Zig Zag Education

Computer Science

A Level | AQA | 7517

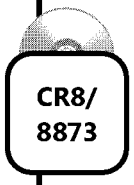
2015 specification for the 2019 exam

PYTHON3

PAPER 1 EXAM RESOURCE PACK 2019

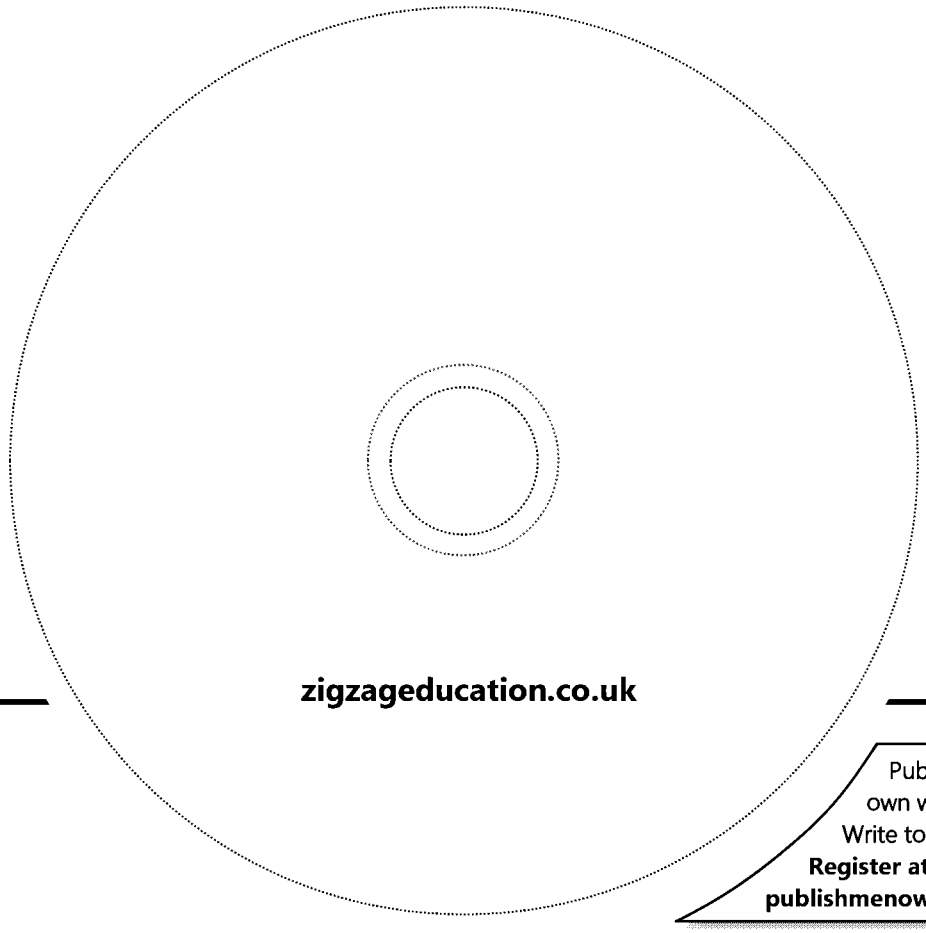
AQA Text Adventures

for A Level AQA Computer Science



CR8/ 8873

POD 8873



zigzageducation.co.uk

Publish your own work... Write to a brief... **Register at publishmenow.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 (Python3) – 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 Python 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.

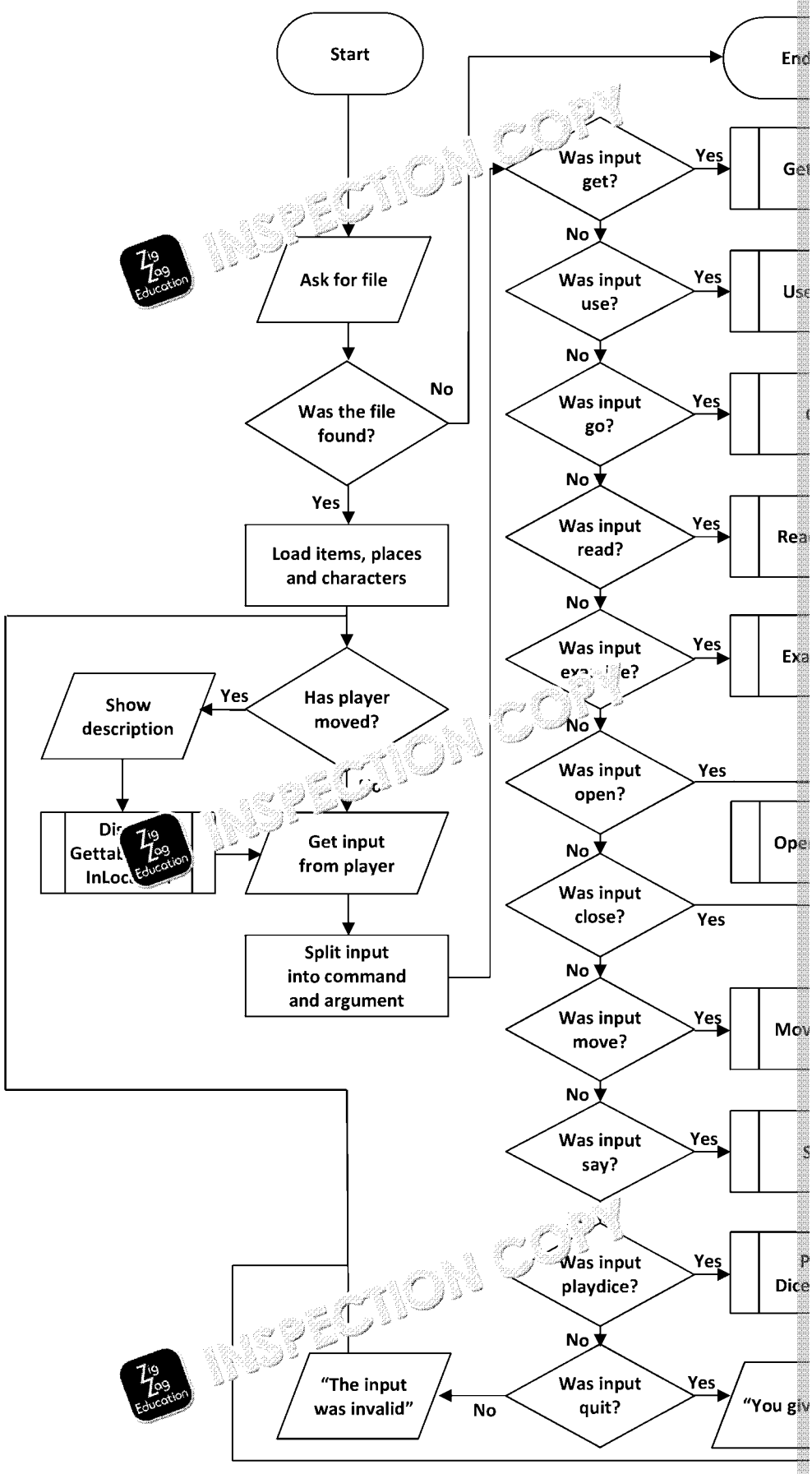
   The recognised commands a[illegible]llows:

`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

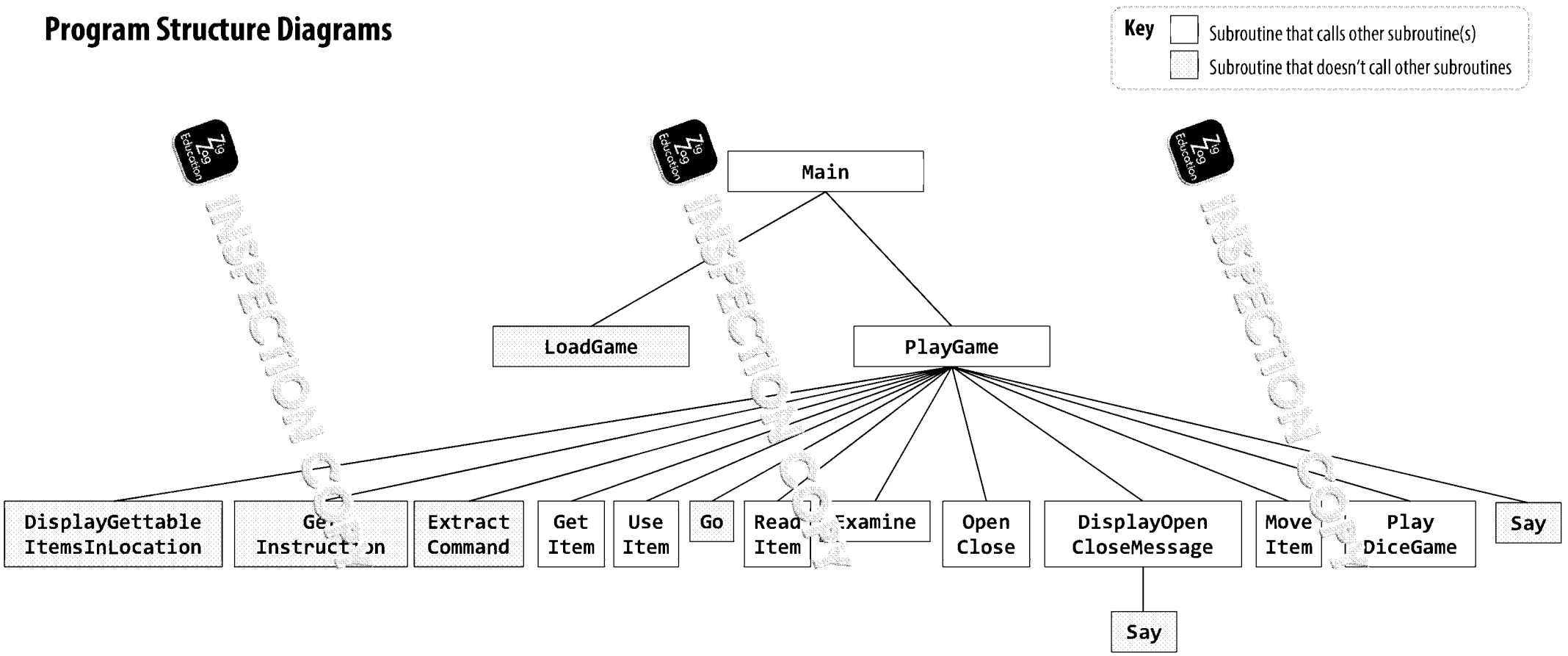
5. If the ga[illegible]do[illegible] [illegible]r[illegible]cognise the command, a message is displayed to say th that a[illegible]

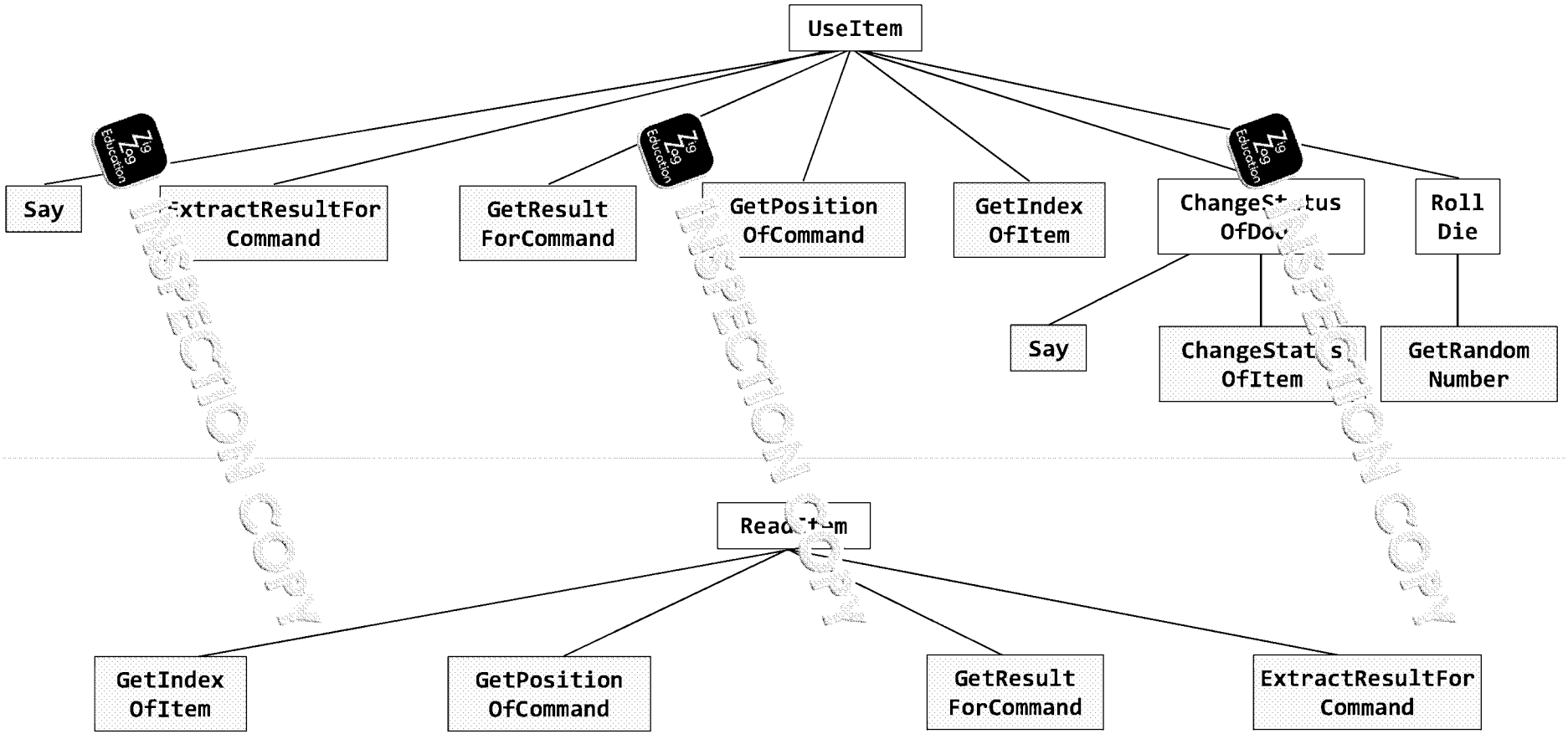
This loop continues until the player quits the game, or gets a particular item that t item is a flag in the provided .gme files). If the player quits the game, a message is given up, and if the player gets the flag (or other winning item), a message is displ

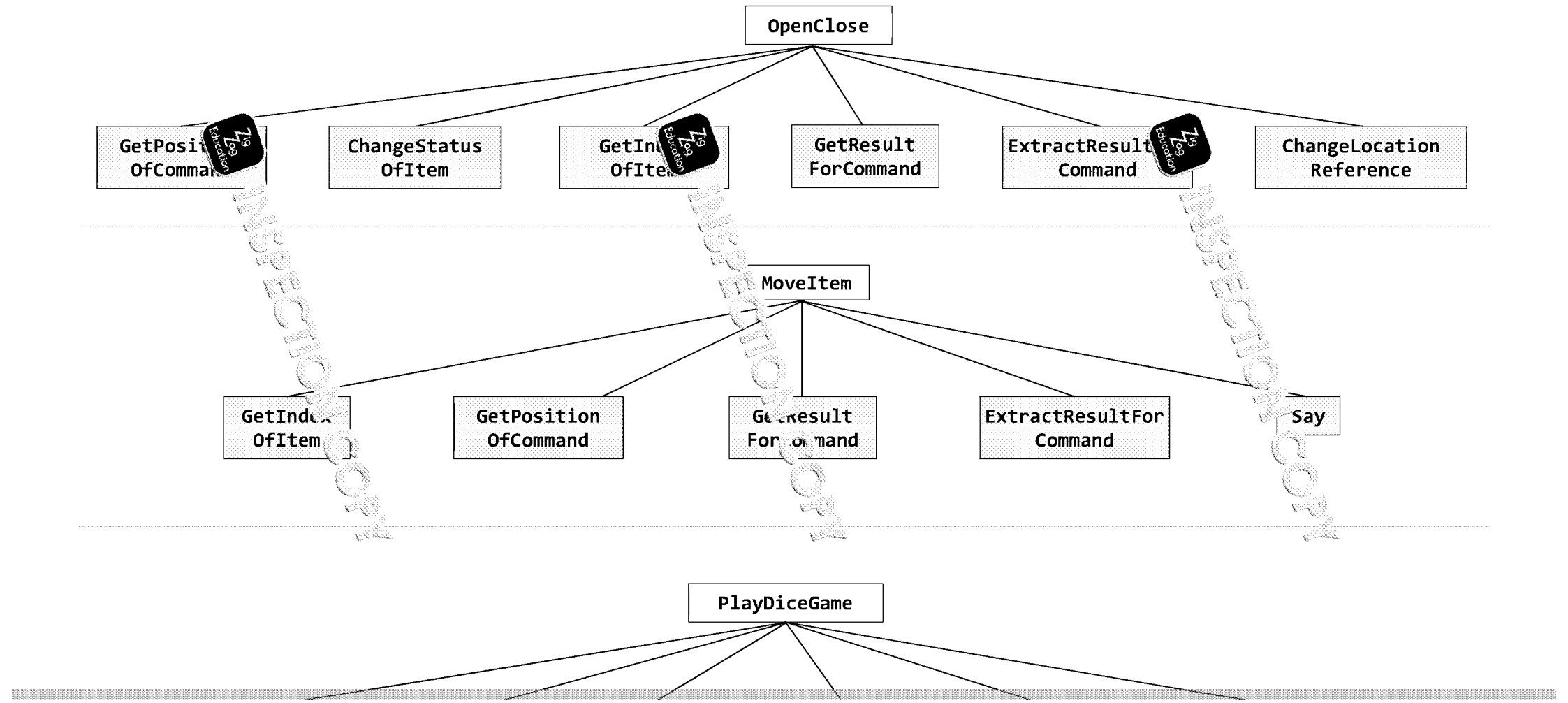
Program Flow Chart



Program Structure Diagrams







Program Subroutines

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

Subroutine	Data	Description
GetInstruc[illegible]	Parameters: – Returns: Instruction(string) Called From: PlayGame Calls: –	1. Prints a line separator 2. Creates the variable Instruction, which is set to the value of the user's input converted to lowercase 3. Returns Instruction
ExtractCommand Ⓕ	Parameters: Instruction(string) Returns: Command(string) Instruction(string) Called From: PlayGame Calls: –	1. If Instruction doesn't contain a space, then returns Instruction as both the Command and Instruction variables 2. Otherwise the characters in Instruction are added to the Command variable until a space is found in Instruction 3. Instruction is set to all of the characters in the Instruction that come after the first space 4. Returns Command and Instruction
Go Ⓕ	Parameters: You(Character) Direction(string) CurrentPlace(Place) Returns: You(Character) Moved(Boolean) Called From: PlayGame Calls: –	1. Creates the variable Moved and sets it to *True* 2. Direction is checked to see if it is a valid direction: "north", "east", "south", "west", "up" or "down" 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 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* 5. If an invalid direction is given, the value of Moved is set to *False* 6. If Moved is *False*, displays the message "You are not able to go in that

Subroutine	Data	Description
`DisplayContentsOf ContainerItem` Ⓟ	Parameters: `Items([Item])` `ContainerID(int)` Returns: – Called From: `Examine` Calls: –	1. Displays the message "It contains: " (end = "" stops the next message from being displayed on a new line) 2. Creates the variable `ContainsItem` and sets it to *False* 3. Loops through all of the items, and checks whether each item is in the location `ContainerID` 4. If the item is in the location `ContainerID` and `ContainsItem` is True, prints ",", sets `ContainsItem` to *True* and prints the name of the item 5. If the item is in the location `ContainerID` and `ContainsItem` is *False*, sets `ContainsItem` to *True* and prints the name of the item 6. If `ContainsItem` is *True*, prints "." 7. If `ContainsItem` is *False*, prints "nothing"
`Examine` Ⓟ	Parameters: `Items([Item])` `Characters([Character])` `ItemToExamine(string)` `CurrentLocation(int)` Returns: – Called From: `PlayGame` Calls: `DisplayInventory` `GetIndexOfItem` `DisplayDoorStatus` `DisplayContentsOfContainerItem`	1. Creates the variable `Count` and sets it to 0 2. If `ItemToExamine` is "inventory", displays inventory 3. Otherwise, gets the index of the item 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 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 6. Loops through each character in `Characters` and if the character's name is equal to `ItemToExamine`, displays the description of that

Subroutine	Data	Description
GetPositionOfCommand Ⓕ	Parameters: CommandList(string) Command(string) Returns: Position(int) Called From: GetItem UseItem ReadItem MoveItem OpenClose PlayDiceGame Calls: –	1. Creates the variable Position and sets it to *0* 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 3. If the substring is equal to Command, Position is returned 4. If the substring begins with ",", the value of Position is increased by 1 5. Once all substrings have been checked, if Command was not found, Position is returned
GetResultForCommand Ⓕ	Parameters: Results(string) Position(int) Returns: ResultForCommand(string) Called From: GetItem UseItem ReadItem MoveItem OpenClose PlayDiceGame Calls: –	1. Creates the variable ResultForCommand and sets it to "" 2. Loops through each character in Results until as many ";"s have been read as the value of Position or the end of the string has been reached 3. Any remaining characters, up until the next ";" or the end of the string, are loaded into ResultForCommand 4. ResultForCommand is returned
Say Ⓟ	Parameters: Speech(string) Returns: – Called From: PlayGame GetItem UseItem MoveItem	1. Prints empty line 2. Prints Speech 3. Prints empty line

Subroutine	Data	Description
ExtractResult ForCommand Ⓕ	Parameters: SubCommand(string) SubCommandParameter(string) ResultForCommand(string) Returns: SubCommand(string) SubCommandParameter(string) Called From: GetItem UseItem ReadItem MoveItem OpenClose Calls: –	1. Loads each character from ResultForCommand into SubCommand in sequence until a ";" is read or the end of the string is reached 2. Any remaining characters up until the next ";", "," or the end of the string, are loaded into SubCommandParameter 3. Returns SubCommand and SubCommandParameter
ChangeLocation Reference Ⓕ	Parameters: Direction(string) NewLocationReference(int) Places([Place]) IndexOfCurrentLocation(int) Opposite(Boolean) Returns: Places([int]) Called From: OpenClose Calls: –	1. Creates the ThisPlace object and has the properties of the default Place object 2. Sets ThisPlace to be equal to the location given by IndexOfCurrentLocation 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 4. The location given by IndexOfCurrentLocation is set to be equal to ThisPlace 5. Returns Places

Subroutine	Data		Description

`OpenClose` Ⓕ	Parameters:	`Open(Boolean)`
		`Items([Item])`
		`Places([Place])`
		`ItemToOpenClose(st`
		`CurrentLocation(int`
			1. If `Open` is *True*, creates the variable `Command` and sets it to "open"
			2. If `Open` is *False*, creates the variable `Command` and sets it to "close"
			3. Loops through each item in `Items` and if the `e` of the item is equal to `ItemToOpenClose`, the item is in `Current` `ion` and the string of that item's commands is at least 4 characters long, chec`e` status of the item

	Returns:	`DirectionChange(int)`
		`Items([Item])`
		`Places([Place])`
			4. If the status of the item is equal to `Command`, re`s` *-2*, `Items` and `Places`
			5. If the status of the item is "locked", returns *-3*, `I` and `Places`
			6. Otherwise, creates the variable `Position` and set`o` be equal to the position of `Command` in the door's list of commands, creates th`riable` `ResultForCommand` and sets it to be equal to the result of using `Comman` `o` the door
			`.` Changes the status of the door
			8. Sets `ActionWorked` to *True*
			8. Loops through each place to change the appropriate loc`n` references in `CurrentLocation` and the location that is linked to `CurrentLocation` by the `do`r
			9. C`a`nges the status of the other side of the door
			11. If `ionWorked` is *False*, returns *-1*, `Items` and `Places`
			12. If `A` `onWorked` is *True*, returns the integer value of `Direct` `Change`, `Items` and `Places`

	Called From:	`PlayGame`	

	Calls:	`GetPositionOfCommand`
		`GetResultForCommand`
		`ExtractResultForCommand`
		`ChangeStatusOfItem`
		`GetIndexOfItem`
		`ChangeLocationReference`	

`GetIndexOfItem` Ⓕ	Parameters:	`ItemNameToGet(string)`
		`ItemIDToGet(int)`
		`Items([Item])`
			1. Creates the variable `Count` and sets it to *0*, and creates the variable `StopLoop` and sets it to *False*
			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*

	Returns:	`Count(int)`	
	Called From:	`GetItem`	

Subroutine	Data		Description

`ChangeLocationOfItem` Ⓕ	Parameters:
	Returns:
	Called From:
	Calls:
		`Items([Item])`
		`IndexOfItem(int)`
		`NewLocation(int)`
		`Items([Item])`
		`GetItem`
		`TakeItemFromOtherCharacter`
		`TakeRandomItemFromPlayer`
		–
			1. Creates the `ThisItem` object that has the properties of the default `Item` object
			2. Sets `ThisItem` to be equal to the given item twice. Doing this the second time provides no benefit
			3. Sets the location of `ThisItem` to be equal to `NewLocation`
			4. Sets the given item to be equal to `ThisItem`
			5. Returns `Items`

`ChangeStatusOfItem` Ⓕ	Parameters:
	Returns:
	Called From:
	Calls:
		`Items([Item])`
		`IndexOfItem(int)`
		`NewStatus(string)`
		`Items([Item])`
		`ChangeStatusOfDoor`
		`OpenClose`
		–
			1. Creates `ThisItem` object that has the properties of the default `Item` object
			2. Sets `ThisItem` to be equal to the given item
			3. Sets the status of `ThisItem` to be equal to `NewStatus`
			4. Sets the given item to be equal to `ThisItem`
			5. Returns `Items`

`GetRandomNumber` Ⓕ	Parameters:
	Returns:
	Called From:
	Calls:
		`LowerLimitValue(int)`
		`UpperLimitValue(int)`
		`(int)`
		`RollDie`
		`TakeRandomItemFromPlayer`
		–
			1. Returns random integer number between `LowerLimitValue` and `UpperLimitValue`

`RollDie` Ⓕ	Parameters:
	Returns:
	Called From:
		`Lower(string)`
		`Upper(string)`
		`(int)`
		`UseItem`
			1. Creates `LowerLimitValue` variable, and sets it to *0*
			2. If the `Lower` string is made up entirely of numeric digits, sets `LowerLimitValue` to the integer value of `Lower`
			3. Otherwise, sets `LowerLimitValue` to be equal to the integer value of

Subroutine	Data	Description

`ChangeStatusOfDoor` Ⓕ	Parameters: `Items([Item])`
	`CurrentLocation(int)`
	`IndexOfItemToLockUnlock(int)`
	`IndexOfOtherSideItemToLockUnlock(int)`
	Returns: `Items([Item])`
	Called From: `UseItem`
	Calls: `ChangeStatusOfItem`
	`Say`
		1. Checks whether the given door or the other side of the given door is in `CurrentLocation`
		2. If either is *True* and the status of the door is "locked", then the statuses of the door and the other side of the door are set to "close", and a message is displayed to say that the door has been unlocked
		3. If the status of the door is "close", then the statuses of the door and the other side of the door are set to "locked" and a message is displayed to say that the item has been locked
		4. Otherwise, displays a message to say that the item is open and so cannot be locked
		5. If neither the door nor the other side of the door is in `CurrentLocation`, displays a message to say that the key cannot be used in that location

`UseItem` Ⓕ	Parameters: `Items([Item])`
	`ItemToUse(string)`
	`CurrentLocation(int)`
	`Places([Place])`
	Returns: `StopGame(Boolean)`
	`Items([Item])`
	Called From: `PlayGame`
	Calls: `Say`
	`ExtractResultForCommand`
	`GetResultForCommand`
	`GetPositionOfCommand`
	`GetIndexOfItem`
		1. Gets the index of the given item
		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"
		3. 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
		4. If `SubCommand` of the result is "say", then displays the `SubCommandParameter` of the result
		5. If `SubCommand` of the result is "lockunlock", then changes the status of the relevant doors

Subroutine	Data	Description

`ReadItem` Ⓟ	Parameters: `Items([Item])`
	`ItemToRead(string)`
	`CurrentLocation(int)`
	Returns: –
	Called From: `PlayGame`
	Calls: `GetIndexOfItem`
	`GetPositionOfCommand`
	`GetResultForCommand`
	`ExtractResultForCommand`
		1. Gets the index of the given item
		2. If the index of the item couldn't be found, then displays a message to say that the item cannot be found
		3. If the item doesn't have "read" in its list of commands, displays a message to say that the item cannot be read
		4. If the item is not in `CurrentLocation` or `INVENTORY`, displays a message to say that the item cannot be found
		5. Otherwise, gets the position of "read" in the item's list of commands and gets the result of using the "read" command on that item
		6. If `SubCommand` of the result is "say", then displays the `SubCommandParameter` of the result

`GetItem` Ⓕ	Parameters: `Items([Item])`
	`ItemToGet(string)`
	`CurrentLocation(int)`
	Returns: `(Boolean)`
	`Items([Item])`
	Called From: `PlayGame`
	Calls: `Say`
	`ExtractResultForCommand`
	`GetResultForCommand`
	`GetPositionOfCommand`
	`GetIndexOfItem`
	`ChangeLocationOfItem`
		1. Gets the index of the given item
		2. If the index of the item couldn't be found, then displays a message to say that the item cannot be found
		3. If the item is in `INVENTORY`, displays a message to say that the item is already in the inventory
		4. If the item doesn't have the command "get", displays a message to say that the player can't get the item
		5. If the item is stored in another item and the location of the item that contains it is not `CurrentLocation`, displays a message to say that the item cannot be found
		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
		7. If the item was found and has "get" in its list of commands, gets the position of "get" in the

Subroutine	Data	Description
`CheckIfDiceGame Possible` Ⓕ	Parameters: `Items([Item])` `Characters([Character])` `OtherCharacterName(string)` Returns: `(Boolean)` `IndexOfPlayerDie(in` `IndexOfOtherCharacte(int)` `IndexOfOtherCharacterDie(int)` Called From: `PlayDiceGame` Calls: `GetIndexOfItem`	1. Loops through each item in `Items` and if an item is in `INVENTORY` and has "die" in its name, gets the index of that die and sets `PlayerHasDie` to *True* 2. Loops through each character in `Characters` until a character is found in the same location as the player and h name `OtherCharacterName` or all of the characters have been ch 3. If a character is found that has the name `OtherCharacterName` and is in the same location as the player, sets `PlayersInSameRoom` to *True* and loops through each item in `Items` 4. If an item is held by the other character and h "die" in its name, gets the index of that die and sets `OtherCharacterHasDie` to *True* 5. Returns result of (`PlayerHasDie` **and** `PlayersInSameRoom` **and** `OtherCharacterHasDie`), `IndeoxOfPlayerDie`, `IndexOfOtherCharacter` and `IndexOfOtherCharacterDie`
`TakeItemFrom OtherCharacter` Ⓕ	Parameters: `Items([Item])` `OtherCharacterID(int)` Returns: `Items([Item])` Called From: `PlayDiceGame` Calls: `ChangeLocationOfItem`	1. Loops through all of the items in `Items`, and chec whether each item is held by the given character 2. If the item is held by that character, the name of tha m is added to a list 3. The message "Which item do you want to take? They ve:" is displayed, followed by each item in the list, with each item separate by "," and the list ending with ".", although the last item in the list is not displayed unless it is the only item in the list 4. Gets the result of user input 5. If the user inputs the name of an item in the list, displays a message to say that the player now has that item and the item is moved to the player's inventory 6. Otherwise, displays a message to say that the character doesn't have that item

Subroutine Name	Data		Description

`PlayDiceGame` Ⓕ	Parameters:
	Returns:
	Called From:
	Calls:
		`Characters([Character])`
		`Items([Item])`
		`OtherCharacterName(string)`
		`Items([Item])`
		`PlayGame`
		`CheckIfDiceGamePossible`
		`GetPositionOfCommand`
		`GetResultForCommand`
		`RollDie`
		`TakeItemFromOtherCharacter`
		`TakeRandomItemFromPlayer`
			1. Calls `CheckIfDiceGamePossible` to see if a dice game can be played
			2. If a dice game can't be played, displays the message "You can't play a dice game."
			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
			4. Displays a message to show what the player rolled
			5. Gets the position of "use" in the other character'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
			6. Displays a message to show what the other character rolled
			7. If the player's roll was higher than the other character's, display a message to say that the player has won, and allow them to take an item from that character
			8. If the player's roll was lower than the other character's, display a message to say that the player has lost, and give one of their items to the other character
			9. If the player's roll was the same as the other character's, display a message to say that the game was a draw
			10. Return `Items`

`MoveItem` Ⓟ	Parameters:
	Returns:
	Called From:
	Calls:
		`Items([Item])`
		`ItemToMove(string)`
		`CurrentLocation(int)`
		–
		`PlayGame`
		`GetIndexOfItem`
		`GetPositionOfCommand`
		`GetResultForCommand`
			1. Gets the index of the given item
			2. If the index of the item is found, then checks the length of the item's list of commands
			3. If the list of commands has at least 4 characters and contains "move", gets the position of "move" in the item's list of commands and gets the result of using the "move" command on that item
			4. If `SubCommand` of the result is "say", then displays the `SubCommandParameter` of

Subroutine	Data	Description
`DisplayInventory` Ⓟ	Parameters: `Items([Item])` Returns: – Called From: `Examine` Calls: –	1. Prints empty line 2. Prints "You are currently carrying the following items:" 3. Loops through every item in `Items` and if the location of that item is `INVENTORY`, prints the name of that item
`DisplayGettableItemsInLocation` Ⓟ	Parameters: `Items([Item])` `CurrentLocation(int)` Returns: – Called From: `PlayGame` Calls: –	1. Loops through each item in `Items` and checks whether the item is in `CurrentLocation` and whether its status contains "gettable" 2. If both of these conditions are *True,* appends the name of the item to a string starting "On the floor there is:" 3. If the name of another item has already been added to the string, prints a "," before the name of the next item If the name of any item was appended to the string, a "." is appended to the end
`DisplayOpenCloseMessage` Ⓟ	Parameters: `ResultOfOpenClose(int)` `OpenCommand(Boolean)` Returns: – Called From: `PlayGame` Calls: `Say`	1. If `ResultOfOpenClose` is greater than *0*, then checks `OpenCommand` 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 3. If `ResultOfOpenClose` is *-3*, displays a message to say that the door is locked 4. If `ResultOfOpenClose` is *-2*, displays a message to say that the door is already in that state 5. If `ResultOfOpenClose` is *-1*, displays a message to say that the item can't be opened

Subroutine	Data	Description
PlayGame Ⓟ	Parameters: Characters([Character]) Items([Item]) Places([Place]) Returns: – Called From: Main Calls: DisplayGettableItemsInLocation GetInstruction ExtractCommand GetItem UseItem Go ReadItem Examine OpenClose DisplayOpenCloseMessage MoveItem PlayDiceGame Say	1. If the game has ended, waits for user input and then terminates, otherwise the subroutine runs the following loop 2. If the player has moved to a new location, prints two empty lines, the description of the location and the gettable items at that location 3. Gets input from the user, converts it to lowercase and splits it into a Command and an Instruction 4. If Command is "get", tries to get the given item 5. If Command is "use", tries to use the given item 6. If Command is "go", tries to move the character in the given direction 7. If Command is "read", tries to read the given item 8. If Command is "examine", tries to examine the given item, character or inventory 9. If Command is "open", tries to open the given item 10. If Command is "close", tries to close the given item 11. If Command is "move", tries to move the given item 12. If Command is "say", calls displays the Instruction of the input 13. If Command is "playdice", tries to play a dice game with the given character 14. If Command is "quit", displays a message to say that the player has given up and ends the game 15. If Command is anything else, displays a message to say that the command isn't recognised

Subroutine	Data	Description

`LoadGame` Ⓕ	Parameters: `Filename(string)`
	`Characters([Character])`
	`Items([Item])`
	`Places([Place])`
	Returns: `(Boolean)`
	`Characters([Character])`
	`Items([Item])`
	`Places([Place])`
	Called From: `Main`
	Calls: –
		1. If an exception is thrown, returns *False*, `Characters`, `Items` and `Places`, otherwise the subroutine runs as follows
		2. Creates the a file object `f` to read the contents of the given file
		3. Reads the number of characters from the file
		4. Creates `TempCharacter`, sets its properties to the values read from the file and adds this character to `Characters`. This is repeated for each character.
		5. Reads the number of places from the file
		6. Creates `TempPlace`, sets its properties to the values read from the file and adds this place to `Places`. This is repeated for each place.
		7. Reads the number of items from the file
		8. Creates `TempItem`, sets its properties to the values read from the file and adds this item to `Items`. This is repeated for each item.
		9. Returns *True*, `Characters`, `Items` and `Places`

`Main` Ⓟ	Parameters: – Returns: – Called From: – Calls: `LoadGame` `PlayGame`	1. Creates `Items`, `Characters` and `Places` variables as empty lists 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) 3. Prints an empty line 4. Tries to load the game from the given file 5. If the game is successfully loaded, calls `PlayGame` 6. If the game is not successfully loaded, displays a message to say that the game could not be loaded and terminates the program after the user enters input

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

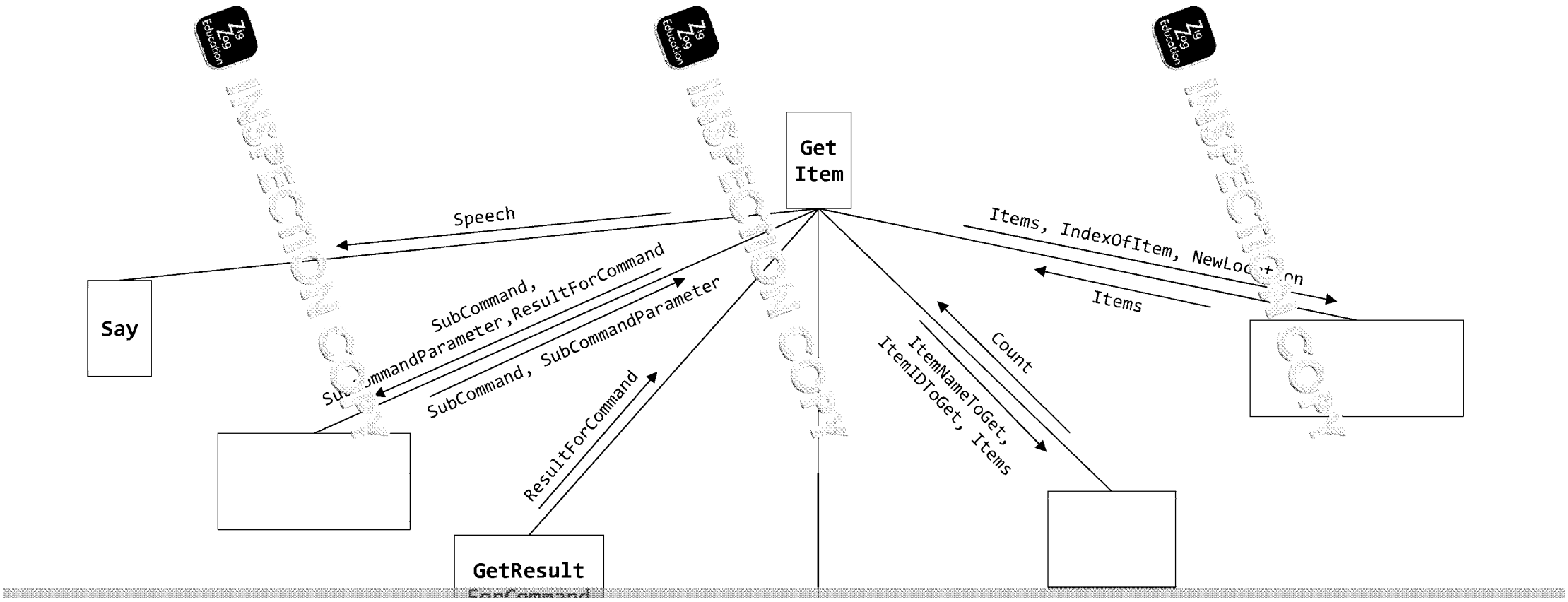
Attribute	Type	Default Value	Description
Description	String	""	The text description of the place, which is shown to the player whenever they move to this place
ID	Integer	0	The ID of this place, which allows it to be referenced by the program or other objects
North	Integer	0	The ID of the location to 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

Item Attributes

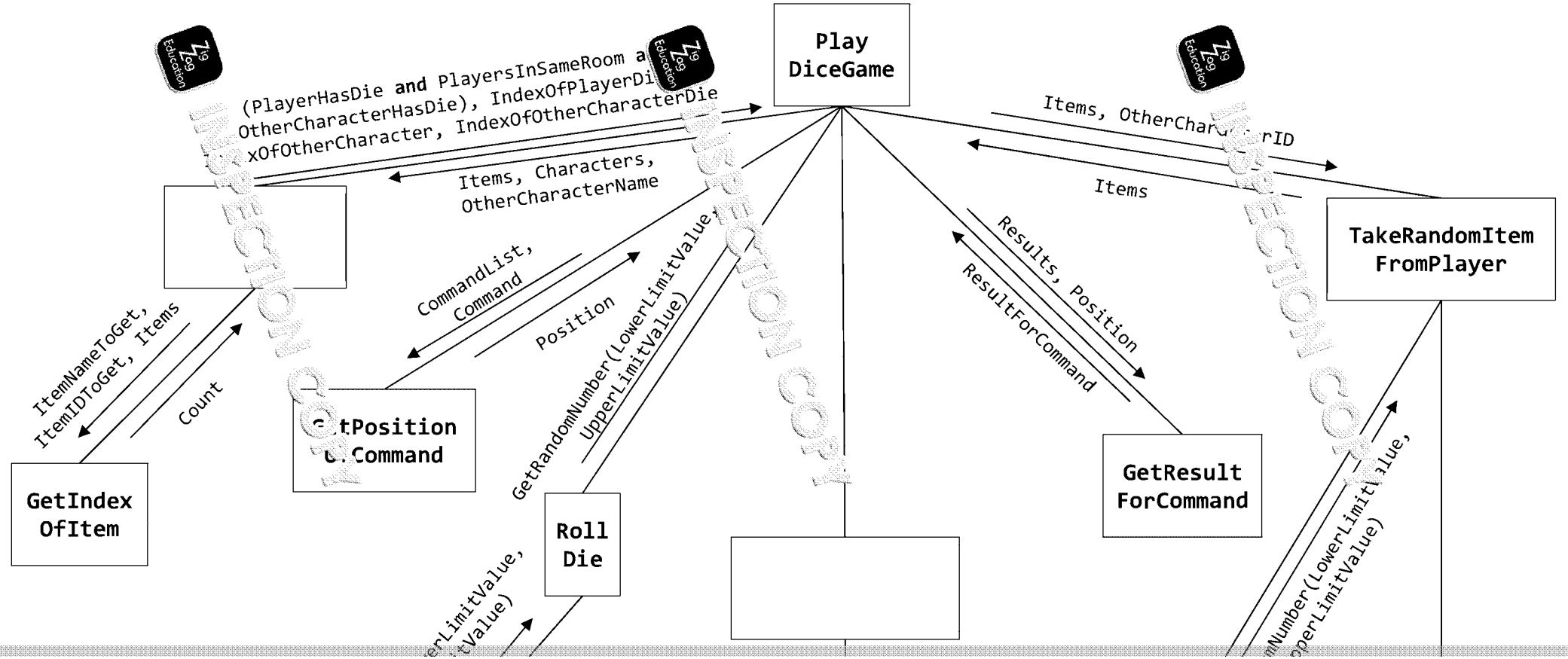
Attribute	Type	Default Value	Description
ID	Integer	0	The ID of this item, which allows it to be referenced by the program or other objects
Location	Integer	0	The ID of the place, item or character holding this item; if this is 0, then the item is not held by a place, item or character
Description	String	""	The text description of the item, which is shown to the player whenever they examine this item
Status	String	""	A string containing various attributes of an item (e.g. if a door has the status "locked", it cannot be opened)
Name	String	""	The name of the item, which is displayed to the player and used by the player to interact with this item
Commands	String	""	A string containing the commands that can be used on this item (such as "get", "use" and "close")
Results	String	""	A string containing the results of using commands on this item (the default result moves the item to `INVENTORY`)

Structure Diagram Activity 1 (GetItem)



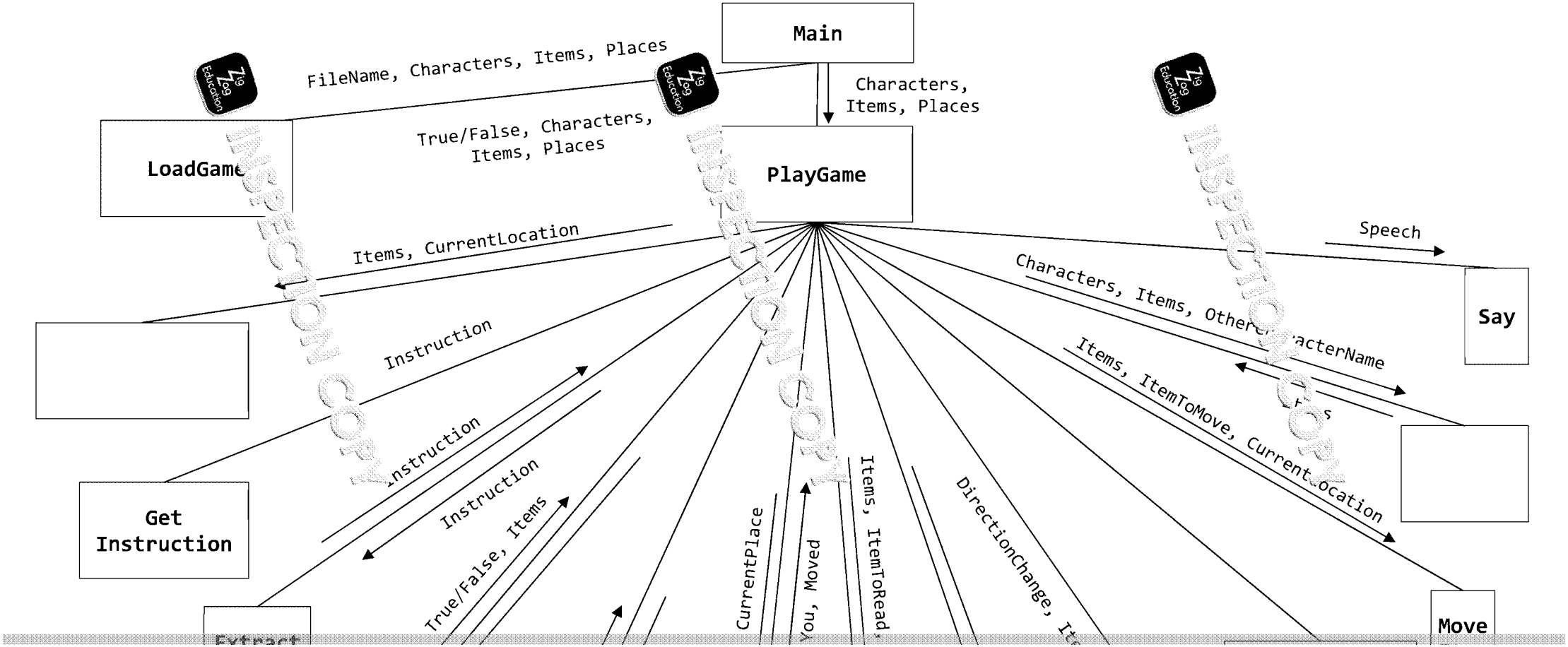
Structure Diagram Activity 2 (PlayDiceGame)

AQA Text Adventures



Structure Diagram Activity 3 (Main)

AQA Text Adventures



AQA Text Adventures

Written Questions (Python)

These questions refer to the preliminary material and require you to load the ske any additional programming.

1. State the name of an appropriate identifier:
   a) A local variable in the function `RollDie` [1]
   b) A function that returns multiple variables [1]
   c) Four built-in functions used in the function `TakeItemFromOtherCha`
   d) A class that has exactly four attributes [1]
   e) A constructor [1]
   f) A variable used as a constant [1]
   g) Two user-defined functions with four parameters [2]
   h) A user-defined function, other than `Main`, that takes no arguments [1]
2. Explain the purpose of the following line in the `GetInstruction` subroutir

```
Instruction = input("> ").lower()
```

3. Explain the role of the variable `Moved` in the function `Go`. [3]
4. Explain the purpose of the command `break` used in the function `GetResult`
5. Using an example from the Skeleton Program, describe what it meant by the
6. Describe in detail the role of the procedure `DisplayInventory`. [5]
7. Describe the role of the variable `TempCharacter` in the function `LoadGam`
8. Explain why the `LowerLimitValue` in the function `RollDie` is initialised
9. Explain the purpose of each of the two while loops in the function `Extract`
10. Describe the purpose of the first `for` loop in the function `CheckIfDiceGa`
11. 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]
12. Describe the changes that would need to be made to the function `GetInstr` two or more characters has been entered before `GetInstruction` returns
13. Describe the effect of a file not being found during the function `LoadGame`. [

- END OF TEST -

AQA Text Adventures

Written Questions (Python)

These questions refer to the preliminary material and require you to load the sk... you to load the sk... require any additional programming.

1. State the name of an appropriate item:

   a) A lo... iable in the function `RollDie` [1]

   .....

   b) A function that returns multiple variables [1]

   .....

   c) Four built-in functions used in the function `TakeItemFromOtherCha...`

   .....

   .....

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

   .....

   e) A constructor [1]

   .....

   f) A variable u... constant [1]

   .....

   g) Two user-defined functions with four parameters [2]

   .....

   h) A user-defined function, other than `Main`, that takes no arguments [1]

   .....

2. Explain the purpose of the following line in the `GetInstruction` subroutin...

```
Instruction = input("> ").lower()
```

.....

.....

.....

.....

3. Explain the role of the variable `Moved` in the function `Go`. [3]

4. Explain the purpose of the comm[illegible]b[illegible]ax used in the function `GetResul`

5. Using any example from the Skeleton Program, describe what it meant by the

6. Describe in detail the role of the [illegible]`DisplayInventory`. [5]

7. Describe the role of the variable `TempCharacter` in th[illegible] function `LoadGa`

8. Explain why the `LowerLimitValue` in the function `RollDie` is initialised

.....

.....

.....

.....

.....

9. Explain the purpose of each of the two while loops in the function `ExtractC`

While loop 1

.....

.....

.....

.....

While loop 2

.....

.....

.....

10. Describe the purpose of the first `for` loop in the function `CheckIfDiceGa`

.....

.....

.....

.....

.....

.....

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

.....

.....

.....

.....

.....

.....

12. Describe the changes that would need to be made to the function `GetInstr` of two or more characters has been entered before `GetInstruction` retu

.....

.....

.....

.....

.....

.....

13. Describe the effect of a file not being found during the function `LoadGame`.

.....

.....

.....

.....

- END OF TEST -

AQA Text Adventures

Programming Tasks (Python)

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 enters an invalid command they are told '**Sorry, you don't know how to...**'

Add a new subroutine called `ShowHelp`, which should display the following text:

You can enter one of the following commands:
go, get, use, examine, say, quit, read, move, open, close and playdice.

Modify the subroutine `PlayGame` so that if a user enters the command 'help', the `ShowHelp` subroutine is called.

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 'flag1'
  - Type 'help' at the first prompt

Task 2

This question refers to `Examine` and `PlayGame`.

Currently, users are able to examine their inventory, as well as objects within a room.

Modify `Examine` so that if the user has entered the text 'examine room', the description of the room should be re-displayed. A call to `DisplayGettableItemsInLocation` should also be made.

You should pass `Places` 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 'flag1'
  - 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

Modify `PlayGame` so that an entry of 'quit' triggers an output of '**Are you sure?**
Entering either 'y' or 'yes' in any combination of upp[illegible]owercase should termi
'**You decide to give up, try again another ti[illegible]st.**' being displayed.

Any other input should ret[illegible]layer to the game and the main prompt.

Evidence y[illegible]d to provide:

- Your amended SOURCE CODE for `PlayGame`
- One screenshot 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 `RollDie`.

Currently, if you enter a non-integer value w[illegible]pted for a minimum or max
throw an error.

Modify `Roll[illegible]` s[illegible]e user enters a non-integer value, the error is handl
another va[illegible] they enter an integer from 1 to 6.

Evidence you need to provide:

- Your amended SOURCE CODE for `RollDie`
- One screenshot showing the output that results from the following:
  - Run the program and load the file 'flag2'
  - Type 'playdice guard' at the first prompt
  - Type 'one' at the 'Enter minimum: ' prompt
  - Type '1.0' at the next 'Enter minimum: ' prompt
  - Type '1' at the next 'Enter minimum: ' prompt
  - Type 'six' at the 'Enter maximum: ' prompt
  - Type '6.0' at the next 'Enter maximum: ' prompt
  - Type '6' at the next 'Enter maximum: ' [illegible]

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 th[illegible] a bubble sort to so
alphabetical order based on their `Name` att[illegible] before displaying the list of ite
algorithm yourself, and should n[illegible] library sorting functions.

Evidence y[illegible]d [illegible]ide:

- You[illegible]ded SOURCE CODE for `DisplayInventory`
- One screenshot showing the output that results from the following:
  - Run 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.

Create an integer constant called `MAX_INVENTO[illegible]` an[illegible]et it to 4. When an att
player with four or more items already in [illegible]ventory, the message '**Inventory f**
The check for a full inventory should take place before any existing selection state
messages should be [illegible]o within `GetItem` other than 'Inventory full'.

This check s[illegible]also be applied to `TakeItemFromOtherCharacter` so tha
after winning a dice game, the 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 [illegible]ts [illegible]m the following:
  - Run the program and lo[illegible]ile 'flag2'
  - Type 'playdice g[illegible]a[illegible]first subsequent prompt
  - At each [illegible]e two prompts, type '6' (repeat the previous ste
  - [illegible]yp[illegible] the next prompt

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 called `DropItem` which accepts three parameters:

- A variable called 'Items', which will contain all items in the game
- A variable called 'ItemToDrop', which will be the name of the item leaving
- A variable called 'CurrentLocation', which will be the ID of the player's loc

`DropItem` should check that the player is carrying the item. If they are not, the should be displayed (with the item name displayed in place of _____). Otherwise, room with the text '_____ **dropped**' indicating which item was 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 prom

Task 8

This question refers to `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

If the user enters a file name that contains the string '.gme' the program should n

You should only make changes to `Main`.

Evidence you need to provide:

- Your amended SOURCE CODE for `Main`
- One screenshot showing the output that results from the following:
  - Type 'flag3' when prompted for a file name
  - At the next prompt, type 'flag2.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 `DisplayCharact` parameter in the form of `Characters`.

When `DisplayCharacters` is called it should display four column headers in

- ID (followed by three spaces)
- NAME (followed by eight spaces)
- DESCRIPTION (followed by 53 spaces)
- CURRENTLOCATION

Beneath these columns, all characters within the game should be displayed, with the attributes `ID`, `Name`, `Description` and `CurrentLocation` should each column header.

Modify `PlayGame` so that if 'debugchar' is entered during gameplay, a call to `Di` made.

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 filename
  - At the next prompt, type 'debugchar'

Task 10

This question relates to `PlayGame` as well as a new subroutine `FillVessel`.

In the flag1 game, there is a room below the starting location that contains a barr
subroutine called `FillVessel` that requires the following parameters

- A variable called 'Characters', that stores all ... ch ...racters in the game
- A variable called 'Items', that stor... ...ot ... items in the game
- A variable called 'Places' ... ...s all of the places in the game
- A variable call... ...st', that stores the name of the item to be filled

`FillVess`...uld 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 '**You cannot do that**' should be
item's `Name` attribute should have the text ' **of water**' appended to it, and the text
displayed, with the vessel appended to the end.

If the vessel's name attribute already contains the text 'of water', the text '**Already**

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 new `FillVessel` subroutine
- One screenshot showing the ... ...t results from the following:
  - Type 'flag1' ... ...pted for a file name
  - At th... ...rompt, type 'open trapdoor'
  - ... ... next prompt, type 'go down'
  - ... 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 variable called 'Pointer', that is initially set to *-1*
- A five-element list called 'Locations', that is initially filled with empty valu

`History` will use a stack, to allow players to backtrack through previous locatio use an implementation of the `Locations` array, which will be subject to two su

- `Pu` ...uld require two parameters called 'self' and 'NewLocation'. If t sho... be moved down by 1 element (so the content of element 0 is lost), should be added to the top. If the stack is not full, the pointer attribute s parameter should be added to the element indicated by `Pointer`. The

- `Pop` requires only a parameter called 'self' and returns an integer. If the be returned. Otherwise the value in the element indicated by `Pointer` should be decremented.

In `PlayGame`, create an instance of the `History` class called 'past'. When the command, this instance should be passed to the `Go` subroutine as an additional p

In `Go`, modify the parameters to accept the instance of the `History` class. As a location onto the stack before they move to a new one. If the instruction 'go bac their previous location using a call to the `Pop` subroutine of `History`. If the cal display the text '**You cannot go back**'.

Evidence you need to upload:

- You... SOURCE CODE for the new `History` class
- You... 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 and a

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

Create a new subroutine called `EatItem` which requires as parameters the list o and an `ItemToEat` variable.

If `ItemToEat` matches the name of an item in the player's inventory, and that it the text 'edible', increase the player's `Health` property by 5 and remove the eat conditions are not met, the text 'You cannot eat that' should be displayed. If an it should display the current 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 SOURCE CODE for the new `EatItem` subroutine
- Your amended SOURCE CODE for `PlayGame`
- One screenshot showing the output that results from the following:
  - Type 'flag1' when prompted for a file name
  - At the next prompt, type 'eat apple'
  - At the next prompt, 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.

Modify `TakeItemFromOtherCharacter` so that if you request an item that **'They don't have a _____ – try again'** is displayed (with _____ replaced by the iter to take).

The player is then asked again which item they want to take, with the available it happen until the player has entered an item that the other player possesses, at w player's inventory as normal.

Evidence you need to provide:

- Your amended SOURCE CODE for the `TakeItemFromOtherCharact`
- One screenshot showing the output that results from the following:
  - Type 'flag1' when prompted for a file name
  - At the next prompt, type 'playdice guard'
  - For each of the next two prompts, type '6' (repeat the previous ste
  - When asked to choose an item, type 'box'
  - 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 c** whether the player has no die, the opponent has no die or the player and the opp

Modify `CheckIfDiceGamePossible` so that the following messages are outp

1. If the player has no die, the output should read '**Player has no die**'
2. If the player and opponent are not in the same room, the output shou ____ being replaced by the name of the other character)
3. If the opponent has no die, the output should be '**_____ has no die**' (with of the other 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 guard'
  - At the next prompt, type 'get red die'
  - At the next prompt, type 'playdice 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 alongside the n displayed in round brackets. Additionally, the opening brackets of the description 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 name
  - At the next prompt, type 'examine inventory'

AQA Text Adventures

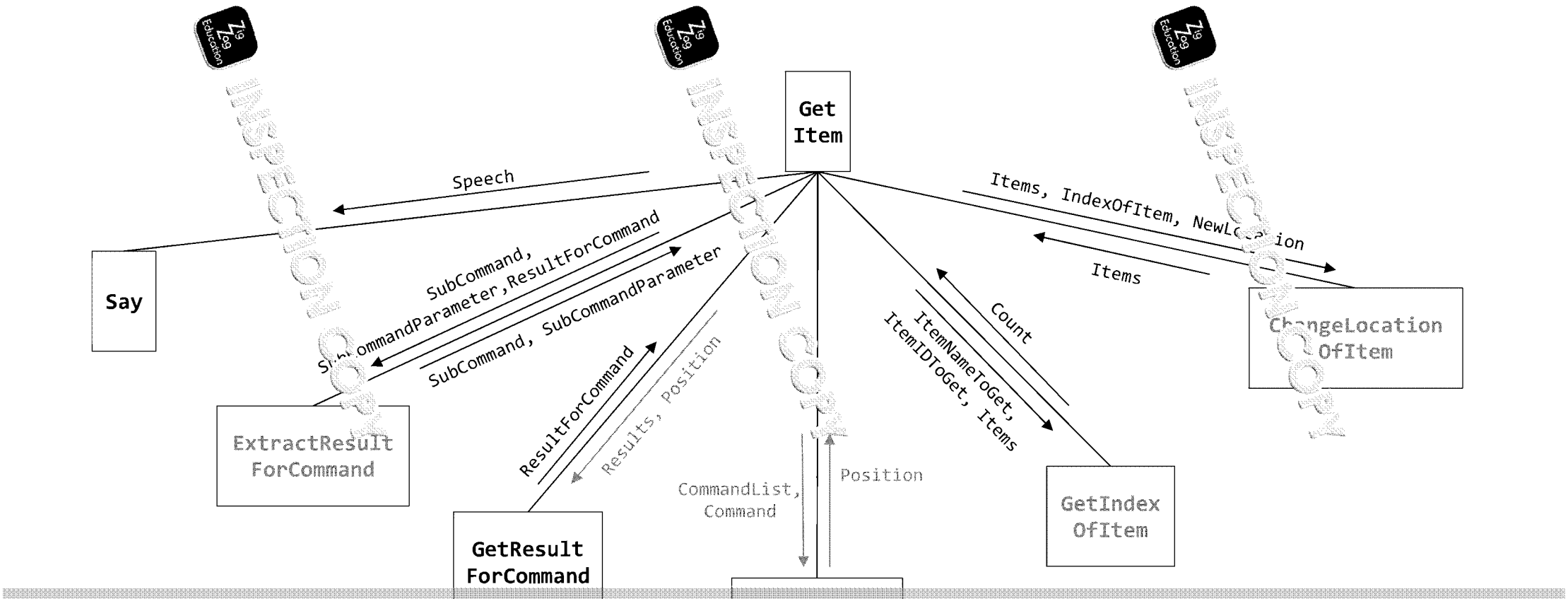
Programming Tasks - Extension (Python)

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.

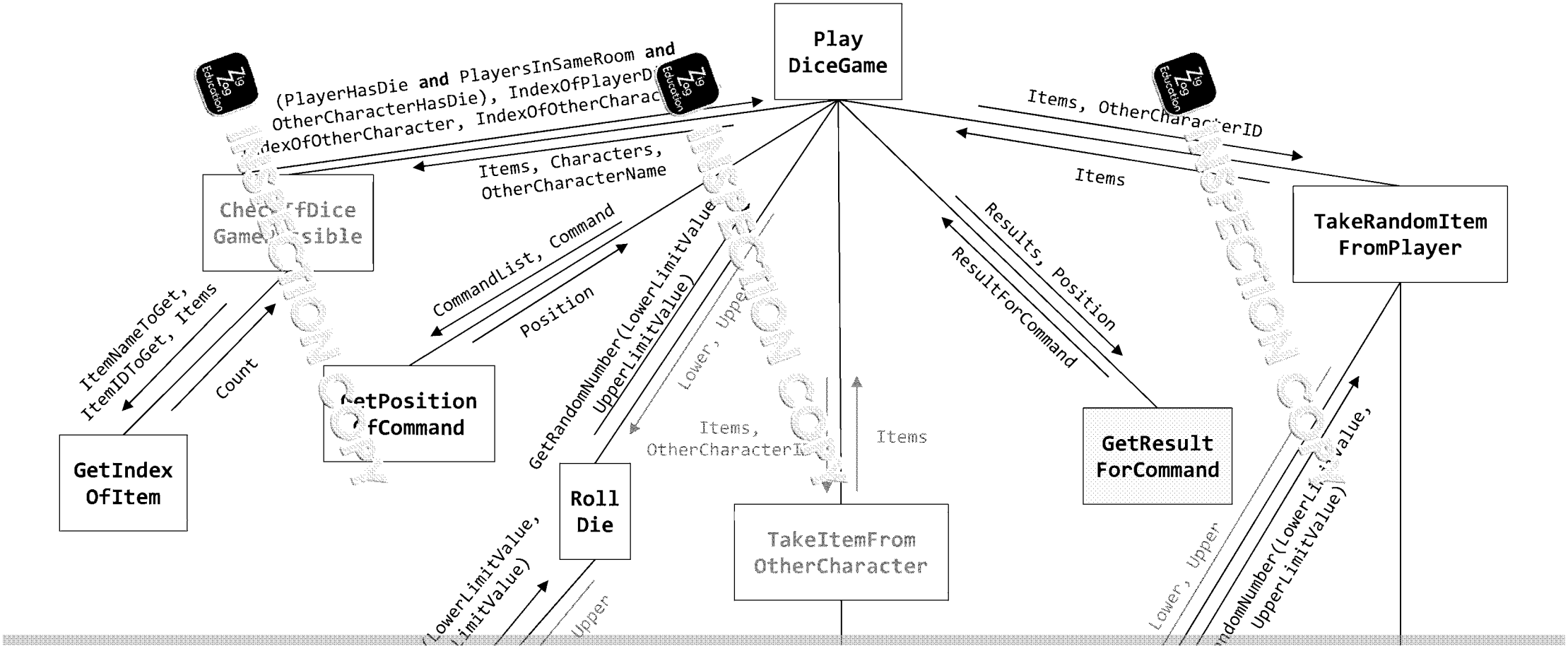
1. Add a 'restart' command that will begin the current game again from the start, either by reloading a file name
2. Modify `UseItem` to incorporate a win condition that involves using an object other than the flag
3. Randomise the location of the guard
4. In the original program, the guard's blue die 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 `TakeItemFrom` so that the player can only take the items that are listed after they have won a dice game, but can always take the die even if the guard has other items.
5. Modify `Go` so that the player will automatically try to unlock any locked doors that they encounter in their way
6. Incorporate an additional command 'look', which could be used instead of 'go' to look at the current location, assuming 'go' 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' instead of 'blue die') successfully picks 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 cannot be moved until 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' in the same room as the guard, the guard is knocked out and all items can be taken without the need to play dice
12. Incorporate 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, saving the new game file

Structure Diagram 1 (GetItem) Solution

AQA Text Adventures

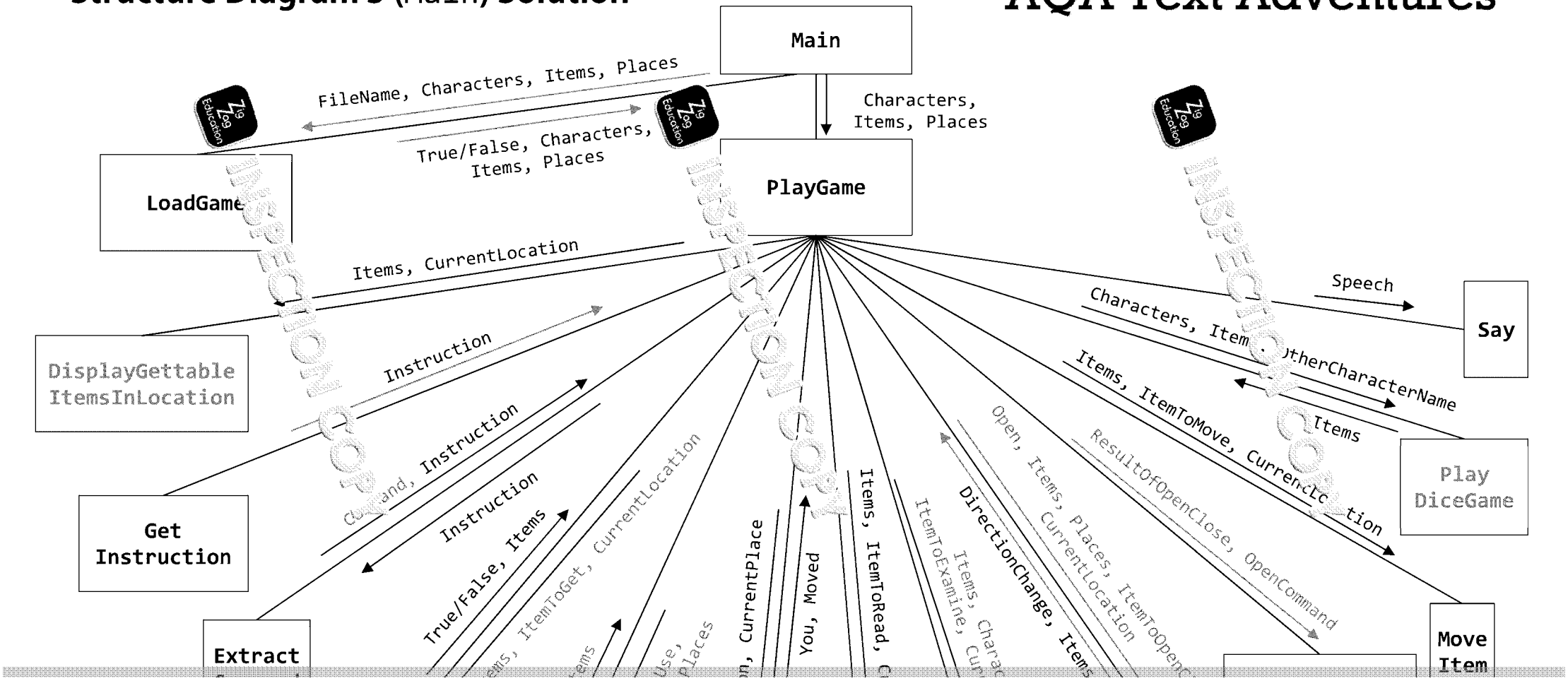


Structure Diagram 2 (PlayDiceGame) Solution



Structure Diagram 3 (Main) Solution

AQA Text Adventures



AQA Text Adventures

Written Questions (Python) - Solutions

Q	Answer/Guidance
1a	1 mark: • `LowerLimitValue` • `UpperLimitV`
1b	1 mark: • `ractCommand` • • `ExtractCommandForCommand` • `OpenClose` • `UseItem` • `GetItem`
1c	4 marks: • `print` • `len` • `input` • `index` • `append`
1d	`Character`
1e	1 mark: • `Place` • `Character` • `Item`
1f	1 mark: • `EN` • `MUM_ID_FOR_ITEM` • `ID_DIFFERENCE_FOR_OBJECT_IN_TWO_LOCATIONS`
1g	2 marks: • `ChangeStatusOfDoor` • `UseItem` • `LoadGame` Reject `Examine`, as it is a procedure and not a function.
1h	`GetInstruction`
2	2 marks: • Converts the user input to lowercase • Because subsequent comparisons will be made with lowercase strings / to from being compared with otherwise identical lo case strings
3	3 marks: • Stores whether the player ha sfully moved • If a player attempt ove, it is set to *False* • If it remain message (you are not able to go in that direction) app
4	2 mar • ol goes to the `while` loop • …To return the current value of `ResultForCommand`

Q	Answer/Guidance
5	2 marks: • (Process of) connecting/joining two (or more) strings together • Any instance of a + used between two strings, e.g. o `ListOfItems += ", ";` o `"You can't move " + ItemToMove + "."` (Does n
6	5 marks: • Prints empty string • Prints 'You are currently carr... following items:' • Loops through all ite... in the game • If an item is ... INVENTORY... • ...int ... name of that item
7	3 mark... • Temporarily holds an instance of `Character` • Attributes are set by reading data from the file • Once attributes are set it is added to the variable `Characters`
8	3 marks: • It may or may not be set to a different value in the following `if` clause • If it is not set to a value, then the following `while` loop would cause an er... `LowerLimitValue` was checked • Initialising the value to 0 prevents this error from occurring
9	3 marks (first `while` loop): • The first `while` loop checks for there being no space in the first character • If there is not a space, the first character is added to `command` • The first character is (also) discarded from `Instruction` / the `Instruc`... a substring from the second character onward • Once the loop is complete, the `Instr...` ...variable contains everythi... onward and the `command` vari... ...is the contents of the initial stri... before the first space 2 marks (second wh... loop): • ...se ... while loop checks for a space (in the first character of what ... • ...ere is a space, it is discarded • This loop removes leading spaces from `Instruction`
10	4 marks: • Loops through each item in the game • If an item is in `INVENTORY` and has 'die' in its name... • ...Sets `PlayerHasDie` to *True* • ...And sets `IndexOfPlayerDie` to the index of the item
11	4 marks: • Reference made to changing procedure `DisplayInventory` • Selection statement positioned before 'You are currently carrying the follo... • Selection statement checks if any items have the location of `INVENTORY` • If none of the items are in `INVENTORY`, display alternative message, othe... Alternative implementations potentially worth full ... would include the selectio... opposite order (i.e. checking that it is not ... pre-selection-statement instruc... items in the list then compare thi...
12	3 marks: • ...ign ... ...tement for instruction variable to be placed within a loop • ...k to be made within the loop that the length of the string is at least 2 ... • ...p to continue until that check returns *True* / length is at least 2 charact...
13	2 marks: • Execution would move to the `except` block / an exception would be thro... • The function `LoadGame` would return a value of *False* along with `Charac`...
TOTAL MARKS	

AQA Text Adventures

Programming Tasks (Python) - 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 no parameters

1 mark Valid call to `print` or `Say` with text indicated in question. The order an including the line break, although any alternative means of displaying the

```
def ShowHelp():
  print("You can enter one of the following commands:"
  print("get, use, examine, say, quit, read, move, ope
```

1 mark Additional `elif` clause in an appropriate location within `PlayGame`

```
elif Command == "help":
  ShowHelp()
```

1 mark Screenshot showing output that matches text added to `ShowHelp`

```
En ame> flag1


You are in a room with blue walls.  There is a green d
 door in the eastern wall. There is a Persian rug on t
On the floor there is: torch, red die.


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

>
```

Task 2

1 mark Additional argument in call to `Examine` from `PlayGame`

```
elif Command == "examine":
  Examine(Items, Characters, Places, Instruction, Chara
```

1 mark Additional parameter added to `Examine` definition

1 mark Addition to selection structure (`elif`) to accommodate additional edge (**A**. 'room' code in an `elif` clause, `elif` clause or `else` clause)

1 mark Current location's description displayed in appropriate part of selection s

1 mark Valid call to `DisplayGettableItemsInLocation` in appropriate p

1 mark Examining inventory and other objects remains unaffected

```
def Examine(Items, Characters, Places, ItemToExamine,
  Count = 0
  if ItemToExamine == "inventory":
    DisplayInventory(Items)
  elif ItemToExamine == "room":
    print()
    print()
    print(Places[Characters[0].CurrentLocation - 1].De
    DisplayGettableItemsInLocation(I , Characters[
  else:
    IndexOfItem = GetInd t (ItemToExamine, -1, It
```

1 mark Screenshot showing updated room description after user enters 'examin

```
Ent name> flag1


You are in a room with blue walls.  There is a green door in the no
 door in the eastern wall. There is a Persian rug on the floor.
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 no
 door in the eastern wall. There is a Persian rug on the floor.
On the floor there is: torch, red die.


>
```

Task 3

1 mark Code within "quit" `elif` clause to display prompt 'Are you sure? (y/n)' (I

1 mark User input is stored in an appropriately named variable

1 mark Selection clause that catches either 'y' or 'yes'

1 mark Selection clause catches both 'y' and 'yes' in any combination of cases, ei
lower (either done automatically by passing `GetInstruction` or manu
etc.) or by checking for every possible combination (including yES, yEs, et

1 mark Ou nd Boolean assignment inside the selection structure and break

```
elif Command == "quit":
  Say("Are you sure? (y/n)")
  response = GetInstruction()
  if response == "y" or response == "yes":
    Say("You decide to give up, try again another tim
    StopGame = True
```

1 mark Output showing 'no' resulting in a prompt, followed by 'yes' resulting in t
terminating after the output

Task 4

1 mark Attempt to convert user input to an integer within the `LowerLimitVa`
`try except` statement (**A.** if done for either `LowerLimitValue` or `U`

1 mark `LowerLimitValue` set to user input if an integer was entered (**A.** if do
`LowerLimitValue` or `UpperLimitValue`)

1 mark Loop should continue if a non-int va ntered (**A.** if done for either `L`
`UpperLimitValue`)

1 mark For a above for both `LowerLimitValue` and `UpperLi`

```
def ble(Lower, Upper):
  LowerLimitValue = 0
  if Lower.isnumeric():
    LowerLimitValue = int(Lower)
  else:
    while LowerLimitValue < 1 or LowerLimitValue > 6:
      try:
        LowerLimitValue = int(input("Enter minimum: "))
      except:
        LowerLimitValue = 0
  UpperLimitValue = 0
  if Upper.isnumeric():
    UpperLimitValue = int(Upper)
  else:
    while UpperLimitValue < LowerLimit  or UpperLimitValue
      try:
        UpperLimitValue = int(input("Enter maximum: "))
      except:
        Upp  .  ue = 0
    G  .omNumber(LowerLimitValue, UpperLimitValue)
```

1 mark Shows 'one' and '1.0' not being accepted for the minimum value, and '1'
minimum

1 mark Shows 'six' and '6.0' not being accepted for the maximum value, and '6' b

```
Enter filename> flag2

Welcome

You are in a guardroom.  There is a
 leading to a jail cell in the nort
.  There is a corridor going eastwa

> playdice guard
Enter minimum: one
Enter minimum: 1.0
Enter minimum:
Ente
Ent
Ente um: 6
You  a 3.
They rolled a 3.
Draw!

>
```

Task 5

1 mark Declaring temporary list of inventory items

1 mark Loop to add all inventory items to the temporary list

```
def DisplayInventory(Items):
  InventoryItems = []
  for Thing in Items:
    if Thing.Location == INVENTORY:
      InventoryItems.append(Thing)
```

1 mark Loop to iterate [illegible] each pass of the sort operation (x in this example

1 mark Lo[illegible]er to through each pair for individual comparisons (y in this exa

1 mark Nei[illegible] iteration would cause a comparison to be made beyond the list's

1 mark Selection structure that attempts to compare adjacent values, even if syn

1 mark Selection structure accesses the `Name` attribute of an item for the compa

1 mark Selection structure is valid, comparing appropriate name values and addr

1 mark Temporary variable used to store one of the items to be switched inside

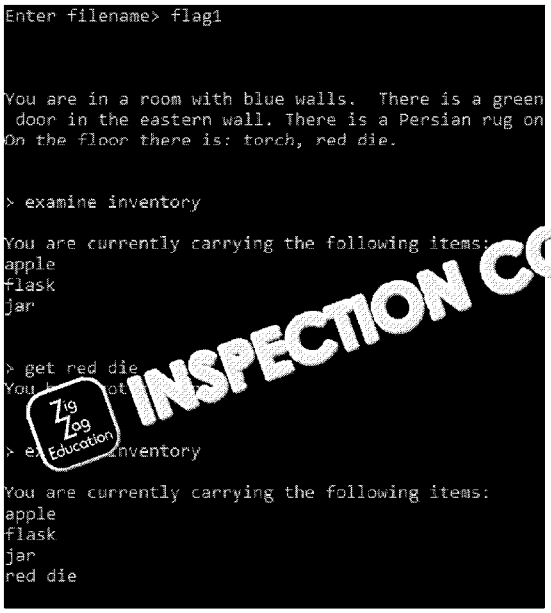
1 mark Pair of items switched inside selection structure

```
for x in range(0, len(InventoryItems) - 1):
  for y in range(0, len(InventoryItems) - 1):
    if InventoryItems[y].Name.lower() > InventoryItems
      temp = InventoryItems[y]
      InventoryItems[y] = InventoryItems[y+1]
      InventoryItems[y+1] = temp
```

1 mark Displaying items in the order the[illegible]ou[illegible]in the sorted array

```
print()
print("Yo[illegible]currently carrying the following items
f[illegible]n [illegible] InventoryItems:
    [illegible]ing.Location == INVENTORY:
    print(Thing.Name)
```

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



Task 6

1 mark Constant correctly declared and set to 4

```
INVENTORY = 1001
MINIMUM_ID_FOR_ITEM = 2001
ID_DIFFERENCE_FOR_OBJECT_IN_TWO_LOCATIONS = 10000
MAX_INVENTORY = 4
```

1 mark Variable to count inventory items

1 mark Iteration through items

1 mark Selection or equivalent to identify items in the inventory

1 mark Incrementing variable inside selection statement

```
inventorySize = 0
for Thing in Items:
  if Thing.Location == INVENTORY:
    inventorySize += 1
```

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 evaluated if inventory is f or equivalent)

```
if inventorySize >= MAX_INVENTORY:
  print("Inventory full")
  CanGet = False
```

1 mark Implementing inventory size check in both `GetItem` and `TakeIt`

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

```
inventorySize = 0
for Thing in Items:
  if Thing.Location == INVENTORY:
    inventorySize += 1
if inventorySize >= MAX_INVENTORY:
  print("Inventory full")
elif ChosenItem in ListOfNamesOfItemsInInventory:
  print("You have that now.")
  Pos = ListOfNamesOfItemsInInventory.index(ChosenItem)
  Items = ChangeLocationOfItem(Items, ListofIndicesOfItemsI
else:
  print("They don't have that item, so you don't take anyt
```

1 mark First screenshot shows torch being picked up, red die not being picked up accordingly

```
Enter filename> flag1



You are in a room with blue walls.  There is a green d
 door in the eastern wall. There is a Persian rug on t
On the floor there is: torch, red die.


> get torch
You have that now.

> get red die
Inventory full


> examine inventory

You are currently carrying the following items:
flask
apple
jar
torch
```

1 mark Second screenshot shows dice game being played, jar won, jar not being accordingly

```
Enter filename> flag2



You are in a guardroom.  There is a metal silver door
 leading to a jail cell in the northern wall.  There i
.  There is a corridor going eastwards.


> playdice guard
Enter minimum: 6
Enter maximum: 6
You rolled a 6.
They rolled a 5.
You win!
Which item do you want to take?  They have:jar, gold k
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 th

1 mark Selection statement to check whether the item being dropped is in the in

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

1 mark If item is in inventory, locati... changed to `CurrentLocation`

1 mark If item is in inve..., correct message displayed

```
def DropItem(Items, ItemToDrop, CurrentLocation):
    IndexOfItem = GetIndexOfItem(ItemToDrop, -1, Items)
    if (not(Items[IndexOfItem].Location == INVENTORY)):
        print("You are not carrying a " + ItemToDrop)
    else:
        ChangeLocationOfItem(Items, IndexOfItem, CurrentLo
        print(ItemToDrop + " dropped")
    return Items
```

1 mark Selection structure of `PlayGame` incorporates 'drop' command

1 mark Parameters correctly passed to `DropItem`

```
elif Command == "drop":
    Items = DropItem(Items, Instruction, Characters[0].C
```

1 mark Screenshot shows 'flask dropped' and 'yo... carrying a flask' as we

```
Enter filename> flag1


You ... in a room with blue walls.  There is a green d
 do... the eastern wall. There is a Persian rug on t
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 follo... tems:
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 check for '.gme' within file name

1 mark '.gme' added to filename if not already within the string

1 mark Successfully loading a file ensures loop would not run again

```
GameLoaded = False
while not GameLoaded:
  Filename = input("Enter filename or enter Q to quit> ")
  if Filename == "Q":
      break
  if ".gme" not in Filename:
      Filename += ".gme"
  print()
  GameLoaded, Characters, Items, Places = LoadGame(Filename,
  if GameLoaded:
    PlayGame(Characters, Items, Places)
  else:
    print("Unable to load game.")
```

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

1 mark 'flag2.gme' causes game to load

```
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.  There is a metal silver
 leading to a jail cell in the northern wall.  T
.  There is a corridor going eastwards.


>
```

Task 9

1 mark `DisplayCharacters` correctly declared with one parameter

1 mark Header row contains names of all fields as appropriate

1 mark Spacing between headers is correct

1 mark Loop to iterate through each element of Characters

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

1 mark All four attributes displayed in a line, with no additional text

1 mark Spacing will be correct regardless of size of a field

```
def DisplayCharacters(Characters):
    Header = ""
    Header += "%-5s" % ("ID",)
    Header += "%-12s" % ("NAME",)
    Header += "%-64s" % ("DESCRIPTION",)
    Header += "CURRENTLOCATION"
    print(Header)
    for Character in Characters:
        Line = ""
        Line += "%-5s" % Character.ID
        Line += "%-12s" % Character.Name
        Line += "%-64s" % Character.Description
        Line += "%d" % Character.CurrentLocation
        print(Line)
```

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

1 mark Valid call to `DisplayCharacters` from `PlayGame`

```
elif Command == "debugchar":
    DisplayCharacters(Characters)
```

1 mark Two characters displayed, lined up correctly in columns in the correct orc

```
Enter filename> flag1


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


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

>
```

Task 10

1 mark `FillVessel` correctly declared with four parameters

1 mark Variable, or equivalent, to store whether the item is a container

1 mark Variable, or equivalent, to store whether the item is large *(equivalence in 'IsContainer' is set to true only if the item is both a container and not conditions to share a single variable, and all applicable marks can still be*

1 mark Variable, or equivalent, to store whether the item is in the player's invent

1 mark Variable, or equivalent, to store whether the player is in the same locatio

1 mark Loop to examine items in turn (**A.** multiple loops, calls to `GetIndexOfI`

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 inventc

1 mark Selection statement to check whether the player and the barrel are in the

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

```
def FillVessel(Characters, Items, Places, Vessel):
  IsContainer = False
  InInventory = False
  ByBarrel = False
  Fillable = None
  for Thing in Items:
    if Vessel == Thing.Name and "container" in Thing.Status and
      IsContainer = True
      Fillable = Thing
    if Vessel == Thing.Name and Thing.Location == INVENTORY:
      InInventory = True
    if Thing.Name == "barrel" and Thing.Location == Characters[0
      ByBarrel = True
```

1 mark Selection statement to check that all conditions are met

1 mark Inside selection statement, append ' of water' to the vessel's name (**A.** ar the item or call to `GetIndexOfItem`; this example code uses a variable above)

1 mark 'You have filled the ' with name of vessel appended is displayed if the ves

1 mark 'You cannot do that' displayed if the 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` is no

1 mark 'Already full' displayed under the correct circumstances

```
if IsContainer and InInventory and ByBarrel and " of water" no
  Fillable.Name = Vessel + " of water"
  print("You have filled the " + Vessel)
elif Fillable != None and " of water" in Fillable.Name:
  print("Already full")
else:
  print("You cannot do that")
```

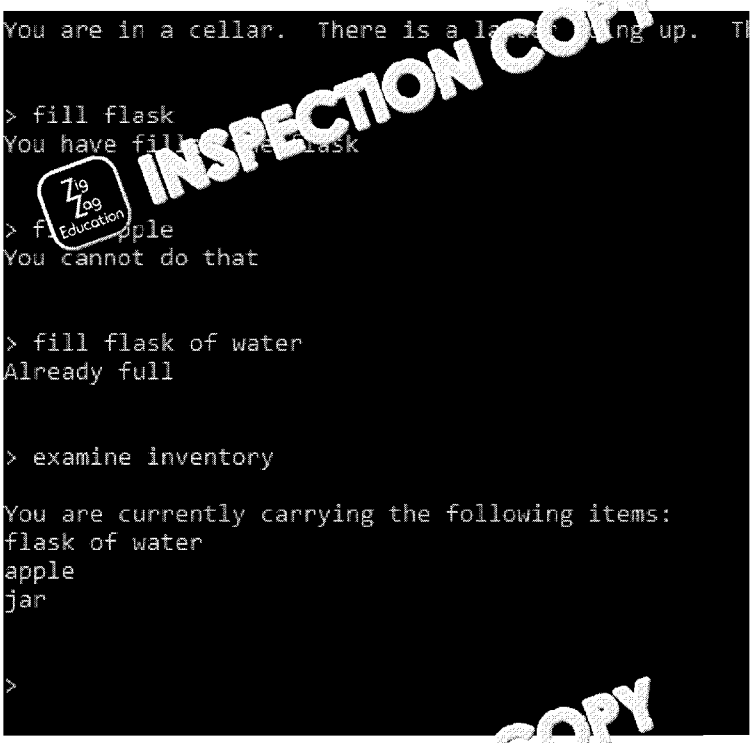
1 mark Selection structure in `PlayGame` adjusted to include 'fill' command

1 mark Valid call to `FillVessel` at valid point in selection structure

```
elif Command == "fill":
    FillVessel(Characters, Items, Places, Instruction)
```

1 mark Input of 'fill flask' displays 'You have filled the flask' and 'fill apple' displa

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



Task 11

1 mark `History` class correctly declared

1 mark Integer variable `Pointer` and integer array `Locations` correctly decl

1 mark `Pointer` and `Locations` initialised to *-1* and 5 elements respectively be declared as list of 5 `None` elements as long as `Push` doesn't allow for more than 5 elements, and Locations hasn't b alised with any oth

```
class History():
  def __init__(self
    self.Poi
    lf. ons = [None] * 5
```

1 mark `Push` correctly declared within the `History` class with two parameters

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

```
def Push(self, NewLocation):
  if self.Pointer < 4:
    self.Pointer += 1
    self.Locations[self.Pointer] = NewLocation
  else:
    for x in range(0, 3):
      self.Locations[x] = s lf.Locations [x+1]
    self.Locations[ ] = N location
```

1 mark Pop ec ed within the `History` class with one parameter

1 mark Ch 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

```
def Pop(self):
  if self.Pointer > -1:
    self.Pointer -= 1
    return self.Locations[self.Pointer + 1]
  else:
    return -1
```

1 mark `Past` declared as an instance of `History` in y ame

```
def PlayGame(Characters , s, Places):
  Past = History()
  StopGame =
  M d =
```

1 mark Pas stance passed to `Go` when 'go' command entered by user (**A.** an

```
elif Command == "go":
  Characters[0], Moved = Go(Characters[0], Instruction
  Places[Characters[0].CurrentLocation - 1], Past)
```

1 mark Declaration of `Go` now includes a parameter of type `History` in same p

```
def Go(You, Direction, CurrentPlace, Past):
```

1 mark Attempt to call `Push` within at least one `if` clause (even if unsuccessful)

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

```
if Direction == "north":
  if CurrentPlace.North != 0:
    Moved = True
    Past.Push(You.CurrentLocation)
    You.CurrentLocation = CurrentPlace.North
```

1 mark New `if` 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*

```
elif Direction == "back":
    NewLocation = Past.Pop()
    if NewLocation != -1:
      You.CurrentLocation = NewLocation
    else:
      print("You cannot go back")
```

1 mark Having gone E[illegible] back, the original room description should re-disp se[illegible] result in 'You cannot go back' being displayed.

```
Enter name> flag2


You are in a guardroom.  There is a metal silver door leading to a jail ce
 leading to a jail cell in the northern wall.  There is a clear desk with
.  There is a corridor going eastwards.


> go east


You are in a long, empty corridor.  There is a yellow door in the eastern


> go back


You are in a guardroom.  There is a metal silver door leading to a jail ce
 leading to a jail cell in the northern wall.  There is a clear desk with
.  There is a corridor going eastwards.


> go back
You cannot go back


You are in a guardroom.  There is a metal silver door leading to a jail ce
 leading to a jail cell in the northern wall.  There is a clear desk with
.  There is a corridor going eastwards.


>
```

Task 12

1 mark Addition of `Health` attribute to `Character` class

1 mark Characters given `Health` of 10 by default

```
class Character():
  def __init__(self):
    self.Name = self.Description = ""
    self.ID = self.CurrentLocation = 0
    self.Health = 10
```

1 mark `EatItem` correctly declared with three parameters

1 mark Index of item required by iteration or call to `GetIndexOfItem`

1 mark `ItemToEat` 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 wr

1 mark Output indicates that the item was eaten and new `Health` value display

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

```
def EatItem(Characters, Items, ItemToEat):
  ItemIndex = GetIndexOfItem(ItemToEat, -1, Items)
  if ItemIndex > -1:
    if "edible" in Items[ItemIndex].Status and Items[ItemInd
      Characters[0].Health += 5
      del Items[ItemIndex]
      print("You have eaten the " + ItemToEat)
      print("Your health is %d" % Characters[0].Health)
    else:
      print("You cannot eat that")
  else:
    print("You cannot eat that")
```

1 mark `Health` displayed after call to `DisplayGettableItemsInLocati`

```
DisplayGettableItemsInLocation(Items, Characters[0].Cu
print("Your health is %d" % Characters[0].Health)
Moved = False
```

1 mark Valid call to `EatItem` within `PlayGame`

```
elif Command == "eat":
  EatItem(Characters, Items, Instruction)
```

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

```
Enter filename> flag1


You are in a room with blu
 door in the eastern
On the floor th
Your h
> ea
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**. appropriate `break` or

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 re

1 mark Output 'They don't have a...' correct i cl din oncatenation with `Chos`

```
ItemTaken = False
while not Ite
  C =
  "Which item do you want to take?  They have:", end =
  (ListOfNamesOfItemsInInventory[0], end = "")
  while Count < len(ListOfNamesOfItemsInInventory) - 1:
    print(",", ListOfNamesOfItemsInInventory[Count], end = "")
    Count += 1
  print(".")
  ChosenItem = input()
  if ChosenItem in ListOfNamesOfItemsInInventory:
    ItemTaken = True
    print("You have that now.")
    Pos = ListOfNamesOfItemsInInventory.index(ChosenItem)
    Items = ChangeLocationOfItem(Items, ListofIndicesOfItemsI
  else:
    print("They don't have a %s" % ChosenItem + " - try again
```

1 mark Attempt to take a box answered with 'the ave a box - try again';
with 'You have that now'.

```
Enter filename> f
You  in a guardroom.  There is a metal silver door
 leading to a jail cell in the northern wall.  There i
.  There is a corridor going eastwards.


> playdice guard
Enter minimum: 6
Enter maximum: 6
You rolled a 6.
They rolled a 3.
You win!
Which item do you want to take?  They have:jar, gold k
box
They don't have a box - try again.
Which item do you want to take?  They have:jar, gold 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 `PlayerHasDie` is *false*

1 mark '_____ is not here' displayed only if `Player` `aleRoom` is *false*, to i

1 mark '_____ has no die' displayed onl `herCharacterHasDie` is *false* previous mark was den a d failure in concatenation)

```
if not Pl      ie:
    t("Player has no die")
eli  PlayersInSameRoom:
    print(OtherCharacterName + " is not here")
elif not OtherCharacterHasDie:
    print(OtherCharacterName + " has no die")
return PlayerHasDie and PlayersInSameRoom and OtherCharacterHa
IndexOfPlayerDie, IndexOfOtherCharacter, IndexOfOtherCharacter
```

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

```
Enter filename> flag1


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

> playdice guard
Player has no die
You can't play
> g      die
You      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 inver

1 mark Descriptions, but not names, are displayed in round brackets

```
def DisplayInventory(Items):
  print()
  print("You are currently carrying the following items:")
  for Thing in Items:
    if Thing.Location == INVENTORY:
      Line = "%-10s" % (Thing.Name,)
      Line += " ("
      Line += Thing.Description
      Line += ")"
      print(Line)
```

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 t
 door in the eastern wall. There is a Persian rug on the floor
On the floor there is: torch, red die.


> examine inventory

You are currently carrying the following items:
flask      (It is an empty plastic flask.)
apple      (It is an apple.  It looks fresh.)
jar        (It is an empty glass 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)	
	(h)	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		

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	