**Text Adventure Document**

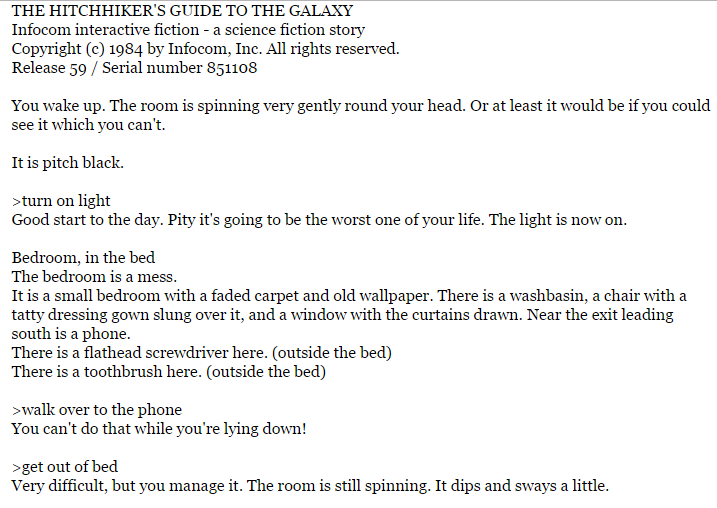
For this assignment, we were asked to create a standard text adventure game, using the source code that was provided to us. We had to implement ourselves an item class while developing a suitable input algorithm to handle commands such as search a room, walk to another location, check inventory, take an item, and use an item. Within this document, I will be covering the development process of my algorithms, the choices I made followed by my research.

**Research**

Before I started the development of my text adventure game, I decided to do some research into some well-known text adventure games to help me figure out how the core of a text adventure game works, as well to gather some inspiration for my own.

The first text adventure game I researched and played was called Zork, which was developed and published by Intercom in the 1980’s for the commondore 64, DOS and many more. The core objective of Zork is for the player to collect the twenty treasures of Zork and place them into a trophy case, but to do so the player must overcome a variety of puzzles, which can lead to player’s death if they are not successful. What was unique about Zork was its input commands, the player is not restricted to verb-noun commands such as take lamp or open mailbox, instead players had the options to feel more immersed into the adventure and were able to input more complicated sentences. For example look under the rug. Although when playing Zork for myself, I notice that the you could input common verbs such as take, open and close.

The second text adventure game I researched and played was The Hitchhiker’s guide to the galaxy, which was developed and published by Infocom Games in 1984. The core objective of The Hitchhiker’s guide to the galaxy is for the player to survive playing as Arthur Dent, the main protagonist, who is an ordinary human being, who must survive a series of inter-galactic misadventures. When playing The Hitchhiker’s Guide to The Galaxy, I noticed that it was more heavily narrative driven than Zork and that the input commands are more restricted to what is happening within the current location. An example of this would be the opening scene where the player wakes up and was strict on what they can input specifically. As seen in the screenshot below, you simply cannot leave your bed without the player having to inform the game that they are getting out of bed.



**Input algorithm & Item Class**

After researching into some text adventure games, and getting an idea of how one should work, I considered many ways to handle my input commands. Looking back on my research, I really liked the concept of using sophisticated sentences very much like how Zork has, for my input commands to allow player to feel more immersed within my text adventure game.

With this thought in mind, I researched into ways of possibly creating a String function that was made up of common phrases that the console would recognise if the player was to input them as a command. My research lead me to discovering a function called a Switch Statement, which is a like a set of if statements known as case function, a list of possibilities each with their own action and outcome. Sometimes a Switch statement can be accompanied with a Default action. The diagram below that I created is an example of how a Switch Statement works.

Input != Case function

Default takes place, breaks then returns

Case function breaks and returns.

Case function action takes place.

Input = Case function

Player’s input

As you can see from the diagram above, if the player inputs a command that equals one of the existing case functions, that case function runs, then breaks, which is an action like a return function. While if the player inputs a command that is not a listed case function, then the default function takes place instead, prompting the player with a reminder that it is not a correct command and then breaks and returns. I chose to use a Switch statement because I found it to be the most efficient way to handle my command inputs, while being easy for myself to understand.

Throughout my text adventure game, the player must collect and use items. These items are created through the pre-made Item class, using a public string called “ItemName” and a public Item class that contains a string within the parameter, both of which can be accessed from any other classes. In this case, the Location and Game class. Using the Location class’s map building function in the Game class, this allow me to refer to the Item class, while being able to access the public item class that contains the string, letting me list the items in the maps construction, then being able to refer to them using the “ItemName” String.

To add to my item class, I created a function within public Item class that calls to the location class to detect the current location of the player, restricting certain items from being used within a specific room. This is done through the same method of creating the items, upon the map building code, where the items are listed for each room. A location is also listed afterwards. I chose to use the following piece of code due to my research on Zork as I like the core objective of the game which was to collect a certain amount of treasures, it was simple but fun. That is what I wanted to try and recreate it in my text adventure game.

**Start Screen and Character creation**

When it came to creating my opening menu screen I used a similar method to Conway’s game of life that we were taught. Using an array to create a grid of 17x34, the grid is made up of 0’s and for each 0 the gird draws an empty cell, but if the number is greater than 0, such as a 1 or 2 an empty cell will be replaced with an asterisk and again depending on the number the asterisk will be a different colour. This is the method I used to create the buildings seen on the start screen.

I wanted my player to feel more immersed in the game, I did so by allowing them to name their character. I did this by creating a public string called playerName in the Game class. That is changed when the player is prompted to input their name, this is done by changing the playerName string by using the function playerName = Console.ReadLine.