Murder Mystery

Project Plan

Name:Ming Shern Siew

Table of Contents

How to Play Murder Mystery

"You are an up and coming Detective out to solve homicide. It's your job to interview the suspects and examine the scene for evidence, discover the murder weapon and where the murder took place ... all before the killer can strike again!"

As you walk into new crime scene, you remember steps to find real killer:

- The killer is among the suspect around crime scene.
- In a single game turn, you will only be able to execute an action such as search, go to location, collect item etc.
- You will have to find murder weapon in any of the location.
- You will have to question each suspect in same residence at the murder scene and you may encounter following suspects:
    - Two pair of suspects will alibi each other.
    - A suspect with no alibi but not a murderer.
    - Murderer will have a false alibi.
- Once you knows whose the murderer, you will have to gather all suspect into crime scene and accuse them.
- If your time limit run out or wrongly accuse killer and did not found murder weapon or wrongly accuse suspect as killer but correct murder weapon or correctly accuse suspect as killer but wrong murder weapon, you will lose.
- If you correctly accuse suspect as killer and correct murder weapon and time limit is still running, you will be able to close case and win.

"You took a deep breath and smile, start walking into crime scene and investigate with all yours's might to stop killer!"

The above is my introduction story for my Murder Mystery game. This will be stored in this text file – `murderMysteryRules.txt` and is loaded at the start of the game for the detective to read.

Outline the Functionality of All Game Classes

After reading the assignment brief, I have decided to use 3 classes in my game: A Suspect class, An Item Class, and the Application file.
Class helps to keep track of multiple object which will be used in game and reduce repetitive code which will be used when multiple object will be used.

The **Suspect class** must be able to do the following:

- have a unique name, description, an alibi and a starting location.
- have status as the detective, victim, murderer, or a suspect.
- set the suspect's starting location and display their status.
- update and display the suspect's statistics (stats) during the game.
- collect items (evidence) and store it in an inventory.

The **Item class** must be able to do the following:

- have a unique name, descriptions, and a starting location
- is either the murder weapon (has traces of blood on it) or not?
- can be picked up and carried by the detective

The **Locations Class** must be able to do the following:

- have a unique name, one or more suspect, and item.
- is either the crime scene location (has traces of blood on it) or not?
- coordinate(X, Y) which are unique, each location will be next to each other and all location together will form 2-D array.
- access and display data about a suspect or an item if it is in the location.
- be loaded from a text file and stored appropriately at the start of the game
- display all appropriate information when the detective enters it (name or number, description, exits, other information if required)

The **Status <<Enumeration>>** must be able to do the following:

- Hold enum value of {DETECTIVE, SUSPECT, MURDERER, VICTIM}
- Indicate type of status for suspect.

The **Difficulty <<Enumeration>>** must be able to do the following:

- Hold enum value of {NOVICE=4, SENIOR=5, ELITE=6}
- Indicate type of difficulty for game.

The **Application files** must do the following:

- display the “how to play” information at the start of the game.
- assign a name for detective which can be requested at the start of the game and used in the feedback given.
- initialise the detective, suspects, items, and locations as required by the brief.
- detective and suspect will share same attribute and behaviour which both origin from Suspect class, but detective does not have alibi which is same as suspect not having alibi and can be initialise with null without negatively affect the game.
- display an appropriate and uncluttered user interface providing relevant information to the detective.
- allow detective to use two-word commands: GO location, SEARCH location, USE item, GET item, DROP item, EXAMINE item, QUESTION suspect, and ACCUSE suspect
- allow detective to use one-word commands: GATHER, MAP, HELP and QUIT
- display the information about each location as it is entered (name, exits, etc.)
- terminate the game when the end game is triggered (detective wins or loses)
- provide detective stats at the end of the game (success or failure, inventory, etc.)
- the detective should be able to QUIT the game at any time.

The Game Setup

- Load and display the **murderMysteryRules.txt** and include a “Press any key to continue”
  - Display an overview of the game rule so the detective knows what to do to win.
- Initialise the game elements:
  - add the detective – ask for the detective’s name which will be created using the Suspect class, set default variables
  - all the other things that will happen during initialisation including
    - Ask the detective to set a skill level which will change the number of locations, the number of suspects and items placed in the world.
      - Declare `enum Difficulty {NOVICE=4, SENIOR=5, ELITE=6}`
      - First, construct Map = Location[M][M-1] where M is a number between 4 to 6 inclusive. [M-1] is chosen to make game slightly easier since it will be complicated otherwise.
      - Secondly, calculate the total number of suspects in a map = M + (M-1). Then, randomly assigned a murderer and a victim among suspects. Both murderer and victim cannot be same person.
      - Lastly, calculate the total number of items in a map = random number between inclusively M and M*(M-1). Then, randomly assigned one of these items as murderer item and randomly distribute all item in different location on map.
    - Initialise time limit with cubic of random number between 3-5 inclusively.
    - Reset all game variables back to their default values to ready to start the game (see UML diagram for a list of variables).
    - Clear the screen and display the title of the game.

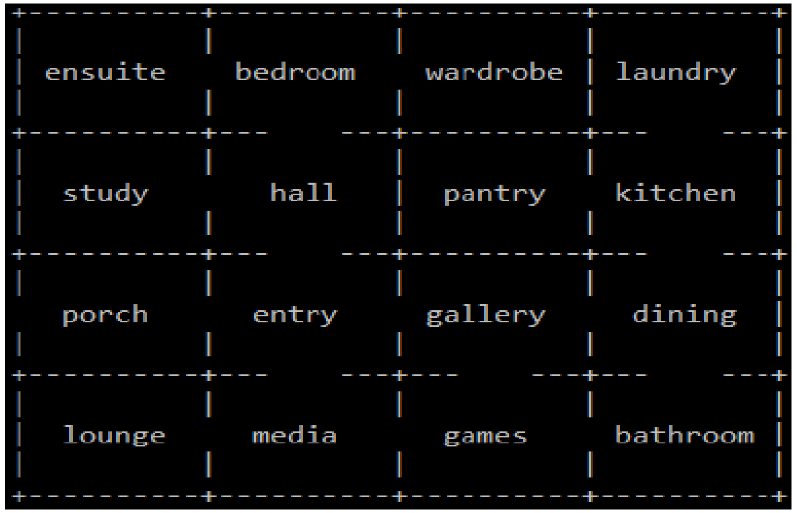
The Detective’s turn

Detective input MAP

```
Turn# 1
------------------------------------------
Detective: Ming
Detective Location: entry
Murder Weapon: None
Murder Weapon Location: None
------------------------------------------
```

Command: MAP

```
------------------------------------------
A map will be displayed which shows
building and detective’s location.
------------------------------------------
```



```
+-----------+-----------+-----------+-----------+
|           |           |           |           |
| ensuite     bedroom     wardrobe  | laundry   |
|           |           |           |           |
+-----------+---     ---+-----------+---     ---+
|           |           |           |           |
|   study        hall   |  pantry     kitchen   |
|           |           |           |           |
+-----------+---     ---+-----------+---     ---+
|           |           |           |           |
    porch       entry      gallery      dining  |
|           |           |           |           |
+-----------+---     ---+---     ---+---     ---+
|           |           |           |           |
|   lounge      media       games     bathroom  |
|           |           |           |           |
+-----------+-----------+-----------+-----------+
Press any key to continue …
```

Detective input GOTO <location>

```
Turn# 1
------------------------------------------
Detective: Ming
Detective Location: entry
Murder Weapon: None
Murder Weapon Location: None
------------------------------------------
Command: GOTO HALL
------------------------------------------
Detective will be relocated to Hall.
------------------------------------------

Turn# 2
------------------------------------------
Detective: Ming
Detective Location: hall
Murder Weapon: None
Murder Weapon Location: None
------------------------------------------
Command:
```

Detective Feedback

In this game, the detectives type a command as an action. Thus, there are no individual turn and only detective determine what action to execute in a turn. Therefore, the main game loop will consist mainly of:

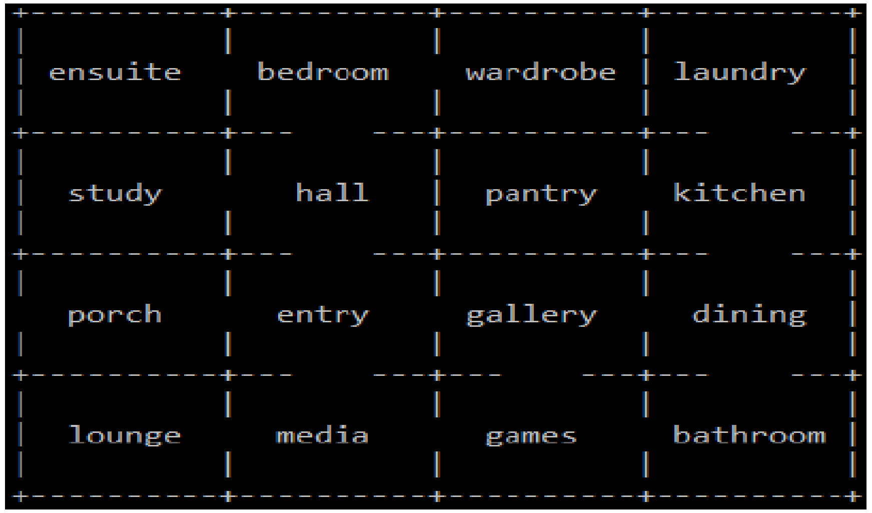
◆Displaying the progress information in the game and get input from detective.

```
Turn# 1
----------------------------------------------
Detective: Ming
Detective Location: entry
Murder Weapon: None
Murder Weapon Location: None
----------------------------------------------
```

Command:

◆Displaying the result for command made by detective in a turn.

```
----------------------------------------------
A map will be displayed which
shows building and detective’s location.
----------------------------------------------
```



```
+----------+----------+----------+----------+
|          |          |          |          |
| ensuite     bedroom    wardrobe | laundry  |
|          |          |          |          |
+----------+---    ---+----------+---    ---+
|          |          |          |          |
|  study        hall  |  pantry    kitchen  |
|          |          |          |          |
+----------+---    ---+----------+---    ---+
|          |          |          |          |
   porch       entry     gallery     dining |
|          |          |          |          |
+----------+---    ---+---    ---+---    ---+
|          |          |          |          |
|  lounge      media      games     bathroom |
|          |          |          |          |
+----------+----------+----------+----------+
```

```
Press any key to continue …
```

- If a detective has won – if so, end the game, otherwise play the new crime case and the game continues.

```
----------------------------------------------
Inventory: Hammer, Knife, Spoon, Fork
Detective Ming arrested Pinto for murder!
Detective Ming has successfully to stop murder.
Pinto killed Vince with Hammer at Garden.
----------------------------------------------

Do you want to play again? (y/n)
```

- If a detective has lost – if so, end the game, otherwise play the new crime case and the game continues.

```
----------------------------------------------
Inventory: Hammer, Knife, Spoon, Fork
Detective Ming arrested Pinto for murder!
Detective Ming has failed to stop murder.
Mark killed Vince with Hammer at Garden.
----------------------------------------------

Do you want to play again? (y/n)
```

Processing Detective Input

I will include three ways in which to process the detective's input.

- Asking for a string input – used when asking the detective for their name and command line such as GO GARDEN, DROP ITEM etc.
- Asking for a single letter input – used when asking if they want to play again and indicate the type of question to be asked to suspect.

All of these are separate functions which use the same structure:

```
    string askForString(string question);
    char askForLetter(string question);
```

Based on the detective's response, the game will perform the required action:

- The detective's name will be sent to the Suspect class when creating a new detective.
- The command line will be used Application file to make changes to detective such as moving character to new location.
    - processCommandInput() function in Application file will takes in command as string and execute the command to perform changes in games.
- The letter will determine whether the program exits or the game resets and starts again.

List of command:

GO <location>
- Relocate detective to location provided in command.

SEARCH <location>
- Detective will search for item found in location and display in command the item found in location.

USE <item>
- Detective will use item found in location and the item may be able to unlock a location which is locked.

GET <item>
- Item will be added to detective inventory

DROP <item>
- Item will be removes from detective inventory and added to location which detective is currently in.

EXAMINE <item>
- Display description of item which includes name of item, is there a blood in it etc.

QUESTION <suspect>
- Display series of question that can be asked to suspect and prompt user to choose one of the questions.
- The question will appear in this form:
    - [a] where were you at time of murder?
    - [b] what is your relationship with victim?
    - ……
- The character will indicate the type of question to be asked and will be feed into the responseQuestion() method in suspect class.
- Once user prompt a given question, display response to the question from suspect with responseQuestion() method in suspect class.

ACCUSE <suspect>
- End the game if a valid input for suspect is given
- If detective found murder weapon and accuse the correct killer, detective will be able to close case and win game. Otherwise, detective will lose when either one of the criteria is not met.

GATHER
- All suspect and detective in a map will be relocated to crime scene.

MAP
- A 2-D array of location will be shown to user.

HELP
- It will display all information from murderMysteryRules.txt

QUIT
- End the game without announcing winner or loser.
- The detective will be navigated to main menu.

The End Game Conditions

This game has only two end game condition:

1) Win condition:
   i. Found murder weapon and accuse the correct killer – killer will have false alibi which detective needs to find out.
   ii. Time limit is more than 0 min.
2) Lose condition:
   i. Did not found murder weapon and accuse wrong killer.
   ii. Did not found murder weapon but accuse correct killer.
   iii. Found murder weapon but accuse wrong killer.
   iv. Time limit reach 0 min or less.
3) Quit condition: detective typed QUIT command which will also end the game without rewards or stats given to detective.

- After every turn the `checkGameOver()` function is called to determine if the end game condition has been met.
- If `checkGameOver()is true`, then the `gameOver()` function displays the result of crime case, and the detective is asked if they want to play again.
- If `checkGameOver()is false`, then the next game turn is run until checkGameOver() function return true.

Additional Features Included

I have decided to include the following 2 extra features in my game:

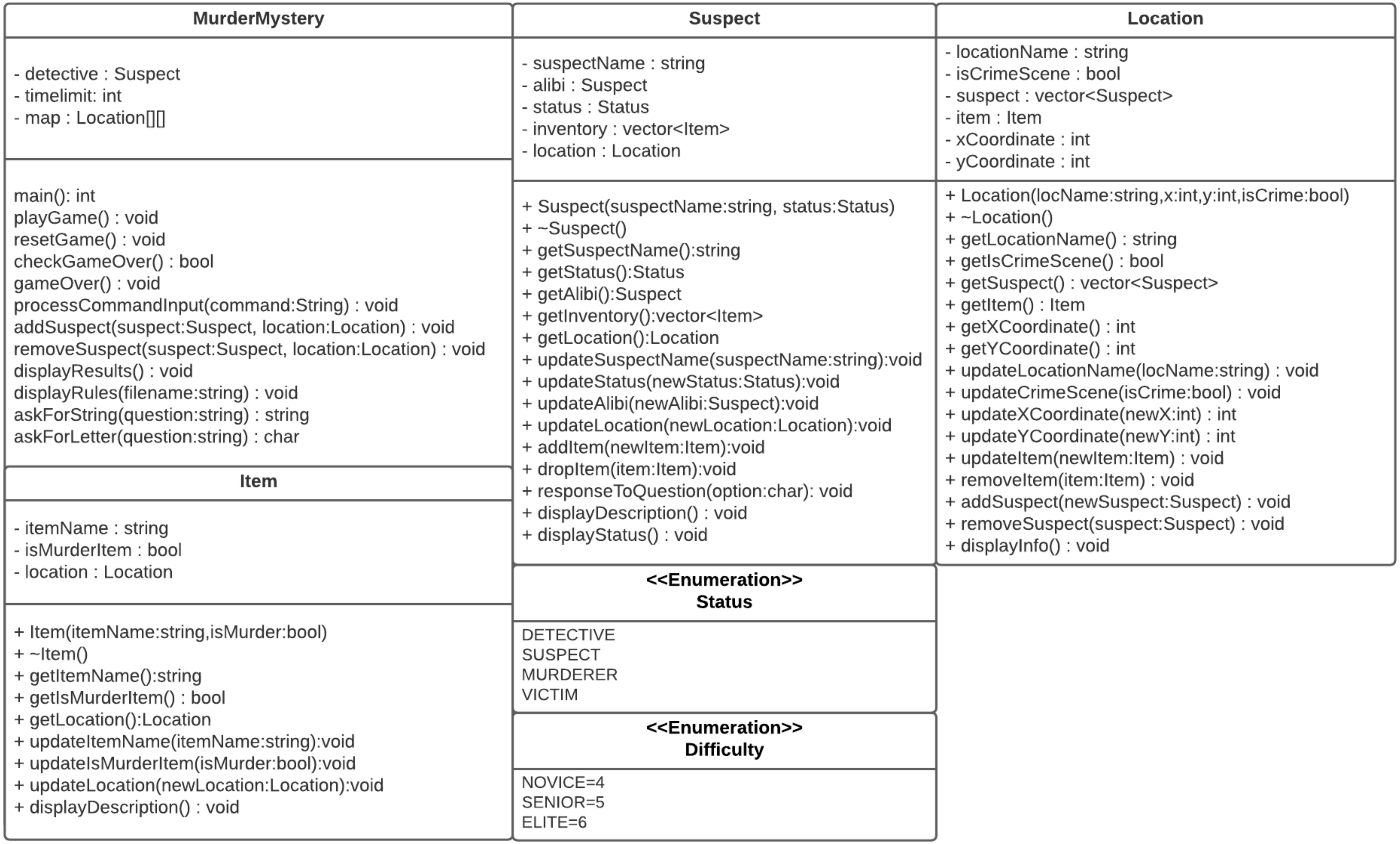
- Adding a detective skill level – I have indicated what this does in the **Game Setup** section.
  The detective can select a skill level which modifies the game parameters – the number of locations, the number of suspects and items placed in the world.
  Approach:
  Construct Map = Location[M][M-1] where M is Integer of Enum string. [M-1] is chosen to make game slightly easier since it will be complicated otherwise.
  The total number of suspects in a map = M + (M-1). Then, randomly assigned a murderer and a victim among suspects. Both murderer and victim cannot be same person.
  The total number of items in a map = random number between inclusively M and M*(M-1). Then, randomly assigned one of these items as murderer item and randomly distribute all item in different location on map.
  Added game difficulty changes at **Game Setup** section.

  The approach above are made based on self-made algorithm to fulfill requirement where lowest difficulty must have 7 suspects, a map must have at least 10 locations. The map is essentially 2-D array of location and number of rows*number of columns will determine number of locations in a game. Thus, it will be feasible to form a map consisting of two numbers to represent number of rows and number of columns. Enumeration is used to ensure a constraint set of value for difficulty of the game where possible values are `NOVICE, SENIOR, ELITE` and will not accept other invalid value. Although this approach is based ad-hoc method, but it has benefit to use variable M which developer can reduce amount of repeat code and hard code.

- The detective has a time limit in which to solve the murder where each move the detective makes reduces the time by 3-5 minutes. The detective loses if they run out of turns.
  Approach:
  Let N = random number between 3 to 5 inclusive.
  Time limit = N^3. Thus, the time limit of a game will be N^3 (For example, 3^3 = 27 min,….)
  Each time detective commands any action except for MAP and HELP will consume a turn and will reduce the time limit by N.
  Added time limit initialisation at **Game Setup** section.
  Added win condition where time limit must be more than 0 min and correct murder weapon and correctly accuse killer.
  Added lose condition where detective will lose when time limit reach 0 min or less.

  The approach above are made based on self-made algorithm which fulfill requirement where time limit is to reduce by 3-5 minutes. The intuition behind Time limit of N^3 is to allow user to have N^2 time to play the game after N (each move takes this time) is divided by N^3. This approach is based ad-hoc but may improve later in programming phase after testing the games or new method found which can improve games experience for user. This approach also has benefit to use variable N which developer can reduce amount of repeat code and hard code.

UML Class Diagrams



```
MurderMystery
-------------------------------------------------
- detective : Suspect
- timelimit: int
- map : Location[][]
-------------------------------------------------
main(): int
playGame() : void
resetGame() : void
checkGameOver() : bool
gameOver() : void
processCommandInput(command:String) : void
addSuspect(suspect:Suspect, location:Location) : void
removeSuspect(suspect:Suspect, location:Location) : void
displayResults() : void
displayRules(filename:string) : void
askForString(question:string) : string
askForLetter(question:string) : char
```

```
Item
-------------------------------------------------
- itemName : string
- isMurderItem : bool
- location : Location
-------------------------------------------------
+ Item(itemName:string,isMurder:bool)
+ ~Item()
+ getItemName():string
+ getIsMurderItem() : bool
+ getLocation():Location
+ updateItemName(itemName:string):void
+ updateIsMurderItem(isMurder:bool):void
+ updateLocation(newLocation:Location):void
+ displayDescription() : void
```

```
Suspect
-------------------------------------------------
- suspectName : string
- alibi : Suspect
- status : Status
- inventory : vector<Item>
- location : Location
-------------------------------------------------
+ Suspect(suspectName:string, status:Status)
+ ~Suspect()
+ getSuspectName():string
+ getStatus():Status
+ getAlibi():Suspect
+ getInventory():vector<Item>
+ getLocation():Location
+ updateSuspectName(suspectName:string):void
+ updateStatus(newStatus:Status):void
+ updateAlibi(newAlibi:Suspect):void
+ updateLocation(newLocation:Location):void
+ addItem(newItem:Item):void
+ dropItem(item:Item):void
+ responseToQuestion(option:char): void
+ displayDescription() : void
+ displayStatus() : void
```

```
<<Enumeration>>
Status
-------------------------------------------------
DETECTIVE
SUSPECT
MURDERER
VICTIM
```

```
<<Enumeration>>
Difficulty
-------------------------------------------------
NOVICE=4
SENIOR=5
ELITE=6
```

```
Location
-------------------------------------------------
- locationName : string
- isCrimeScene : bool
- suspect : vector<Suspect>
- item : Item
- xCoordinate : int
- yCoordinate : int
-------------------------------------------------
+ Location(locName:string,x:int,y:int,isCrime:bool)
+ ~Location()
+ getLocationName() : string
+ getIsCrimeScene() : bool
+ getSuspect() : vector<Suspect>
+ getItem() : Item
+ getXCoordinate() : int
+ getYCoordinate() : int
+ updateLocationName(locName:string) : void
+ updateCrimeScene(isCrime:bool) : void
+ updateXCoordinate(newX:int) : int
+ updateYCoordinate(newY:int) : int
+ updateItem(newItem:Item) : void
+ removeItem(item:Item) : void
+ addSuspect(newSuspect:Suspect) : void
+ removeSuspect(suspect:Suspect) : void
+ displayInfo() : void
```