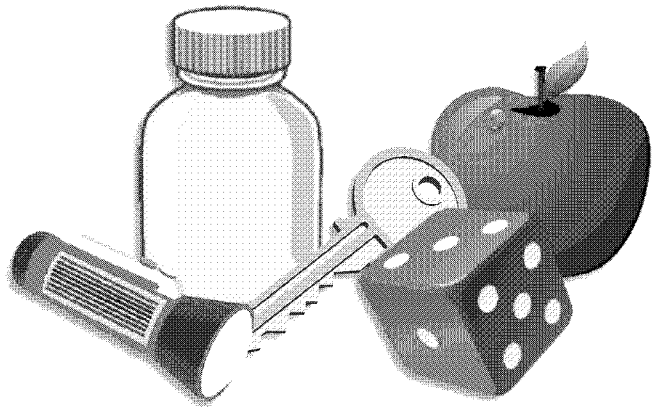
Zig Zag Education

**Computer Science**

A Level | AQA | 7517

2015 specification for the 2019 exam

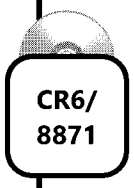


**JAVA**

# PAPER 1 EXAM RESOURCE PACK 2019

# AQA Text Adventures

## for A Level AQA Computer Science



**CR6/ 8871**

**POD 8871**

**zigzageducation.co.uk**

# Contents

# Teacher's Introduction

This resource pack is designed to help you support your students taking the **A Level Computer Science Paper 1** examination. It is based on the **'AQA Text Adventures'** preliminary material (Java) – for examination June 2019.

On the CD, you will find the following files.

| | | |
|---|---|---|
| 🗀 | `AQATextAdventures` | for student use – this folder contains all of the content, accessible via a HTML interface |
| 🗀 | `editable` | for teacher use – this folder contains ALL of the documents in editable (docx/pptx) formats |
| 🗎 | `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:** Copy the entire `AQATextAdventures` 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 in the 'Answers & Solutions' HTML page (`answers.html`) are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

| | |
|---|---|
| `Commentary.pdf` | `1158` |
| `Diagram1Complete.pdf` | `4773` |
| `Diagram2Complete.pdf` | `5382` |
| `Diagram3Complete.pdf` | `3091` |
| `QuestionsMarkScheme.pdf` | `7642` |
| `TaskMarkScheme.pdf` | `2966` |

Should you wish to give students access to ALL protected-PDFs, the master password for all files is:

`zz2ghc4`

The resource pack consists of the following:

① **Pre-release Commentary**, consisting of two parts:

- A general walkthrough of the skeleton program; a written description, flowchart and an animated PowerPoint giving a visual demonstration of the game. It is non-technical in the sense that it doesn't reference or explain any actual code elements – only how the program works when it is run.
- A detailed, technical overview of the skeleton program, describing how all Java subroutines, classes and variables work, including the relationship between them (also shown visually as a structure diagram).

**Note:** although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves. For this reason, this content has been placed on the 'Answers & Solutions' HTML page as a password-protected file, to allow you to control if/when students access it.

② **Structure Diagram Activities**

Three partially complete structure diagram activities for students to complete while getting to grips with the skeleton program. Any missing identifiers, data types, return values, directional arrows, etc. must be added to the diagram. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the skeleton program, like Section C in the exam. 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 version are available format. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test, like Section D in the exam. Solutions are provided on the *Answers & Solutions* page as a protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

**Free Updates**

Register your email address to receive any future free minor updates made to this resource or other Computing resources your school has purchased, and details of any promotions for your subject.

** resulting from minor specification changes, suggestions from teachers and peer reviews, or occasional errors reported by customers*

**zzed.uk/freeupdates**

An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

# AQA Text Adventu

## Introduction

*AQA Text Adventures* is a text adventure game, w[illegible] the player enters text comm locations, use various items and interact [illegible] ff[illegible]nt characters. The program d show the user the result of their [illegible] the current state of the game.

When the pr[illegible]m [illegible] there are no items, characters or places in the game. T game file (w[illegible] ill have the .gme file extension) they would like to use. If the pr a message w[illegible] be displayed to the player telling them that the game could not be terminate. If the file is found, then all of the items, characters and places in that fi and the game runs as below.

1. The game checks whether the player has moved to a new place. If they have, then the game displays the description of that place and the items in that place which the player can take.
2. The game asks the player to input an action.
3. The game identifies the command by splitting up the input into the command and the argument (e.g. if the player enters "get torch", the game will split this into the command "get" and the argument "torch").
4. If the command is recognised, the game will perf[illegible]e corresponding action and display the result.

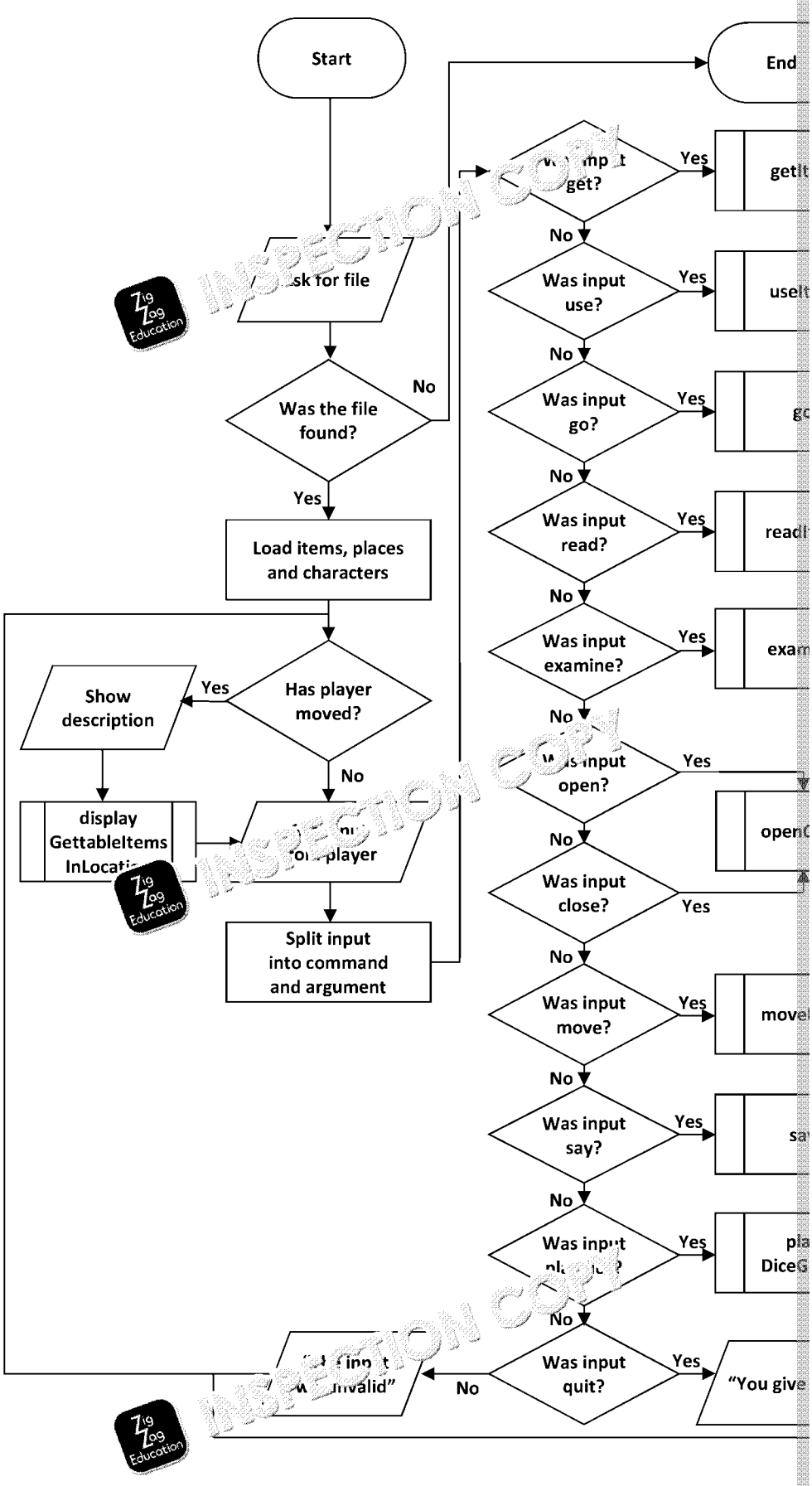
   The recognised commands a[illegible]llows:

| | |
|---|---|
| `get` | [illegible] the specified item to the player inventory |
| `use` | Uses an item in the current place or the player's inventory |
| `go` | Moves the player from their current place to the place in th |
| `read` | Displays the text of a readable item in the current place or t |
| `examine` | Provides a description of the specified item, place or charac player's inventory |
| `open` | Opens the specified item in the current place |
| `close` | Closes the specified item in the current place |
| `move` | Moves the specified item in the current place |
| `say` | Prints the specified input to the screen |
| `playdice` | Plays a dice game with the [illegible] fi[illegible] character in the curren |
| `quit` | Ends the game |

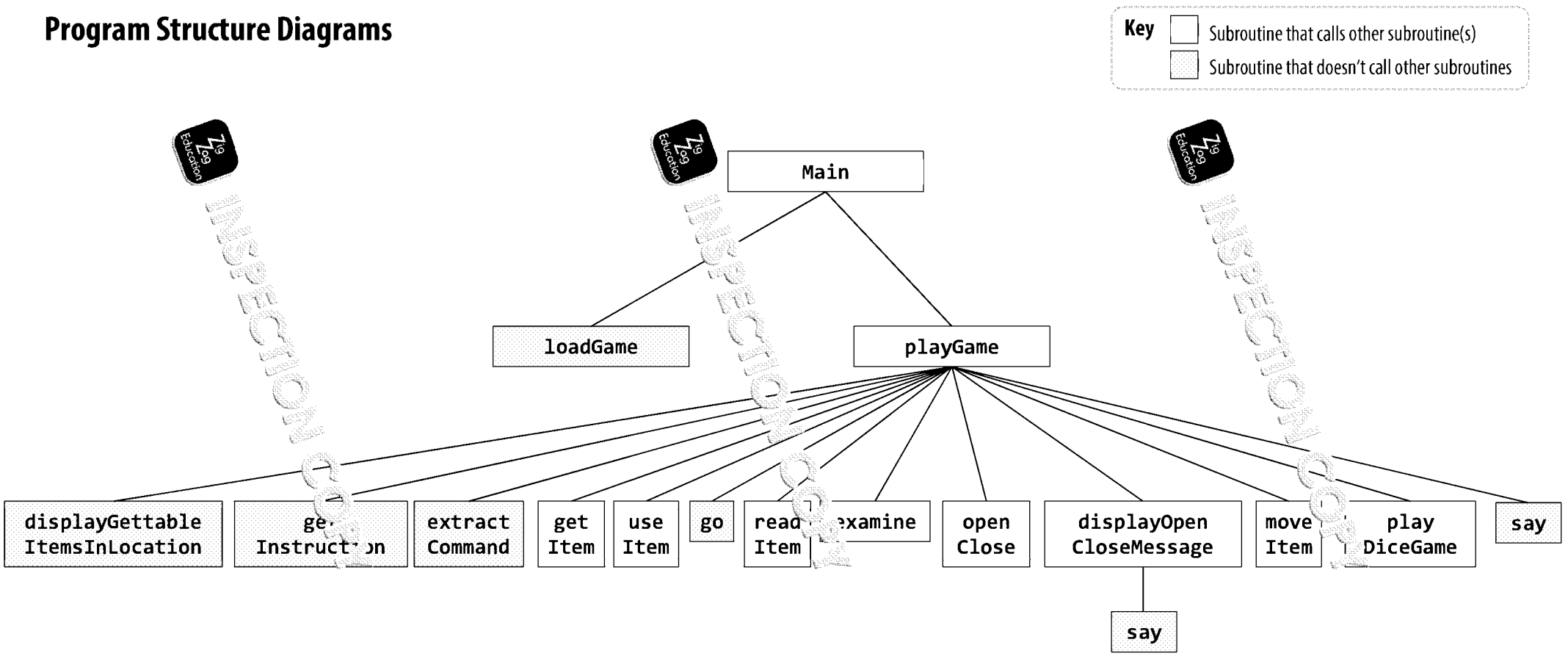
5. If the ga[illegible]do[illegible] r[illegible]cognise the command, a message is displayed to say th that a[illegible]

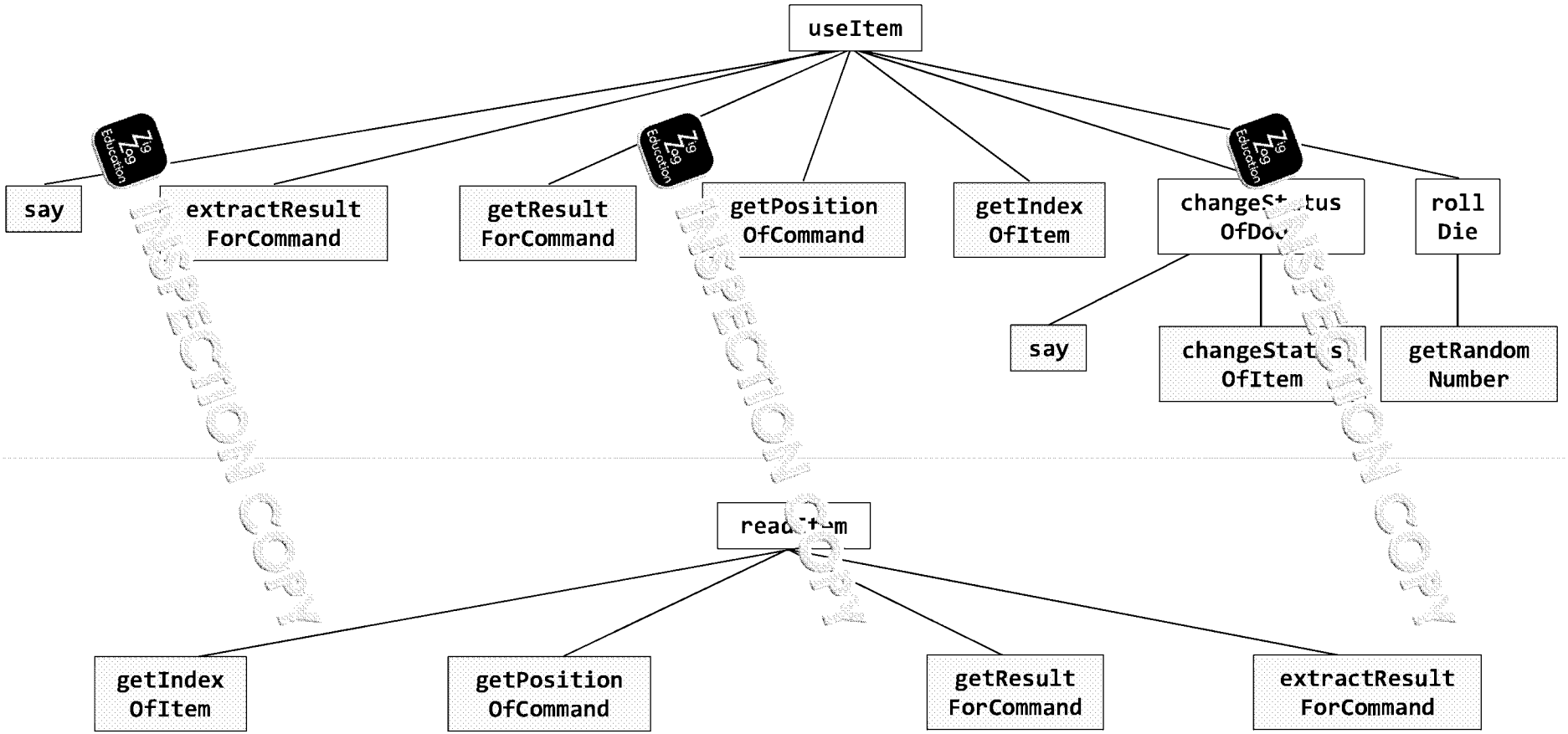
This loop continues until the player quits the game, or gets a particular item that t item is a flag in the provided .gme files). If the player quits the game, a message is given up, and if the player gets the flag (or other winning item), a message is displ

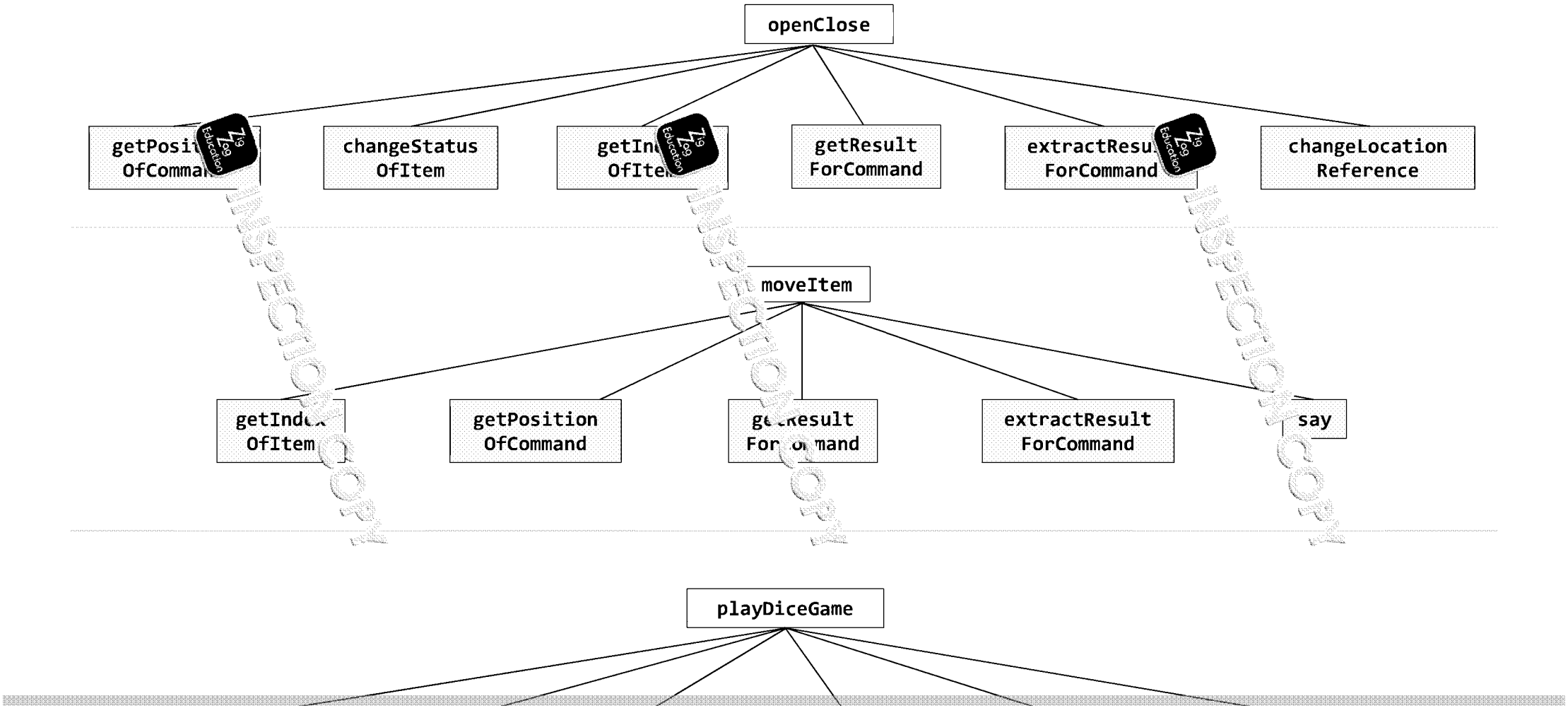
# Program Flow Chart



# Program Structure Diagrams







## Program Subroutines

The program's functions Ⓕ and procedures Ⓟ are described below.

| Subroutine | Data | | Description |
|---|---|---|---|
| changeLocaOfItem Ⓟ | Parameters:<br>Returns:<br>Called From:<br>Calls: | items([Item])<br>indexOfItem(int)<br>newLocation(int)<br>–<br>getItem<br>takeItemFromOtherCharacter<br>takeRandomItemFromPlayer<br>– | 1. Creates the thisItem object of type<br>2. Sets thisItem to be equal to the given item<br>3. Sets the location of thisItem to be equal to newLocation<br>4. Sets the given item to be equal to thisIte |
| changeLocationReference Ⓟ | Parameters:<br>Returns<br>Called Fr<br>Calls: | direction(String)<br>newLocationReference(int)<br>places([Place])<br>indexOfCurrentLocation(int)<br>opposite(Boolean)<br>–<br>openClose<br>– | 1. Creates the thisPlace object of type Place<br>2. Sets thisPlace to be equal to the location given by indexOfCurrentLocation<br>3. The values of direction and opposite are checked to find the direction reference that needs to be changed. If opposite is *false* then the direction reference of thisPlace for the direction direction is set to newLocationReference, but if opposite is *true*, then the direction reference of thisPlace for the opposite direction to direction is set to newLocationReference<br>4. The location given by indexOfCurrentLocation is set to be equal to thisPlace |
| changeStatusOfDoor Ⓟ | Parameters: | items([Item])<br>currentLocation(int)<br>indexOfItemToLockUnlock(int) | 1. Checks whether the given door or the other side of the given door is in currentLocation<br>2. If either is *true*, the status of the door is "locked", then the statuses of the |

| Subroutine | Data | Description |
|---|---|---|
| changeStatusOfItem Ⓟ | Parameters: items([Item])<br>indexOfItem(int)<br>newStatus(String)<br>Returns: –<br>Called From: changeStatusOfDoo<br>openClose<br>Calls: – | 1. Creates thisItem object of type Item<br>2. Sets thisItem to be equal to the given item<br>3. Sets the status of thisItem to be e o newStatus<br>4. Sets the given item to be equal to t em |
| checkIfDiceGam Possible Ⓕ | Parameters: items([Item])<br>characters([Character]<br>indexOfPlayerDie(int)<br>indexOfOtherCharacter( n )<br>indexOfOtherCharacterDie nt)<br>otherCharacterName(Strin<br>Returns: (int[])<br>Called From: playDiceGame<br>Calls: getIndexOfItem | 1. Loops through each item in items and if tem is in INVENTORY and has "die" in its name, gets the index of that die a l sets playerHasDie to *true*<br>2. Loops through each character in characte until a character is found in the same location as the player and has the na e otherCharacterName or all of the charac s have been checked<br>3. If a character is found that has the name othe haracterName and is in the same location as the player, sets player ameRoom to *true* and loops through each item in Items<br>4. If an item is held by the other character and has "d in its name, gets the index of that die and sets otherCharacterHasDi + *true*<br>5. Sets diceGamePossible to *1* if playerHasDie, play rsInSameRoom and otherCharacterHasDie are *true*<br>6. Returns a four-element integer array comprising a new integer (diceGamePossible, with 0 and 1 corresponding to *false* and *true* respectively) followed by indexOfPlayerDie, indexOfOtherCharacter and indexOfOtherCharacterDie |
| displayContents | Parameters: items([Item]) | 1. Displays the message "It contains: " |

| Subroutine | Data | Description |
|---|---|---|
| displayDoorStatus Ⓟ | Parameters: status(String)<br>Returns: –<br>Called From: examine<br>Calls: – | 1. If status is "open", display the message "The door is open."<br>2. Otherwise, display the message "The door is closed." |
| displayGettableItemsInLocation Ⓟ | Parameters: items([Item])<br>currentLocation(int)<br>Returns: –<br>Called From: playGame<br>Calls: – | 1. Loops through each item in items and checks whether the item is in currentLocation and whether its status contains "gettable"<br>2. If both of these conditions are *true*, appends the name of the item to a string starting "On the floor there is:"<br>3. If the name of another item has already been added to the string, prints a "," before the name of the next item<br>4. If the name of any item was appended to the string, a "." is appended to the end |
| displayInventory Ⓟ | Parameters: items([Item])<br>Returns: –<br>Called From: examine<br>Calls: – | 1. Prints empty line<br>2. Prints "You are currently carrying the following items:"<br>3. Loops through every item in items and if the location of that item is INVENTORY, prints the name of that item |
| displayOpenCloseMessage Ⓟ | Parameters: resultOfOpenClose(int)<br>openCommand(Boolean)<br>Returns: –<br>Called From: playGame<br>Calls: say | 1. If resultOfOpenClose is greater than *0*, then checks openCommand<br>2. If openCommand is *true*, then displays a message to say that the door has been opened, otherwise displays a message to say that the door has been closed<br>3. If resultOfOpenClose is *-3*, displays a message to say that the door is locked<br>4. If resultOfOpenClose is *-2*, displays a message to say that the |

| Subroutine | Data | Description |
|---|---|---|
| examine Ⓟ | Parameters: items([Item])<br>characters([Character])<br>itemToExamine(String)<br>currentLocation(…<br>Returns: –<br>Called From: playGame<br>Calls: displayInventory<br>getIndexOfItem<br>displayDoorStatus<br>displayContentsOfcontainerItem | 1. Creates the variable count and sets it to *0*<br>2. If itemToExamine is "inventory", displays inventory<br>3. Otherwise, gets the index of the item<br>4. If the index of the item can be found and the item is in the player's inventory or current location, displays the description of the item<br>5. Once the item description has been displayed, if the item contains "door", the subroutine displays the status of the door, otherwise if the status of the item contains "container", the subroutine returns to playGame after displaying the contents of the container<br>6. Loops through each character in characters and if the character's name is equal to itemToExamine, displays the description of that character and returns to playGame<br>7. Displays the message "You cannot find " + itemToExamine + " to look at." |
| extractCommand Ⓕ | Parameters: instruction(String)<br>Returns: command(String)<br>instruction(String)<br>Called From: playGame<br>Calls: – | 1. If instruction doesn't contain a space, then returns instruction as both the command and instruction variables<br>2. Otherwise the characters in instruction are added to the command variable until a space is found in instruction<br>3. instruction is set to all of the characters in the instruction that come after the first space<br>4. Returns command and instruction |
| extractResultFor Command Ⓕ | Parameters: subCommand(String)<br>subCommandParameter(String)<br>resultForCommand(String) | 1. Loads each character from resultForCommand into subCommand in sequence until a "," is read or the end of the string is reached<br>2. Any remaining characters, up until the next ";", "," or the end of the |

| Subroutine | Data | Description |
|---|---|---|
| `getIndexOfItem` Ⓕ | Parameters: `itemNameToGet(String)`<br>`itemIDToGet(int)`<br>`items([Item])`<br>Returns: `count(int)`<br>Called From: `getItem`<br>`useItem`<br>`readItem`<br>`examine`<br>`moveItem`<br>`openClose`<br>`checkIfDiceGamePossible`<br>Calls: – | 1. Creates the variable `count` and sets it to *0*, and creates the variable `stopLoop` and sets it to *false*<br>2. Loops through each item in `items` by increasing the `count` variable for as long as `count` is less than the length of `items` and `stopLoop` is *false*<br>3. For each item, if `itemNameToGet` is equal to the name of that item or `itemIDToGet` is equal to the ID of that item, sets `stopLoop` to *true*<br>4. If `itemNameToGet` is not equal to the name of any item in `items` and `itemIDToGet` is not equal to the ID of any number in Items, returns *-1*<br>5. Returns `count` |
| `getInstruction` Ⓕ | Parameters: –<br>Returns: `instruction(String)`<br>Called From: `playGame`<br>Calls: – | 1. Prints a line separator<br>2. Creates the variable `instruction`, which is set to the value of the user's input converted to lowercase<br>3. Returns `instruction` |

| Subroutine | Data | Description |
|---|---|---|
| `getItem` Ⓕ | Parameters: `items([Item])`<br>`itemToGet(String)`<br>`currentLocation(int)`<br>Returns: `stopGame(boolean)`<br>Called From: `playGame`<br>Calls: `say`<br>`extractResultForCommand`<br>`getResultForCommand`<br>`getPositionOfCommand`<br>`getIndexOfItem`<br>`changeLocationOfItem` | 1. Gets the index of the given item<br>2. If the index of the item couldn't be found, then displays a message to say that the item cannot be found<br>3. If the item is in `INVENTORY`, displays a message to say the item is already in the inventory<br>4. If the item doesn't have the command "get", displays a message to say that the player can't get item<br>5. If the item is stored in another item and the location of that item that contains it is not `currentLocation`, displays a message to say that the item cannot be found<br>6. If the item is not stored in another item and its location is not `currentLocation`, displays a message to say that the item cannot be found<br>7. If steps 2 to 7 above are all false, `canGet` is set to *true*<br>8. If `canGet` is *true*, gets the position of "get" in the item's list of commands and gets the result of using the "get" command on that item<br>9. If `subCommand` of the result is "say", then displays the `subCommandParameter` of the result<br>10. If `subCommand` of the result is "win", displays a message to say that the player has won the game and returns *true*<br>10. If the item's status contains "gettable", then displays the `subCommandParameter` of the result, changes the location of the item to `INVENTORY` and displays a message to say that the player now has that item<br>12. Returns *false* |
| `getPosition OfCommand` Ⓕ | Parameters: `commandList(String)`<br>`command(String)`<br>Returns: `position(int)` | 1. Creates the variable `position` and sets it to *0*<br>2. Loops through the characters in `commandList`, using each character as the starting position for a substring of `commandList` that is the length of `command` |

| Subroutine | Data | | Description |
|---|---|---|---|
| `getRandomNumber` Ⓕ | Parameters:<br><br>Returns:<br>Called From:<br><br><br>Calls: | `lowerLimitValue(int)`<br>`upperLimitValue(int)`<br>`(int)`<br>`rollDie`<br>`takeRandomIte` `Player`<br>– | 1. Returns random integer number between `lowerLimitValue` and `upperLimitValue` |
| `getResultForC` `nand` Ⓕ | Parameters:<br><br><br>Returns:<br>Called From:<br><br><br><br><br><br>Calls: | `results(String)`<br>`position(int)`<br>`resultForCommand(` `ing)`<br>`getItem`<br>`useItem`<br>`readItem`<br>`moveItem`<br>`openClose`<br>`playDiceGame`<br>– | 1. Creates the `resultForCommand` and set` to ""<br>2. Loops through each character in `results` `d as many ";"s have been read as the value of `position` or the end of the st` `g has been reached<br>3. Any remaining characters, up until the next ";" or the end of the string, are loaded into `resultForCommand`<br>4. `resultForCommand` is returned |
| `go` Ⓕ | Parameters:<br><br><br>R` `urns:<br>Called From:<br>Calls: | `you(Character)`<br>`direction(String)`<br>`currentPlace(Place)`<br>`moved(Boolean)`<br>`playGame`<br>– | 1. Creates the variable `moved` and sets it to *true*<br>2. `direction` is checked to see if it is a valid direction: "` `rth", "east", "south", "west", "up" or "down"<br>3. If a valid direction is given and `currentPlace` is adjacent to another place in that direction, the location of `you` is set to point to that other location<br>4. If a valid direction is given and `currentPlace` is not adjacent to another place in that direction, the value of `moved` is set to *false*<br>5. If an invalid direction is given, the value of `moved` is set to *false*<br>6. If `moved` is *false*, display the message "You are not able to go in that direction." |

| Subroutine | Data | Description |
|---|---|---|
| `loadGame` Ⓕ | Parameters: `filename(String)`<br>`characters([Character])`<br>`items([Item])`<br>`places([Place])`<br>Returns: `(boolean)`<br>Called From: `Main`<br>Calls: – | 1. If an exception is thrown, returns *false*, otherwise the subroutine runs as follows<br>2. Creates a `DataInputStream` object `reader` to connect to a binary file<br>3. Reads the number of characters from the file<br>4. Creates `tempCharacter` object of type `Character`, sets its properties to the values read in by `reader` and adds this character to `characters`. This is repeated for each character.<br>5. Reads the number of places from the file<br>6. Creates `tempPlace` object of type `Place`, sets its properties to the values read in by `reader` and adds this place to `places`. This is repeated for each place.<br>7. Reads the number of items from the file<br>8. Creates `tempItem` object of type `Item`, sets its properties to the values read in by `reader` and adds this item to `items`. This is repeated for each item.<br>9. Returns *true*, assuming no exceptions were thrown |
| `Main` Ⓟ | Parameters: –<br>Returns: –<br>Called From: `main method`<br>Calls: `loadGame`<br>`playGame` | 1. Creates `items`, `characters` and `places` variables as empty `ArrayLists`<br>2. Gets filename from user input (the program adds ".gme" to the user's input, so the file will not be recognised if the user enters the .gme to the end of the filename themselves)<br>3. Outputs an empty line<br>4. If call to `loadGame` returns *true*, calls `playGame`<br>5. If call to `loadGame` returns *false*, displays a message to say that the game could not be loaded and terminates the program after the user enters input |
| `moveItem` Ⓟ | Parameters: `items([Item])`<br>`itemToMove(String)`<br>`currentLocation(int)`<br>Returns: – | 1. Gets the index of the given item<br>2. If the index of the item is found and is in `currentLocation`, then checks the length of the item's list of commands |

| Subroutine | Data | | Description |
|---|---|---|---|
| `openClose` Ⓕ | Parameters: | `open(boolean)`<br>`items([Item])`<br>`places([Place])`<br>`itemToOpenClose(String)`<br>`currentLocation(int)` | 1. If `open` is *true*, creates the variable `command` and sets it to "open"<br>2. If `open` is *false*, creates the variable `command` and sets it to "close"<br>3. Loops through each item in `items` and if the name of the item is equal to `itemToOpenClose`, the item is in `currentLocation` and the string of that item's commands is at least 4 characters long, checks the status of the item |
| | Returns: | `directionChange(int)` | 4. If the status of the item is equal to `command`, returns *-2* |
| | Called From: | `playGame` | 5. If the status of the item is "locked", returns *-3* |
| | Calls: | `getPositionOfCommand`<br>`getResultForCommand`<br>`extractResultForCommand`<br>`changeStatusOfItem`<br>`getIndexOfItem`<br>`changeLocationReference` | 6. Otherwise, creates the variable `position` and sets it to be equal to the position of `command` in the door's list of commands, creates the variable `resultForCommand` and sets it to be equal to the result of using `command` on the door<br>7. Changes the status of the door<br>8. Sets `actionWorked` to *true*<br>9. Loops through each place to change the appropriate location references in `currentLocation` and the location that is linked to `currentLocation` by the door<br>10. Changes the status of the other side of the door<br>11. If `actionWorked` is *false*, returns *-1*<br>12. If `actionWorked` is *true*, returns `directionChange` cast as an integer |
| `playDiceGame` Ⓟ | Parameters: | `characters([Character])`<br>`items([Item])`<br>`otherCharacterName(String)` | 1. Calls `checkIfDiceGamePossible` to see if a dice game can be played<br>2. If a dice game can't be played, displays the message "You can't play a dice game."<br>3. If a dice game can be played, gets the position of "use" in the player's die's list of commands and gets the result of using the "use" command on that die to get the result of the dice roll<br>4. Displays a message to show what the player rolled |
| | Returns: | – | |
| | Called From: | `playGame` | |
| | Calls: | `checkIfDiceGamePossible` | |

| Subroutine | Data | Description |
|---|---|---|
| playGame Ⓟ | Parameters: characters([Character])<br>items([Item])<br>places([Place])<br>Returns: –<br>Called From: Main<br>Calls: displayGettableItemsInLocation<br>getInstruction<br>extractCommand<br>getItem<br>useItem<br>go<br>readItem<br>examine<br>openClose<br>displayOpenCloseMessage<br>moveItem<br>playDiceGame<br>say | 1. If the game has ended, waits for user input and then terminates, otherwise the subroutine runs the following loop<br>2. If the player has moved to a new location, prints two empty lines, the description of the location and the gettable items at the location<br>3. Gets input from the user, converts it to lowercase and splits it into a command and an instruction<br>4. If command is “get”, tries to get the given item<br>5. If command is “use”, tries to use the given item<br>6. If command is “go”, tries to move the character in the given direction<br>7. If command is “read”, tries to read the given item<br>8. If command is “examine”, tries to examine the given item, character or inventory<br>9. If command is “open”, tries to open the given item<br>10. If command is “close”, tries to close the given item<br>11. If command is “move”, tries to move the given item<br>12. If command is “say”, calls displays the instruction of the input<br>13. If command is “playdice”, tries to play a dice game with the given character<br>14. If command is “quit”, displays a message to say that the player has given up and ends the game<br>15. If command is anything else, displays a message to say that the command isn’t recognised |
| readItem Ⓟ | Parameters: items([Item])<br>itemToRead(String) | 1. Gets the index of the given item<br>2. If the index of the item couldn’t be found, then displays a message to say that the |

| Subroutine | Data | Description |
|---|---|---|
| `rollDie` Ⓕ | Parameters: `lower(String)` `upper(String)`<br>Returns: `(int)`<br>Called From: `useItem` `playDiceGame`<br>Calls: `isNumeric` `getRandomNumber` | 1. Creates `lowerLimitValue` variable, and sets it to *0*<br>2. If the `lower` string is made up entirely of numeric digits, as determined by a call to `isNumeric`, sets `lowerLimitValue` to `lower` (cast as an integer)<br>3. Otherwise, sets `lowerLimitValue` to be equ[illegible]e `int` equivalent of user input until `lowerLimitValue` is an `int` from *1* to *6*<br>4. Creates `upperLimitValue` variable, and sets it t[illegible]<br>5. If the `upper` string is made up entirely of numeric d[illegible] as determined by a call to `isNumeric`, sets `upperLimitValue` to `upper` (cas[illegible]an integer)<br>6. Otherwise, sets `upperLimitValue` to be equal to th[illegible]nt equivalent of user input until `upperLimitValue` is an int from *1* to *6*<br>7. [illegible]eturns a random integer number between `lowerLimi[illegible]lue` and [illegible]perLimitValue |
| `say` Ⓟ | Parameters: `speech(String)`<br>Returns: –<br>Called From: `playGame` `getItem` `useItem` `moveItem`<br>Calls: – | 1. [illegible]nts empty line<br>2. Prints `speech`<br>3. Pr[illegible] empty line |
| `takeItemFrom OtherCharacter` Ⓟ | Parameters: `items([Item])` `otherCharacterID(int)`<br>Returns: –<br>Called From: `playDiceGame`<br>Calls: `changeLocationOfItem` | 1. Loops through all of the items in `items`, and checks whether each item is held by the given character<br>2. If the item is held by that character, the name of that item is added to a list<br>3. The message "Which item do you want to take? They have:" is displayed, followed by each item in the list, with each item separated by ", " and the list ending with ".", |

| Subroutine | Data | | Description |
|---|---|---|---|
| `takeRandomItemFromPlayer` Ⓟ | Parameters:<br><br>Returns:<br>Called From:<br>Calls: | `items([Item])`<br>`otherCharacterID(int)`<br>–<br>`playDiceGame`<br>`getRandomItem`<br>`changeLocationOfItem` | 1. Loops through each item in `items` and checks whether each item is in `INVENTORY`<br>2. If the item is in `INVENTORY`, the position of that item in `items` is added to a list<br>3. Gets a random number from *0* to the highest position in the list<br>4. Displays a message to say which item has been chosen (the item who's position in the list matches the random number)<br>5. Moves the item from `INVENTORY` to the other character |
| `useItem` Ⓟ | Parameters:<br><br><br><br>Returns:<br>Called From:<br>Calls: | `items([Item])`<br>`itemToUse(String)`<br>`currentLocation(int)`<br>`places([Place])`<br>–<br>`playGame`<br>`say`<br>`extractResultForCommand`<br>`getResultForCommand`<br>`getPositionOfCommand`<br>`getIndexOfItem`<br>`changeStatusOfDoor`<br>`rollDie` | 1. Gets the index of the given item<br>2. If the index of the item can be found, then checks whether the item is in `INVENTORY` or is in `currentLocation` and has the status "usable"<br>If either is *true*, gets the position of "use" in the item's list of commands and gets the result of using the "use" command on that item<br>3. If `subCommand` of the result is "say", then displays the `subCommandParameter` of the result<br>5. If `subCommand` of the result is "lockunlock", then changes the status of the relevant doors<br>6. If `subCommand` of the result is "roll", then displays a message to say what number was rolled<br>7. If the index of item couldn't be found, then displays a message to say that the item can't be used |

# Program Classes

The classes defined in the program, and their attributes, are described below.

| Class | Description |
|---|---|
| Place | Class that defines the properties of each of the game's locations |
| Character | Class that defines the properties of each of the game's characters |
| Item | Class that defines the properties of each of the game's items |

## Place Attributes

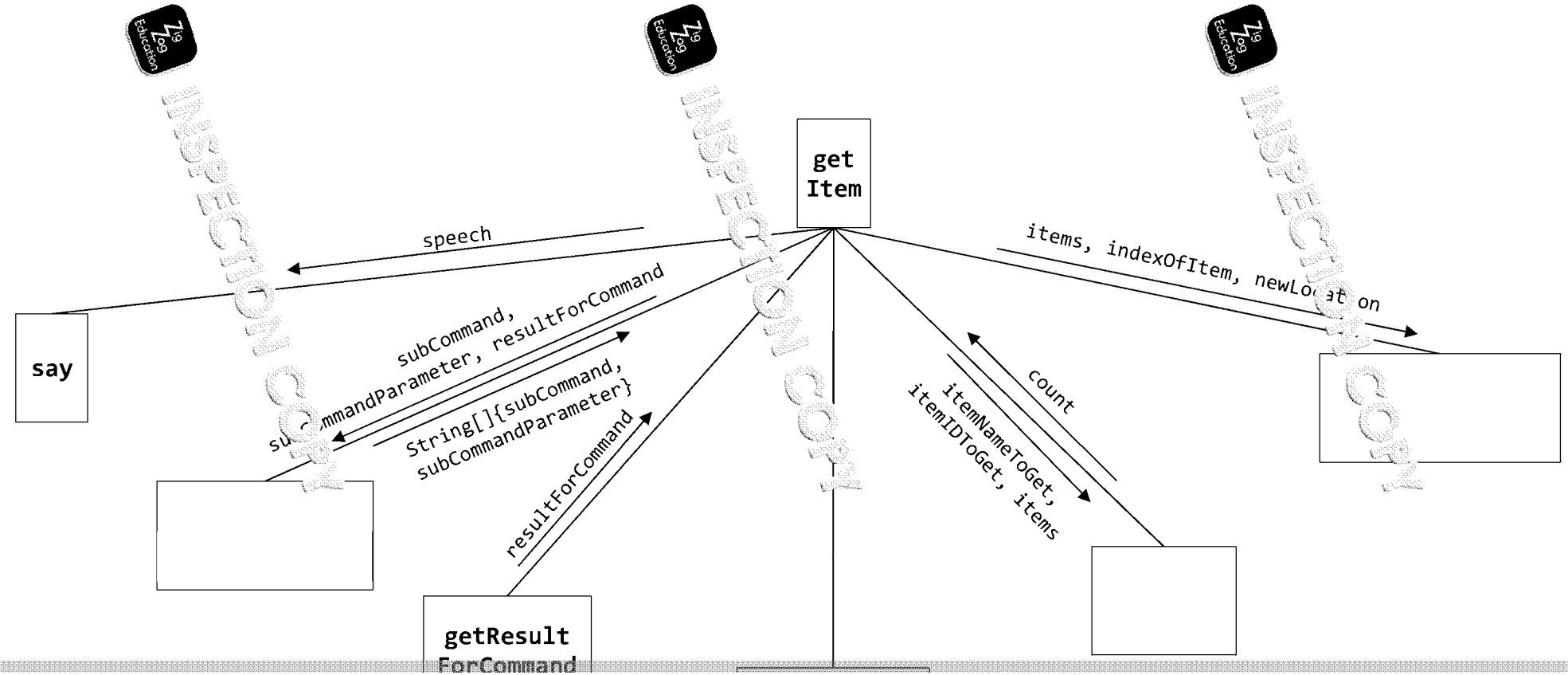
| Attribute | Type | Default Value | Description |
|---|---|---|---|
| description | String | "" | The text description of the place, which is shown to the player whenever they move to this place |
| id | Integer | 0 | The ID of this place, which allows it to be referenced by the program or other objects |
| north | Integer | 0 | The ID of the location to the north of this place; if this is 0, then there is no location to the north of the place |
| east | Integer | 0 | The ID of the location to the east of this place; if this is 0, then there is no location to the east of the place |
| south | Integer | 0 | The ID of the location to the south of this place; if this is 0, then there is no location to the south of the place |
| west | Integer | 0 | The ID of the location to the west of this place; if this is 0, then there is no location to the west of the place |
| up | Integer | 0 | The ID of the location above this place; if this is 0, then there is no location above the place |
| down | Integer | 0 | The ID of the location below this place; if this is 0, then there is no location below the place |

## Character Attributes

| Attribute | Type | Default Value | Description |
|---|---|---|---|
| name | String | "" | The name of the character, which is displayed to the player and used by the player to interact with this character |
| description | String | "" | The text description of the character, which is shown to the player whenever they examine this character |
| id | Integer | 0 | The ID of this character, which allows it to be referenced by the program or other objects |
| currentLocation | Integer | 0 | The ID of the place where the character is; if this is 0, then the character is not in a place |

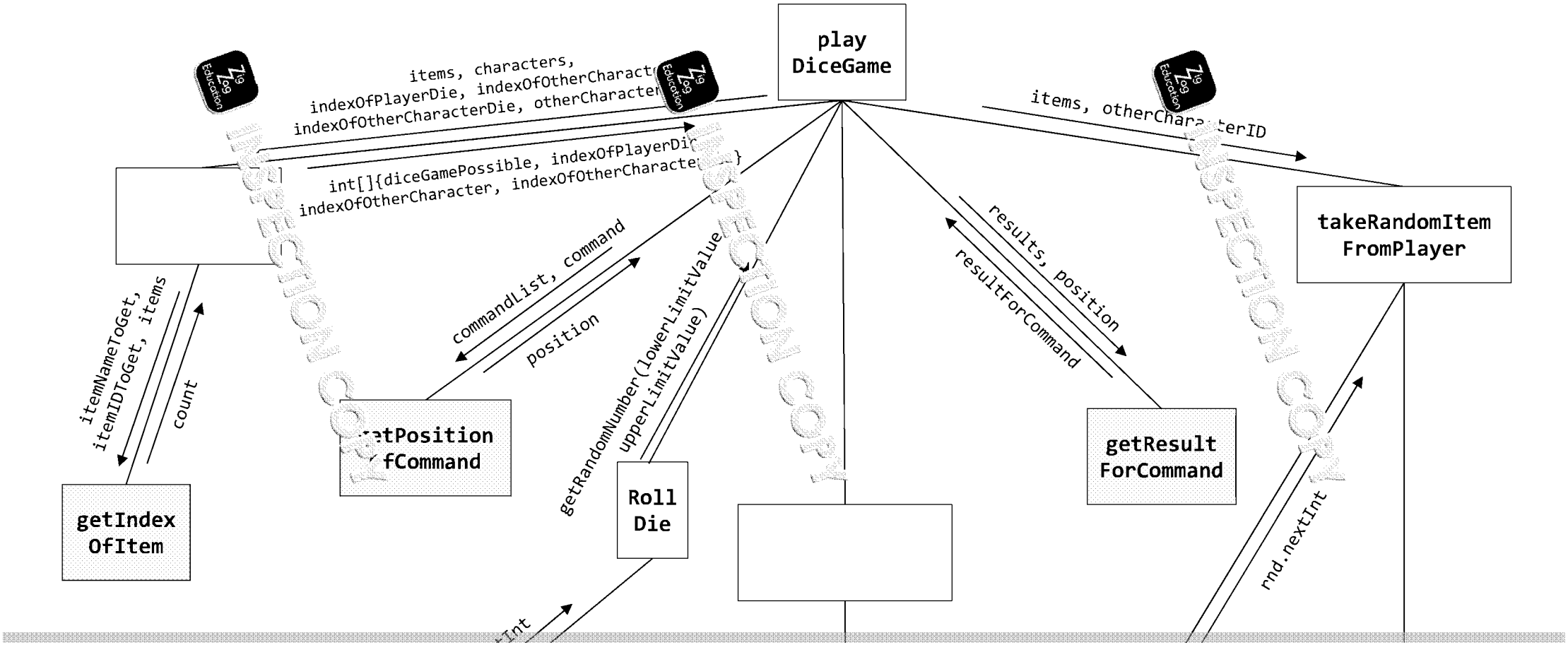
# Structure Diagram Activity 1 (getItem)





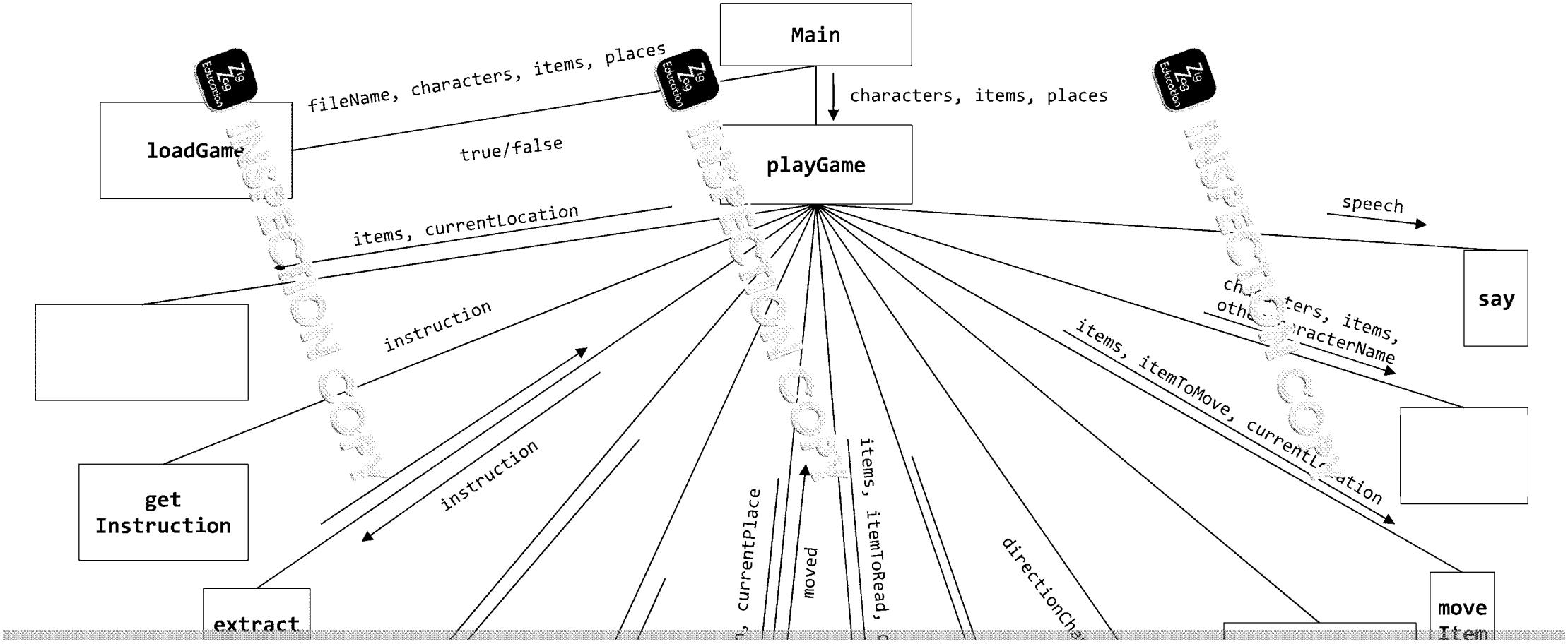
# Structure Diagram Activity 2 (playDiceGame)

# AQA Text Adventures



# Structure Diagram Activity 3 (Main)

# AQA Text Adventures



# AQA Text Adventures

## Written Questions (Java)

**These questions refer to the preliminary material and require you to load the skeleton program and make any additional programming.**

1. Ignoring the Console class, state the name of an identifier for:
   a) A local integer variable in the subroutine `loadGame` [1]
   b) A subroutine that returns an integer array [1]
   c) Four built-in functions used in the subroutine `extractCommand` [4]
   d) A class that has exactly four attributes [1]
   e) A constructor [1]
   f) A constant [1]
   g) An exception [1]

2. Explain the purpose of the following code in the `GetInstruction` subroutine:

```
Console.readLine().toLowerCase();
```

3. Explain the role of the variable `moved` in the subroutine `go`. [3]
4. Describe the purpose of the command block used throughout the subroutine `go`.
5. Using an example taken from anywhere in the skeleton program, describe what is meant by *string concatenation*.
6. Describe in detail the role of the subroutine `isNumeric`. [5]
7. Describe the role of the variable `tempCharacter` in the subroutine `loadGame`.
8. The subroutine `displayInventory` accepts an `ArrayList` as a parameter. Explain why it would be better to pass an `ArrayList` in `displayInventory` instead of an array. [4]
9. Explain why the `lowerLimitValue` in the subroutine `rollDie` is initialised to 1. [2]
10. Explain the purpose of each of the two while loops in the subroutine `extractCommand`. [4]
11. Describe the purpose of the `for` loop in the subroutine `displayInventory`. [3]
12. Describe the changes that would need to be made so that the game would output 'you are carrying nothing' if the player attempts to examine an empty inventory. [4]
13. Describe the changes that would need to be made to the to the subroutine `getInstruction` if a string of two or more words has been entered before `getInstruction` is called. [4]
14. Describe the effect of a file not being found during the subroutine `loadGame`. [2]

- END OF TEST -

# AQA Text Adventures

## Written Questions (Java)

**These questions refer to the preliminary material and require you to load the skeleton project. You do not require any additional programming.**

1. Ignoring the `Console` class, state the name of an identifier for:

   a) A local integer variable in the subroutine `loadGame` [1]

   .....

   b) A subroutine that returns an integer array [1]

   .....

   c) Four built-in functions used in the subroutine `extractCommand` [4]

   .....

   .....

   .....

   .....

   d) A class that has exactly four attributes [1]

   .....

   e) A constructor [1]

   .....

   f) A constant [1]

   .....

   g) An exception [1]

   .....

2. Explain the purpose of the following code in the `GetInstruction` subroutine:

```
Console.readLine().toLowerCase();
```

.....

.....

.....

.....

3. Explain the role of the variable `moved` in the subroutine `go`. [3]

.....

.....

.....

.....

.....

4. Describ... purpose of the command break used throughout the subroutine

.....

.....

.....

5. Using an example taken from anywhere in the skeleton program, describe wh... *string concatenation*. [2]

.....

.....

.....

6. Describe in detail the role of the ... routine `isNumeric`. [5]

.....

.....

.....

.....

.....

.....

7. Describe the role of the variable `tempCharacter` in the subroutine `loadG`

.....

.....

.....

.....

8. The subroutine `displayInventory` accepts an `ArrayList` as a parame
using an `ArrayList` in `displayInventory` instead of an array. [4]

.....

.....

.....

.....

.....

.....

9. Explain why the `lowerLimitValue` in the subroutine `rollDie` is initialise

.....

.....

.....

.....

.....

10. Explain the purpose of each of the two while loops in the subroutine `extrac`

While loop 1

.....

.....

.....

.....

While loop 2

.....

.....

.....

11. Describe the purpose of the `for` loop in the subroutine `displayInvento`

.....

.....

.....

12. Describe the changes that would need to be made so that the game would ou
nothing' if the player attempts to examine an empty inventory. [4]

.....

.....

.....

.....

.....

.....

13. Describe the changes that would need to be made to the to the subroutine g
that a string of two or more characters has been entered before `getInstr`

.....

.....

.....

.....

.....

14. Describe the effect of a file not being found d[illegible] subroutine `loadGam`

.....

.....

.....

- END OF TEST -

# AQA Text Adventures

## Programming Tasks (Java)

**The following require you to open the skeleton program and make modifications**

### Task 1

This question refers to `playGame` as well as a new subroutine `showHelp`.

Currently, users are not offered any help regarding valid commands. If a user ent
told 'Sorry, you don't know how to...'. Add a new subroutine called `showHelp`,
text on two lines:

```
You can enter one of the following commands:
go, get, use, examine, say, quit, read, move, open, clo
```

Modify the subroutine `playGame` so that if a user enters the command 'help', th

**Evidence you need to provide:**

- Your amended SOURCE CODE for `playGame`
- Your SOURCE CODE for the new `showHelp` subroutine
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flags1
  - Type 'help' at the first prom

### Task 2

This question refers to `examine` and `playGame`.

Currently, users are able to examine their inventory, as well as objects within a ro
Modify `examine` so that if the user has entered the text 'examine room', the des
be re-displayed. A call to `displayGettableItemsInLocation` should also

You should pass the `places ArrayList` from `playGame` to `examine`.

**Evidence you need to provide:**

- Your amended SOURCE CODE for `examine`
- Your amended SOURCE CODE for `playGame`
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flags1
  - Type 'examine room' at the first prompt

## Task 3

This question refers to `playGame`.

Currently, if the user types 'quit' at the prompt, the game ends without any furth
`playGame` so that an entry of 'quit' triggers an output of 'are you sure? (y/n)' an
either 'y' or 'yes' in any combination of upper or lowercase should terminate the
decide to give up, try again another time' still being displayed. Any other input sh
and the main prompt.

**Evidence you need to provide:**

- You[illegible] SOURCE CODE for `playGame`
- One [illegible]enshot showing the output that results from the following:
  - Run the program and load the file 'flag1'
  - Type 'quit' at the first prompt
  - Type 'no' in response to 'Are you sure'
  - Type 'quit' at the next prompt
  - Type 'yes' in response to 'Are you sure'
  - Press enter after resulting output

## Task 4

This question refers to `say` and `Main`.

Currently, output that is displayed as a result of a call to [illegible]y is displayed in a sing
displayed, multiple calls to say have to be made.

Create an additional implementation of `say` that receives a string array instead o
content of each element of the array on separate lines, maintaining the line break

In order to use your modified version of `say`, you will need to add code to the `Ma`
before the call to `playGame`, create an array consisting of the strings "WELCOM
"AQATEXTADVENTURES" and pass this array to `say`.

**Evidence you need to provide:**

- Your amended SOURCE CODE for `say`
- Your amended SOURCE CODE for `Main`
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flag1'

## Task 5

This question refers to `displayInventory`.

Currently, when the inventory is examined items are displayed in the order in wh
binary file. Modify the `displayInventory` subroutine, so that it uses a bubbl
ascending alphabetical order based on their `name` attribu[illegible]efore displaying the
sort algorithm yourself, and should not use any li[illegible]ing functions.

**Evidence you need to provide:**

- Your amend[illegible] CL CODE for `displayInventory`
- On[illegible]ns ot showing the output that results from the following:
  - [illegible]n the program and load the file 'flag1'
  - Type 'examine inventory' at the first prompt
  - Type 'get red die' at the second prompt
  - Type 'examine inventory' at the third prompt

## Task 6

This question refers to `getItem` and `takeItemFromOtherCharacter` as w

Currently, the player's inventory can contain as many items as can be found. Crea
`MAX_INVENTORY` and set it to 4.

When an attempt is made to get an item by a player wit[illegible]or more items alre
'Inventory full' should be displayed. The check fo[illegible]nventory should take pla
statements. If the inventory is full [illegible]es[illegible]e should be output from within `ge`
This check should also be a[illegible]to `takeItemFromOtherCharacter` so tha
after winning[illegible]ice [illegible]e message 'Inventory full' will be displayed if the pla

**Evidence you need to provide:**

- Your amended SOURCE CODE for `getItem`
- Your amended SOURCE CODE for `takeItemFromOtherCharacter`
- Your SOURCE CODE for the new `MAX_INVENTORY` constant
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flag1'
  - Type 'get torch' at the first subsequent prompt
  - Type 'get red die' at the second prompt
  - Type 'examine inventory' at the third prompt
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flag2'
  - Type 'playdice guard' at the first subs[illegible]t prompt
  - At each of the next two prompts, typ[illegible](repeat the previous ste
  - Type 'jar' at the next [illegible]m

## Task 7

This question refers to `playGame` as well as a new subroutine `dropItem`.

Currently, items can be picked up but not dropped. Create a new subroutine calle
parameters:

- An `ArrayList` called 'items', which will co... all ...ems in the game
- A string called 'itemToDrop', which ... be ...e name of the item leaving t
- An integer called 'curren... ...', which will be the ID of the player's loc

`dropItem` ...d ... that the item exists and that the player is carrying the it
the text 'Yo... ...ot carrying a _____' should be displayed, with the item name d
Otherwise, the item should be placed into the room with the text '_____ dropped
dropped.

You should modify `playGame` to call `dropItem` if a command of 'drop' is enter

**Evidence you need to provide:**

- Your amended SOURCE CODE for `playGame`
- Your SOURCE CODE for the new `dropItem` subroutine
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flag1'
  - Type 'drop flask' at the first prompt
  - Type 'drop flask' again at the second prompt
  - Type 'examine inventory' at the third ...

## Task 8

This questi... ...rs ... `Main`.

Currently, if the user enters the name of a non-existent file, the program ends wit
different file name. Additionally, the file extension `.gme` is always appended to t
included the extension themselves. This would result in a file not being found.

Modify the user prompt in `Main` to read 'Enter filename or enter Q to quit'. If th
should end. If the user enters a valid file name, the file should load as normal. Ot
continually re-display until the user has entered 'Q' or a file name that successfull
file name that contains the string `.gme` the program should not append `.gme` to th

You should only make changes to `Main`.

**Evidence you need to provide:**

- Your amended SOURCE CODE ... `M...`
- One screenshot sho... ... the ... put that results from the following:
  - Type '... ...' when prompted for a file name
  - ... t... next prompt, type 'flag1.gme'

## Task 9

This question relates to `playGame` as well as a new subroutine `displayChara`

In order for a developer to work effectively on existing code, it is helpful for them content of a data structure. Create a new subroutine called `displayCharacte` parameter in the form of the `characters` `ArrayList`.

When `displayCharacters` is called, it should displ[illegible] column headers in

- ID (followed by three spaces)
- NAME (followed by eight sp[illegible]
- DESCRIPTION (follo[illegible] spaces)
- CURRENTLC[illegible]

Beneath the [illegible]mns, all characters within the game should be displayed, with each attributes `id`, `name`, `description` and `currentLocation` should each line up

Modify `playGame` so that if 'debugchar' is entered during gameplay, a call to `dis`

**Evidence you need to provide:**

- Your amended SOURCE CODE for `playGame`
- Your SOURCE CODE for the new `displayCharacters` subroutine
- One screenshot showing the output that results from the following:
  - Type 'flag1' when prompted for a file name
  - At the next prompt, type 'debugchar'

## Task 10

This question relates to `playGame` as well as a [illegible]w [illegible]routine `fillVessel`. below the starting location that cont[illegible] a [illegible]l of water.

Create a new subroutine [illegible] `fillVessel` that requires the following parame

- An [illegible]yL[illegible]ed `characters`
- An [illegible]List called `items`
- An `ArrayList` called `places`
- A string called `vessel`

`fillVessel` should check whether the player and the barrel are in the same lo whose `status` attribute contains the text 'container' and does not contain the t be in the player's inventory. If any of these conditions are not met, the message displayed. Otherwise the vessel item's `name` attribute should have ' of water' app filled the ' should be displayed, with the vessel appended to the end. If the vessel the text 'of water', the text 'already full' should be displayed.

Add a call to `fillVessel` in `playGame` that will occur if the player enters the

**Evidence you need to provide:**

- Your amended SOURCE CODE for `playGame`
- Your SOURCE CODE for the [illegible] `essel` subroutine
- One screenshot show[illegible] the [illegible]put that results from the following:
  - Type '[illegible]' [illegible] prompted for a file name
  - [illegible] next prompt, type 'open trapdoor'
  - [illegible] the next prompt, type 'go down'
  - At the next prompt, type 'fill flask'
  - At the next prompt, type 'fill apple'
  - At the next prompt, type 'fill flask of water'
  - At the next prompt, type 'examine inventory'

## Task 11

This question relates to `go`, `playGame` and a new class, `History`.

Create a new class called `History`, with two attributes:

- An integer called `pointer`, initially set to -1
- A five-element integer array called `locations`

`History` will use a stack, to allow players to backtrack through previous locations. You should use an implementation of the `locations` array, which will be subject to two subroutines:

- `push` should require an integer as a parameter. If the stack is full, all existing values move down by 1 element (so the content of element 0 is lost), and the value of the parameter is added to the top. If the stack is not full, the pointer attribute should be incremented and the parameter added to the element indicated by the `pointer`. The `push` subroutine should not return anything.
- `pop` requires no parameters but returns an integer. If the stack is empty, it should return -1. Otherwise the value in the element indicated by `pointer` should be returned and `pointer` decremented.

In `playGame`, create an instance of the `History` class called 'past'. When the player moves, this instance should be passed to the `go` subroutine as an additional parameter.

In `go`, modify the parameters to accept the instance of the `History` class. As a player moves, add their location onto the stack before they move to a new one. If the instruction 'go back' is entered, move them to their previous location using a call to the `pop` subroutine in `History`. If the call returns -1, display the text 'you cannot go back'.

**Evidence you need to provide:**

- Your SOURCE CODE for the new `History` class
- Your amended SOURCE CODE for `playGame`
- Your amended SOURCE CODE for `go`
- One screenshot showing the output that results from the following:
  - Type 'flag2' when prompted for a file name
  - At the next prompt, type 'go east'
  - At the next prompt, type 'go back'
  - At the next prompt, type 'go back' a second time

## Task 12

This question relates to the `Character` class, the `playGame` subroutine, the `l` subroutine `eatItem`.

Modify the `Character` class so that each character in the game has a `health`

Modify the `loadGame` subroutine so that each cha... initial `health` value

Create a new subroutine called `eat...` which requires as parameters the `cha... items ArrayList` and ... called `itemToEat`. If the `itemToEat` string ... player's inv..., a... item's `status` attribute contains the text 'edible', in ... property by ... remove the eaten item from the `ArrayList`. If any of these ... 'you cannot eat that' should be displayed. If an item is successfully eaten, the ga... player's health.

Modify `playGame` so that the player's health is displayed immediately after the ... `displayGettableItemsInLocation`. Include a call in `playGame` to `eat` ... instruction beginning with 'eat'.

**Evidence you need to provide:**

- Your amended SOURCE CODE for the `Character` class
- Your amended SOURCE CODE for `loadGame`
- Your SOURCE CODE for the new `eatItem` subroutine
- Your amended SOURCE CODE for `playGame`
- One screenshot showing the output that re... from the following:
  - Type 'flag1' when prompted ... a fil... name
  - At the next prom... ... 'e... apple'
  - At the nex... ... type 'eat apple' a second time

## Task 13

This question relates to `takeItemFromOtherCharacter`.

Currently, if you win a dice game against an opponent and mis-type the item you ... told that you do not receive an item and you are given no opportunity to re-type. `takeItemFromOtherCharacter` so that if you request an item that they do ... have a _____ - try again' is displayed (with _____ replaced by the item the player ... asked again which item they want to take, with the available items re-listed. This ... player has entered an item that the other player possesses, at which point the ite... inventory as normal.

**Evidence you need to provide:**

- Your amended SOURCE C... ... the `takeItemFromOtherCharact...`
- One screensho... ... output that results from the following:
  1. ...p... '...' when prompted for a file name
  2. ... the next prompt, type 'playdice guard'
  3. At each of the next two prompts, type '6' (repeat the previous ste...
  4. When asked to choose an item, type 'box'
  5. At the next prompt, type 'jar'

## Task 14

This question relates to `checkIfDiceGamePossible`.

Currently, if a dice game is not possible, the program outputs the message 'you ca
whether the player has no die, the opponent has no die or the player and the opp
Modify `checkIfDiceGamePossible` so that the follo g messages are outp

1. If the player has no die, the outp ld read 'Player has no die'
2. If the player and the opp t ot in the same room, the output shou
   _____ being re name of the other character)
3. If t or has no die, the output should be '_____ has no die' (with
   of t r character)

In all cases, these messages should still be followed by the message 'You can't pla
more than one of these conditions are met, only one should be displayed, with co
which they are numbered above. So if the player has no die, the program will not
or that the opponent has a die. If the opponent is not present, no check will be m

No code should be modified besides the content of `checkIfDiceGamePossi`

**Evidence you need to provide:**

- Your amended SOURCE CODE for `checkIfDiceGamePossible`
- One screenshot showing the output that results from the following:
  - Type 'flag1' when prompted for a file name
  - At the next prompt, type 'playdice gu
  - At the next prompt, type 'get ed die
  - At the next prompt, t p ce guard'

## Task 15

This question relates to `displayInventory`.

Currently, when the user enters 'examine inventory', a list of the names of the ite
displayed. Modify `displayInventory` so that descriptions are displayed alon
should be displayed in round brackets. Additionally, the opening brackets of the
column, regardless of variations in an item's name.

**Evidence you need to provide:**

- Your amended SOURCE CODE for `displayInventory`
- One screenshot showing the output that results from the following:
  - Type 'flag1' when prompted for a file
  - At the next prompt, type 'exa ir in tory'

# AQA Text Adventures

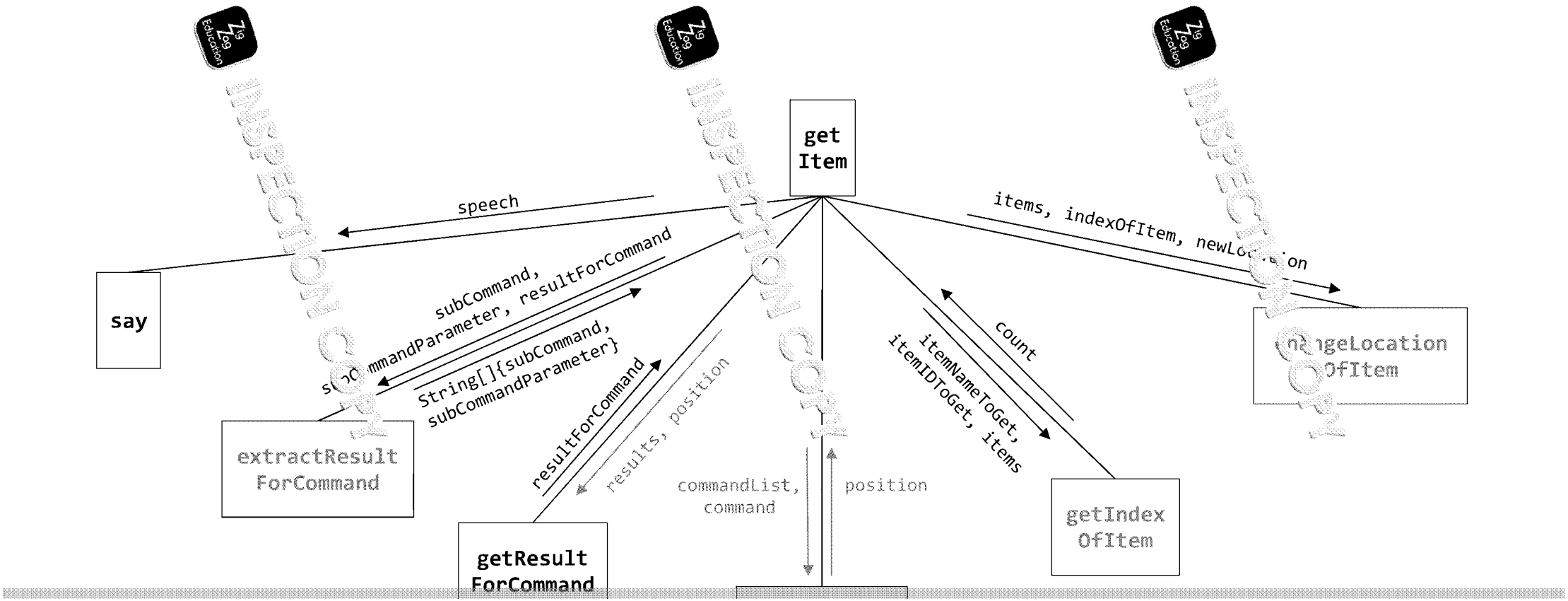
## Programming Tasks - Extension (Java)

**These challenges are presented without solutions, and offer opportunities to further explore and develop the program. They are listed here in approximate order of difficulty, beginning with the easiest.**

1. Add a 'restart' command to begin the current game again from the start, without having to load a new game by entering a file name
2. Modify `useItem` to incorporate a win condition that involves using an object other than the flag
3. Randomise the location of the guard
4. In the original program, the guard's blue die is not presented as an option for the player to take item to take after winning a dice game unless it is the guard's only remaining item. Modify `takeItemFromC` take the die even if the guard has other items. Modify `takeItemFromC` so that the player can only take the items that are listed after they have won a dice game.
5. Modify `go` so that the player will automatically try to unlock any locked doors that are in their way
6. Incorporate an additional command 'look', which could be used instead of 'go'. For example, 'look north' wouldn't have been blocked by a closed door
7. Attempting to pick up an item by using part of its name (such as 'die' in lieu of 'blue die') will successfully pick up the item
8. If the player is carrying two dice, both can be rolled and the total used against the guard
9. In the starting room for 'flag1.gme', the trapdoor is hidden by a rug but can be revealed by moving the rug. Edit the game so that the rug must be moved before the trapdoor can be opened
10. Edit `playDiceGame` so that characters cannot lose a die in a dice game (so that they always possess nothing but dice)
11. If you enter 'use truncheon' while in the same room as the guard, the guard is knocked out and items can be taken from him without the need to play dice
12. Include a 'save' command for players to save their progress, which could be loaded later
13. An option to create a game file from the console, with the new game file loaded

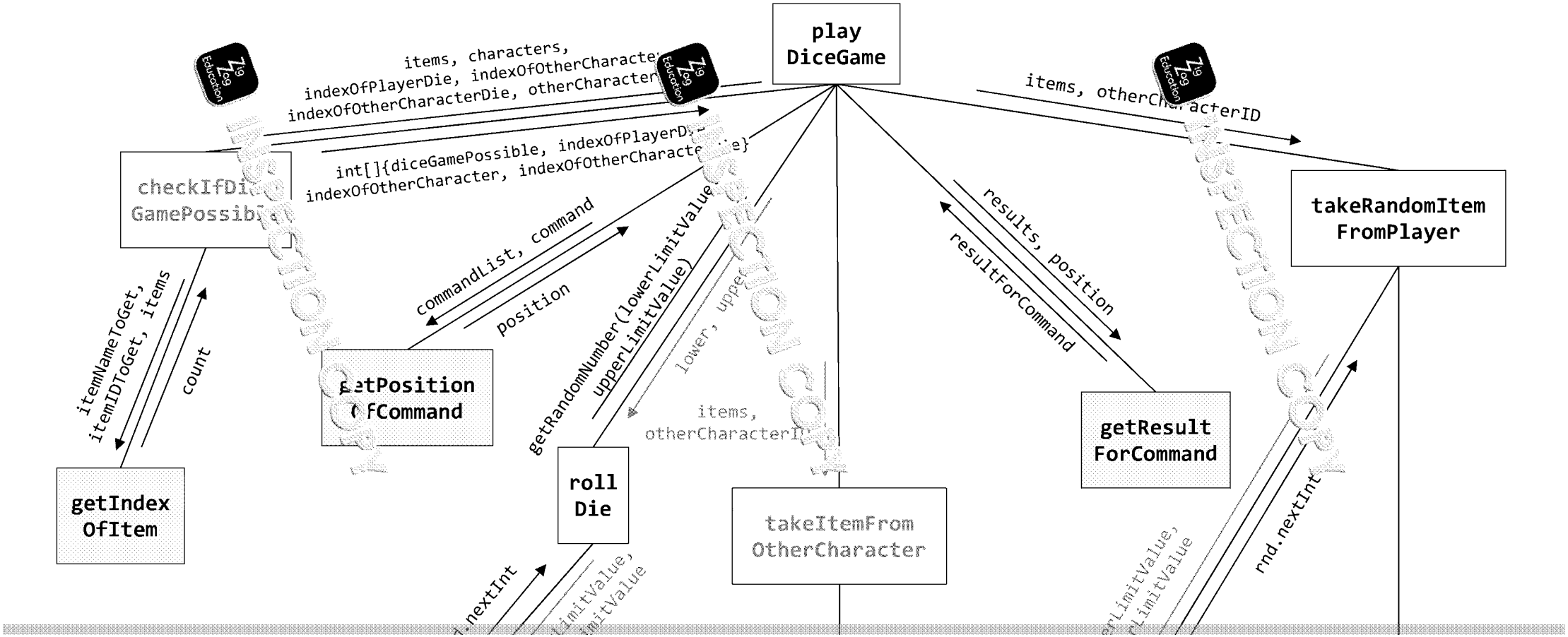
# Structure Diagram 1 (`getItem`) Solution





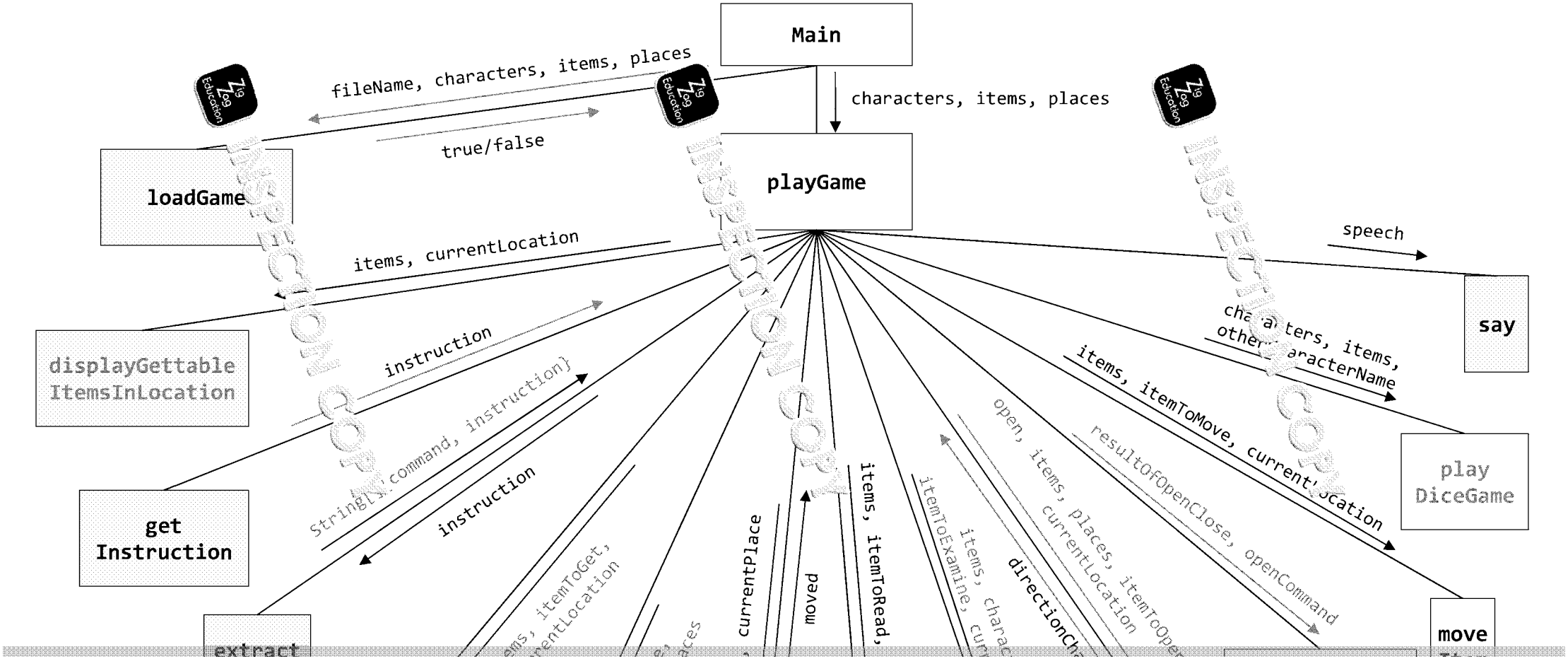
# Structure Diagram 2 (playDiceGame) Solution

# AQA Text Adventures



# Structure Diagram 3 (Main) Solution

# AQA Text Adventures



# AQA Text Adventures

## Written Questions (Java) - Solutions

| Q | Answer/Evidence |
|---|---|
| 1a | 1 mark:<br>• noOfCharacters<br>• noOfPlaces<br>• count<br>• [illegible] |
| 1b | checkPlaceGamePossible |
| 1c | 4 marks:<br>• contains<br>• length<br>• charAt<br>• substring |
| 1d | Character |
| 1e | 1 mark:<br>• Main<br>• Place<br>• Character<br>• Item |
| 1f | 1 mark:<br>• INVENTORY<br>• MINIMUM_ID_FOR_ITEM<br>• ID_DIFFERENCE_FOR_OBJECT_IN_TWO_LOCATIONS |
| 1g | e |
| 2 | 2 mar[illegible]<br>• [illegible]nvert the user input to lowercase<br>• Because subsequent comparisons will be made with lowercase strings / to [illegible] from being compared with otherwise identical lowercase strings |
| 3 | 3 marks:<br>• Stores whether the player has successfully moved<br>• If a player attempts an invalid move, it is set to *false*<br>• If it remains *false*, a message (you are not able to go in that direction) appe |
| 4 | 2 marks:<br>• Execution leaves that iteration of the switch structure / control goes to t<br>• ...to allow for a check as to whether the game should stop / whether stop |
| 5 | 2 marks:<br>• (Process of) connecting/joining two (or mor[illegible]g together<br>• Any instance of a + used between t[illegible]o [illegible]rin[illegible].g.<br>o listOfItems +=<br>o "You can[illegible] + itemToMove + "."<br><br>(Does n[illegible]ed [illegible] entire line) |
| 6 | 5 mar[illegible]<br>• Accepts a string as a parameter<br>• Tries / uses try block to convert the string to an integer<br>• Exception is thrown is conversion fails<br>• If an exception is thrown / if the catch block is triggered, *false* is returned<br>• Otherwise *true* is returned |

| Q | Answer/Guidance |
|---|---|
| 7 | 3 marks:<br>• Temporarily holds an instance of `Character`<br>• Attributes are set by reading data from the file<br>• Once attributes are set it is added to the `characters ArrayList` |
| 8 | Two marks for advantages, two for linking advantages to AQATextAdventure invent<br>• ArrayLists are dynamic / have no fixed max size<br>• ...so an item will always be able to be added to the inventory<br>• ArrayLists can be more easily added to in the middle of the data structure..<br>• ...so items could more easily be stored in a specified order (e.g. alphabetica |
| 9 | 3 marks:<br>• ...may not be set to a different value in the following `if` clause<br>• ...not set to a value, then the following `while` loop would cause an er `lowerLimitValue` was checked<br>• Initialising the value to 0 prevents this error from occurring |
| 10 | 3 marks (first `while` loop):<br>• The first `while` loop checks for there being no space in the first character<br>• If there is not a space, the first character is added to `command`<br>• The first character is (also) discarded from `instruction` / the `instruc` a substring from the second character onward<br>• Once the loop is complete, the `instruction` variable contains everythi onward and the `command` variable contains the contents of the initial stri before the first space<br>2 marks (second `while` loop):<br>• The second `while` loop checks for a space (in the first character of what<br>• If there is a space, it is discarded<br>• This loop removes leading spaces from `instruction` |
| 11 | 2 marks:<br>• Loops through each item in the `items ArrayList`<br>• Displays each item whose location is `INVENTORY` |
| 12 | 4 mar...<br>• ...ence made to changing method `displayInventory`<br>• Selection statement positioned before 'You are currently carrying' output<br>• Selection statement checks if any items have the location of `INVENTORY`<br>• If none of the items are in `INVENTORY`, display alternative message, othe message<br><br>Alternative implementations potentially worth full marks would include the selecti opposite order (i.e. checking that it is not empty) or pre-selection-statement instru items in the `ArrayList` then compare this with zero. |
| 13 | 3 marks:<br>• Assignment statement for instruction variable to be placed within a loop<br>• Check to be made within the loop that the length of the string is at least 2<br>• Loop to continue until that check returns *true* / length is at least 2 characte |
| 14 | 2 marks:<br>• Execution would move to the ...block / an exception would be throw<br>• The method `loadGame()` ...return a value of *false* |
| **TOTAL MARKS** | |

# AQA Text Adventures

## Programming Tasks (Java) - Suggested Solutions and Mark Scheme

Note that the following are recommended solutions and not an exhaustive list of a task. The marking guidance should be used as a guide only. Discretion should be alternative solutions are giv



### Task 1

**1 mark** correct declaration of `showHelp`, with `void` return type

**1 mark** valid call to `Console.writeLine` or `say` with text indicated in questi wording **must match**, including the line break, although any alternative m two lines is acceptable.

```
void showHelp() {
    Console.writeLine("You can enter one of the following c
    + "\ngo, get, use, examine, say, quit, read, move, open
}
```

**1 mark** additional `case` statement in an appropriate lo within `playGame`

```
    break;
case "help":
    showHelp()
    bre
case "g
```

**1 mark** screenshot showing output that matches text added to `showHelp`

```
Enter filename> flag1



You are in a room with blue walls.  There is a green door in the nor
On the floor there is: torch, red die.

> help
You can enter one of the following commands:
go, get, use, examine, say, quit, read, move , close and playdi

>
```

## Task 2

**1 mark** additional argument in call to `examine` from `playGame`

```
case "examine":
    examine(items, characters, places, instruction,
            characters.get(0).currentLocation);
    break;
```

**1 mark** additional parameter added to `examine` declaration

**1 mark** additional to selection structure (`else if`) to accommodate additional ec
code added in `if` clause, `else if` clause or `else` clause)

**1 mark** current location's description displayed in appropriate part of selection st

**1 mark** valid call to `displayGettableItemsInLocation` in appropriate pa

**1 mark** examining inventory and other objects remains unaffected

```
void examine(ArrayList<Item> items, ArrayList<Character> cha
        ArrayList<Place> places, String itemToExamine, int c
    int count = 0;
    if (itemToExamine.equals("inventory")) {
        displayInventory(items);
    } else if (itemToExamine.equals("room")) {
        Console.writeLine
            (places.get(characters.get(0).currentLocation -
        displayGettableItemsInLocation(items, characters.get
    } else {
        int index = getIndexOfItem(itemToExamine, -1,
```

**1 mark** screenshot showing repeated room description after user enters 'examine

```
Enter filename> flag1



You are in a room with blue walls.  There is a green door in the nort
the western wall and a cupboard door in the eastern wall. There is a
On the floor there is: torch, red die.

> examine room
You are in a room with blue walls.  There is a green door in the nort
the western wall and a cupboard door in the eastern wall. There is a
On the floor there is: torch, red die.

>
```

## Task 3

**1 mark** code within `case "quit"` to display prompt 'Are you sure? (y/n)' (**R**. al

**1 mark** user input is stored in a string variable

**1 mark** selection clause that catches either 'y' or 'yes' (**I**. case)

**1 mark** selection clause catches both 'y' and 'yes' in any combination of cases, eit
lower or by checking for every possible combination (including yES, yEs, e

**1 mark** output and [illegible] assignment inside the selection structure and break

```
case "qu
    say("Are you sure? (y/n)");
    String response = Console.readLine();
    if (response.toUpperCase().equals("Y") || response.toUpp
        say("You decide to give up, try again another time.
        stopGame = true;
    }
    break;
default:
```

**1 mark** output showing 'no' resulting in a prompt, followed by 'yes' resulting in th
terminating after the output

```
> quit

Are you sure? (y/n)

no

> quit

Are you su          (y/n)

yes

You decide to give up, try again another time.


BUILD SUCCESSFUL (total time: 19 seconds)
```

## Task 4

**1 mark** valid declaration of `say` with void return type and string array parameter

**1 mark** suitable loop to iterate through the array

**1 mark** output of each element within the loop

**1 mark** blank lines output both before and after the loop

```
void say(String[] speech) {
    Console.writeLine();
    for (String line : speech) {
        Console.writeLine(line);
    }
    Console.writeLine();
}
```

**1 mark** valid declaration and instantiation of a string array

**1 mark** passing array to say

```
if (loadGame(filename, characters, items, places)) {
    say(new String[]{"WELCOME", "TO", "AQATEXTADVENTURES"});
    playGame(characters, items, places);
} else {
```

**1 mark** screenshot showing each element of the array on a separate line before t

```
Enter filename> flag1


WELCOME
TO
AQATEXTADVENTURES



You are in a room with blue walls.  There
On the floor there is: torch, red die.

>
```

## Task 5

**1 mark** declaring temporary `ArrayList` of inventory items

**1 mark** loop to add all inventory items to the temporary `ArrayList`

```
ArrayList<Item> inventoryItems = new ArrayList<>();

for (Item thing : items) {
    if (thing.location == [illegible]) {
        inventoryIt[illegible]d(thing);
    }
}
```

**1 mark** loop to iterate through each pass of the sort operation (x in this example)

**1 mark** loop to iterate through each pair for individual comparisons (y in this exar

**1 mark** neither iteration would cause a comparison to be made beyond the `Arra`

**1 mark** selection structure that attempts to compare adjacent values, even if synt

**1 mark** selection structure accesses the name attribute of an item for the compar

**1 mark** selection structure is valid, comparing appropriate name values and addre

**1 mark** temporary variable used to store one of the items to be switched inside s

**1 mark** pair of items switched inside selection structure

```
Item temp;
for (int x = 0; x < inventoryItems.size() - 1; x++){
    for (int y = 0; y < inventoryItems.size() - 1; y++) {
        if (inventoryItems.get(y).name.compareToIgnoreCase
                (inventoryItems.get(y + 1).name) > 0){
            temp = inventoryItems.get(y);
                inventoryItems.set(y, inventoryItems.get(y+1
                inventoryItems.set(y+1, temp);
        }
    }
}
```

**1 mark** displaying items in the order they are found in the sorted array

```
Console.writeLine();
Console.writeLine("You are currently [illegible] the following

for (Item thing : inventoryItems) {
        Console.writeLine(thing.name);
}
Console.writeLine();
```

**1 mark** screenshot showing flag1 being loaded, 'examine inventory' being entere
jar) being displayed in alphabetical order. After 'get red die' and 'examine
inventory should again be displayed in alphabetical order (apple, flask, jar

```
You are in a room with blue walls.  There is a green door
On the floor there is: torch, red die.

> examine inventory

You are currently carrying the following items:
apple
flask
jar

> get red die
You have got that now.

> examine inventory

You are currently carrying the following items:
apple
flask
jar
red die


>
```

## Task 6

**1 mark** constant correctly declared and set to 4

```
static final int INVENTORY = 1001;
static final int MINIMUM_ID_FOR_ITEM = 2001;
static final int ID_DIFFERENCE_FOR_OBJECT_IN_TWO_LOCATIONS =
static final int MAX_INVENTORY = 4;
```

**1 mark** variable to count inventory it

**1 mark** iteration through [illegible] ArrayList

**1 mark** sel[illegible] equivalent to identify items in the inventory

**1 mark** incrementing variable inside selection statement

```
int inventorySize = 0;
for (Item thing : items){
    if (thing.location == INVENTORY){
        inventorySize++;
    }
}
```

**1 mark** selection statement to check that inventorySize >= constant (**R.** ==)

**1 mark** 'Inventory full' displayed only under correct conditions when the player t

**1 mark** remainder of selection structure would not be ev[illegible]ed if inventory is fu

```
if (inventorySize >= MAX_INVE[illegible]{
    Console.writeLine("[illegible] full");
} else if (index[illegible] -1) {
```

**1 mark** imp[illegible]ing inventorySize check in both getItem and takeIte

**1 mark** 'Inventory full' displayed only under correct conditions when the player trie

```
int inventorySize = 0;
for (Item thing : items){
    if (thing.location == INVENTORY){
        inventorySize++;
    }
}

if (listOfNamesOfItemsInInventory.contains(chosenItem) && in
        < MAX_INVENTORY) {
    Console.writeLine("You have that now.");
    int pos = listOfNamesOfItemsInInventory.indexOf(chosenIt
    changeLocationOfItem(it[illegible], listOfIndicesOfItemsInInvent
            INVENTORY);
} else if (inventorySize >= MAX_INVENTORY){
    Console.writeLine("Inventory full");
} else {
    Console.writeLine
            ("They don't have that item, so you don't take
}
```

**1 mark** first screenshot shows torch being picked up, red die not being picked up a

```
On the floor there is: torch, red die.

> get torch
You have got that now.

> get red die
Inventory full

> examine inventory

You are currentl[illegible] the following items:
flask
apple
jar
torch

>
```

**1 mark** second screenshot shows dice game being played and won, jar not being ta accordingly

```
Enter filename> flag2



You are in a guardroom.  There is a metal silver [illegible]r leading to a
chair behind it.  Sitting in the chair is [illegible].  There is a corri

> playdice guard
Enter minimum: 6
Enter maximum: 6
You roll[illegible].
They roll[illegible].
You win!
Which item do you want to take?  They have: jar, gold key, silver ke
jar
Inventory full

>
```

## Task 7

**1 mark** `dropItem` correctly declared with three parameters

**1 mark** call to `getIndexOfItem` (**A**. a correctly-implemented loop that has the

**1 mark** selection statement to check that item exists at all

**1 mark** selection statement to check whether the item being dropped is in the inv

**1 mark** if item is not in inventory, correct message displayed (**R**. if non-existent it same error messa

**1 mark** if it in inventory, location is changed to `currentLocation`

**1 mark** if item is in inventory, correct message displayed

```
void dropItem(ArrayList<Item> items, String itemToDrop, int
    int indexOfItem = getIndexOfItem(itemToDrop, -1, items);
    if (indexOfItem > -1) {
        if (!(items.get(indexOfItem).location == INVENTORY))
            Console.writeLine("You are not carrying a " + it
        } else {
            changeLocationOfItem(items, indexOfItem, current
            Console.writeLine(itemToDrop + " dropped");
        }
    } else {
        Console.writeLine("You are not carrying a " + itemTo
    }
}
```

**1 mark** selection structure in game incorporates 'drop' command

**1 mark** parameters correctly passed to `dropItem`

```
    break;
case "drop":
    dropItem(items, instruction, characters.get(0).currentL
    break;
```

**1 mark** screenshot shows 'flask dropped' and 'you are not carrying a flask' as well

```
You are in a room with blue walls.  There is a green door
On the floor there is: torch, red die.

> drop flask
flask dropped

> drop flask
You are not carrying a flask

> examine inventory

You are currently carrying the following items:
apple
jar


>
```

## Task 8

**1 mark** variable to track whether loop should repeat, or equivalent

**1 mark** loop repeats until a file name has successfully loaded

**1 mark** prompt amended to include quit option

**1 mark** selection (or inclusion in a loop) to address entry of 'Q'

**1 mark** execution always terminates when 'Q' is entered

**1 mark** selection statement to detect '.gme' within file name

**1 mark** '.gme' added only if it is not already within the string

**1 mark** successfully loading a file ensures loop would not run again

```
boolean isLoaded = false;
while (!isLoaded) {
    Console.write("Enter filename or enter Q to quit> ");
    filename = Console.readLine();
    if (filename.equals("Q")) {
        System.exit(0);
    }
    if (!filename.contains(".gme")) {
        filename = filename + ".gme";
    }
    Console.writeLine();
    if (loadGame(filename, rooms, items, places)) {
        isLoaded = true;
        playGame(rooms, items, places);
    } else {
        Console.writeLine("Unable to load game.");
    }
}
```

**1 mark** 'flag3' causes prompt to repeat (**I**. extra line breaks)

**1 mark** 'flag2.gme' causes game to begin

```
run:
Enter filename or enter Q to quit> flag3

Unable to load game.
Enter filename or enter Q to quit> flag2.gme



You are in a guardroom [illegible] a metal silver
ith a chair beh[illegible] Sitting in the chair is a

>
```

## Task 9

**1 mark** correct declaration of `displayCharacters`, including parameter

```
void displayCharacters(ArrayList<Character> characters) {
```

**1 mark** header row contains names of all fields as appropriate

**1 mark** spacing between headers is correct

```
String header = "";
header = header + "ID   ";
header = header + "NAME         ";
header = header + String.format("%-64s", "DESCRIPTION");
header = header + "CURRENTLOCATION";

Console.writeLine(header);
```

**1 mark** loop to iterate through each element of the `characters` ArrayList

**1 mark** each entry (even if inaccurate) will display on a separate line

**1 mark** all four attributes displayed on a line, with no additional text

**1 mark** spacing will be correct regardless of size of a field

```
for (Character c: characters) {
    String line = "";
    line = line + String.format("%-5s", c.id);
    line = line + String.format("%-12s", c.name);
    line = line + String.format("%-64s", c.description);
    line = line + c.currentLocation;

    Console.writeLine(line);
}
```

**1 mark** selection clause adapted to include checking for input of 'debugchar'

**1 mark** valid call to `displayCharacters` from `playGame`

```
    break;
case "debugchar":
    displayCharacters(characters);
    break;
```

**1 mark** two characters displayed, lined up correctly in columns in the correct order

```
> debugchar
ID   NAME         DESCRIPTION
1001 me           I think you look OK
1002 guard        The guard is about 180cm tall.  He is wearing a red

>
```

## Task 10

**1 mark** valid declaration

```
void fillVessel(ArrayList<Character> characters, ArrayList<
        ArrayList<Place> places, String vessel) {
```

**1 mark** variable, or equivalent, to store whether the vessel is a container

**1 mark** variable, or equivalent, to store whether the item is large *(equivalence in 'isContainer' is true only if the item is both a container and not large; correct to have a single variable, and all applicable marks can still be awarded)*

**1 mark** variable, or equivalent, to store whether the item is in the player's inventory

**1 mark** variable, or equivalent, to store whether the player is in the same location as the barrel

**1 mark** loop to examine items in turn (**A.** multiple loops, calls to `getIndexOfItem`)

**1 mark** selection statement to check whether the vessel is a container

**1 mark** selection statement to check whether the vessel is (not) large

**1 mark** selection statement to check whether the vessel is in the player's inventory

**1 mark** selection statement to check whether the player and the barrel are in the same location

**1 mark** all variables are set to the correct values under the circumstances

```
boolean isContainer = false;
boolean inInventory = false;
boolean byBarrel = false;

Item fillable = null;
for (Item i: items) {
    if(vessel.equals(i.name) && i.status.contains("container") &&
            !i.status.contains("large")) {
        isContainer = true;
        fillable = i;
    }
    if(vessel.equals(i.name) && i.location == INVENTORY) {
        inInventory = true;
    }
    if(i.name.equals("barrel") &&
            i.location == characters.get(0).currentLocation) {
        byBarrel = true;
    }
}
```

**1 mark** selection statement to check that all conditions are met

**1 mark** inside selection statement, append 'of water' to the vessel item's name (**A.** to identify the item or call to `getIndexOfItem`; this example code uses a loop above)

**1 mark** output, comprising the vessel appended to 'You have filled the'

**1 mark** 'You cannot do that' output if vessel has not been filled

**1 mark** check for presence of 'of water' within the name attribute

**1 mark** for ensuring that the program will not crash even if the fillable item is null

**1 mark** 'Already full' displayed under the correct circumstances

```
if(isContainer && inInventory && hyfr[] && !fillable.name.
    fillable.name = vessel + " of water";
    Console.writeLine("You have filled the " + vessel);
} else if (fillable != null && fillable.name.contains("of w
    Console.writeLine("Already full");
} else {
    Console.writeLine("You cannot do that");
}
```

**1 mark** selection structure in `playGame` adjusted to include 'fill' command

**1 mark** valid call to `fillVessel` at valid point in selection structure

```
    break;
case "fill":
    fillVessel(characters, items, places, instruction);
    break;
```

**1 mark** input of 'fill flask' displays 'You have filled the flask' and 'fill apple' displays

**1 mark** input of 'fill flask of water' displays 'Already full' and 'examine inventory'

```
> open trapdoor

You have opened a trapdoor

> go down

You are in a cellar. There is a ladder going up.

> fill flask
You have filled the flask

> fill apple
You cannot do that

> fill flask of water
Already full

> examine inventory

You are currently carrying the following items:
flask of water
apple
jar

>
```

## Task 11

**1 mark** `History` class correctly declared

**1 mark** integer variable `pointer` and integer array `locations` correctly decla

**1 mark** `pointer` and `locations` initialised to -1 and 5 elements respectively

```
class History {

    int pointer = -1;
    int[] locations;

    History() {
        locations = new int[5];
    }
```

**1 mark** `push` declared correctly, including integer parameter

**1 mark** selection statement to check whether `pointer` is < 4 or <= 3

**1 mark** if so, `pointer` is incremented…

**1 mark** …and the parameter is correctly inserted into the array

**1 mark** if not, contents of elements 4, 3, 2 and 1 are shifted down, with contents

**1 mark** …and the parameter is inserted into element 4 of the array

```
void push(int newLocation) {
    if (pointer < 4)
    {
        pointer++;
        locations[pointer] = newLocation;
    } else {
        for(int x = 0; x <= 3; x++) {
            locations[x] = locations[x + 1];
        }
        locations[4] = newLocation;
    }
}
```

**1 mark** `pop` correctly declared with no parameters and an integer return

**1 mark** check for a `pointer` value of -1

**1 mark** if the `pointer` is set to -1, then -1 should be the return value

**1 mark** `pointer` is decremented

**1 mark** value that was at the index previously indicated by `pointer` is returned

```
int pop() {
    if (pointer > -1) {
        pointer--;
        return locations[pointer + 1];
    } else {
        return -1;
    }
}
```

**1 mark** `past` declared as an instance of `History` in `playGame`

```
void playGame(ArrayList<Character> characters,
        ArrayList<Item> items, ArrayList<Place> places) {
    History past = new History();
    boolean stopGame = false, moved = true;
```

**1 mark** `past` instance passed to `go` when 'go' comm[illegible]ered by user (**A.** any

```
case "go":
    moved = go(charact[illegible], instruction,
            plac[illegible]aracters.get(0).currentLocation -
    bre[illegible]
```

**1 mark** declaration of `go` now includes a parameter of type `History` in same p[illegible]

```
boolean go(Character you, String direction,
        Place currentPlace, History past) {
```

**1 mark** attempt to call `push` within at least one `case` clause (even if unsuccessf[illegible]

**1 mark** `push` called correctly in all six cases, before player is moved to the new l[illegible] current location before movement, then variable pushed after movement should contain all six calls, each of which will be structured as below:

```
case "north":
    if (currentPlace.north == 0) {
        moved = false;
    } else {
        past.push(you.currentLocation);
        you.currentLocation = currentPlace.north;
    }
    break;
```

**1 mark** new case clause added for 'back'

**1 mark** call to `pop` (**R.** if any pass could call `pop` repeatedly)

**1 mark** selection statement to address return value of -1

**1 mark** message 'You cannot go back' displayed if -1 has been returned

**1 mark** location set to popped value only if it was not -1

**1 mark** `break` statement (otherwise default would still apply)

```
case "back":
    int newLocation = past.pop();
    if (newLocation == -1) {
        Console.writeLine("You cannot go back");
    } else {
        you.currentLocation = newLocation;
    }
    break;
```

**1 mark** having gone East, then back, the original room description should re-displ
second time should result in 'you cannot go back' being displayed.

```
Enter filename> flag2



You are in a guardroom.  There is a metal silver d   r leading to a je
, there is a gold metal door leading to a jail cell in the northern w
with a chair behind it.  Sitting in the c ai   s a guard.  There is a

> go east


You are i      ng, empty corridor.  There is a yellow door in the ea
western en     the corridor.

> go back


You are in a guardroom.  There is a metal silver door leading to a je
, there is a gold metal door leading to a jail cell in the northern w
with a chair behind it.  Sitting in the chair is a guard.  There is a

> go back
You cannot go back


You are in a guardroom.  There is a metal silver door leading to a je
, there is a gold metal door leading to a jail cell in the northern w
with a chair behind it.  Sitting in the chair is a guard.  There is a

>
```

## Task 12

**1 mark** addition of `health` attribute to `Character` class (**R.** if not integer)

```
class Character {
    String name, description;
    int id, currentLocation, health;
}
```

**1 mark** each character added to the `characters` ArrayList given `health`

```
tempCharacter.currentLocation = reader.readInt();
tempCharacter.health = 10;
characters.add(tempCharacter);
```

**1 mark** correct declaration for `eatItem`

```
void eatItem(ArrayList<Character> characters, ArrayList<Item
        String itemToEat) {
```

**1 mark** index of item acquired by iteration or call to `getIndexOfItem`

**1 mark** `item` index of -1 would not throw an exception

**1 mark** selection checks for 'edible' within the status of an item (**A.** if wrong item)

**1 mark** selection checks that an item is in the player's inventory (**A.** if wrong item)

**1 mark** five added to `health` only if item is edible and in the inventory (**R.** if wrong)

**1 mark** output indicates that item was eaten and new `health` value displayed

**1 mark** 'You cannot eat that' displayed in all other circumstances, including if an item (non-existent or misspelled item) was 'eaten'.

```
int itemIndex = getIndexOfItem(itemToEat, -1, items);
if (itemIndex > -1) {
    if (items.get(itemIndex).status.contains("edible") &&
        items.get(itemIndex).location == INVENTORY) {
        characters.get(0).health += 5;
        items.remove(itemIndex);
        Console.writeLine("You have eaten the " + itemToEat)
        Console.writeLine("Your health is " + characters.get
    } else {
        Console.writeLine("You cannot eat that");
    }
} else {
    Console.writeLine("You cannot eat that");
}
```

**1 mark** `health` displayed after call to `displayGettableItemsInLocati`

```
displayGettableItemsInLocation(items, characters.get(0).cur
Console.writeLine("Your health is " + characters.get(0).hea
moved = false;
```

**1 mark** valid `case` clause added within `playGame`

```
    break;
case "eat":
    eatItem(charact      s, instruction);
    break;
```

**1 mark** `hea` initially output as 10; first response to 'eat apple' is that the app
health is now 15; second call to 'eat apple' results in 'You cannot eat that'

```
You are in a room with blue walls.  There is
On the floor there is: torch, red die.
Your health is 10

> eat apple
You have eaten the apple
Your health is 15

> eat apple
You cannot eat that

>
```

## Task 13

**1 mark** variable to track whether item has been taken (**A**. break command if item

**1 mark** loop to begin before `count = 1;`

**1 mark** an item being successfully taken will terminate the loop

**1 mark** an item not being taken will cause the loop to repeat

**1 mark** output 'They don't ha... ...ject including concatenation with `chose`

```
boolean itemTaken = false;
do {
    count = 1;
    Console.write("Which item do you want to take?  They ha
    Console.write(listOfNamesOfItemsInInventory.get(0));
    while (count < listOfNamesOfItemsInInventory.size() - 1)
        Console.write(", " + listOfNamesOfItemsInInventory.
        count++;
    }
    Console.writeLine(".");
    String chosenItem = Console.readLine();
    if (listOfNamesOfItemsInInventory.contains(chosenItem))
        Console.writeLine("You have that now.");
        int pos = listOfNamesOfItemsInInventory.indexOf(cho
        changeLocationOfItem(items,
                listOfIndicesOfItemsIn...tory.get(pos), I
        itemTaken = true;
    } else {
        Console.wri... "They don't have a " +
            ...senItem + " - try again.");
    }
} while (itemTaken == false);
```

**1 mark** attempt to take a box answered with 'They don't have a box - try again'; a with 'You have that now'.

```
> playdice guard
Enter minimum: 6
Enter maximum: 6
You rolled a 6.
They rolled a 2.
You win!
Which item do you want to take?  They have: jar, gold key, silver k
box
They don't have a box - try again.
Which item do you want to take? ...ave: jar, gold key, silver k
jar
You have that now.

>
```

## Task 14

**1 mark** selection structure within `checkIfDiceGamePossible` after `playe` `playersInSameRoom` and `otherCharacterHasDie` have been as statement

**1 mark** 'Player has no die' displayed only if `playerHasDi` is *false*

**1 mark** '_____ is not here' displayed only if `ay sInSameRoom` is *false*, to ir

**1 mark** '_____ has no die' displayed only if `otherCharacterHasDie` is *false* ( previous mark denied due to failure in concatenation)

```
int diceGamePossible = 0;
if (playerHasDie && playersInSameRoom && otherCharacterHasDi
    diceGamePossible = 1;
}

if (!playerHasDie) {
    Console.writeLine("Player has no die");
} else if (!playersInSameRoom) {
    Console.writeLine(otherCharacterName + " is not here");
} else if (!otherCharacterHasDie) {
    Console.writeLine(otherCharacterName + " has no die");
}

return new int[]{diceGamePossible, indexOfPlayerDie,
    indexOfOtherCharacter, indexOfOtherCharacterDie};
```

**1 mark** 'Player has no die' on first attempt; 'guard is not here' on second attempt should appear both times (**R.** if any additional output)

```
Enter filename> flag1



You are in a room with blue walls.  There is a
On the floor there is: torch, red die.

> playdice guard
Player has no die
You can't play a dice game.

> get red die
You have got that now.

> playdice guard
guard is not here
You can't play a dice game.

>
```

## Task 15

**1 mark** item names are padded with space to make them the same width

**1 mark** descriptions are displayed on the same line as names (**R**. if multiple inven

**1 mark** descriptions, but not names, are displayed in round brackets

```
for (Item thing : items) {
    if (thing.location == [illegible]) {
        Console.writ[illegible]fin.format("%-10s", thing.name));
        Conso[illegible]("(");
        [illegible].write(thing.description);
        [illegible].write(")\n");
    }
}
```

**1 mark** descriptions in brackets in a straight-line column

```
Enter filename> flag1


You are in a room with blue walls.  There is a green door in the
On the floor there is: torch, red die.

> examine inventory

You are currently carrying the following [illegible]s:
flask       (It is an empty plastic [illegible].)
apple       (It is an apple. [illegible] tasty.)
jar         (It is an [illegible] jar.  The lid is missing.)

>
```

| Name | |
|---|---|

ZigZag Education supporting

# A Level AQA Computer Science Pap

## Summer 2019

# AQA Text Adventures

## 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:**

# Written Questions

Answer all questions.
Remember to save this document regularly.

| Q | | |
|---|---|---|
| 1 | (a) | |
| | (b) | |
| | (c) | |
| | (d) | |
| | (e) | |
| | (f) | |
| | (g) | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |
| 11 | | |
| 12 | | |
| 13 | | |
| 14 | | |

# Programming Tasks

Answer all questions.
Remember to save this document regularly.

| Q | Answer |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | |