1	Q X ~ X T			

COPYRIGHT  
PROTECTED



INSPECTION COPY

## **Preview of Answers Ends Here**

---

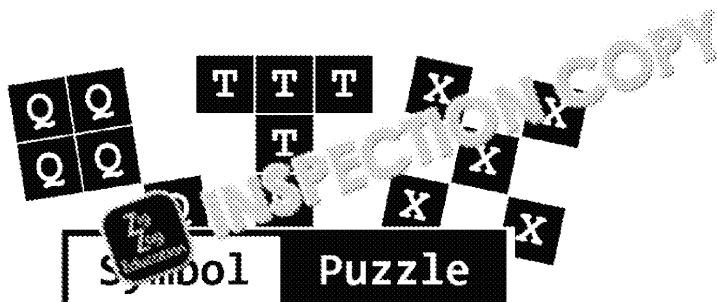
This is a limited inspection copy. Sample of answers ends here to stop students looking up answers to their assessments. See contents page for details of the rest of the resource.

Name

ZigZag Education supporting

## A Level AQA Computer Science Paper

Summer 2024



### Electronic Answer Document (EAD)

#### Instructions

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and any additional pages
- Answer **all** questions
- The marks available for each question are shown in brackets
- You will need:
  - ☐ access to a computer
  - ☐ access to a printer
  - ☐ access to appropriate software
  - ☐ electronic copies of the required skeleton code
  - ☐ EAD (Electronic Answer Document)

Total marks:



INSPECTION COPY

COPYRIGHT  
PROTECTED



## Exam-style Questions

Answer all questions. Remember to save this document

Q	Answer
1	(a)
	(b)
2	(a)
	(b)
3	(a)
	(b)
	(c)
4	
5	(a)
	(b)
	(c)
6	(a)
	(b)
7	(a)
	(b)
8	(a)
	(b)
9	(a)
	(b)
10	(a)
	(b)
11	(a)
	(b)
12	(a)
	(b)
	(c)
13	
14	(a)
	(b)
	(c)
15	

INSPECTION COPY

COPYRIGHT  
PROTECTED



## Exam-style Programming Task

Answer all questions. Remember to save this document

Q	Answer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

INSPECTION COPY

COPYRIGHT  
PROTECTED

