Introduction

Author: Sofia Šišić

Text adventure or “interactive fiction” games have been around for a while. We all enjoyed playing them, therefore, our group decided to create one, but with a twist in the storyline.

Our game will be revolving around a Pizza Hut employee, whose main goal is to find the clues about Stinky Monkey’s favourite pizza, gather those ingredients and then deliver the pizza in time. It will consist of levels or stages, where the player can interact with characters and objects, encounter riddles and solve puzzles

What we expect from this project is to learn how to build a game as a team, as well as creating an enjoyable and refreshing experience for our fellow computer science students or any text adventure enthusiast.

The goal for us is to create a game that is independent of the storyline and we will try to follow through with this as accurately as possible. The descriptions will be kept separate from the source code (which will enable other users to design their own levels for a more personalized experience). We intend to implement sound effects, cheat codes and other features that are explained in more detail in the following section.

Features

Functional features:

Author: Daniel McHugh

Six actors in feature set design: Simplicity, functionality, familiarity, novelty, automation, control

Traditionally, text-based adventure games have a small number of functional features displayed to the user. As the name implies, text-based games are (usually) limited with the interaction between the game and the user through a terminal (text-only). Using these limitations as a starting point, we wish to further improve aspects of text-based games by adding additional features. Most of the functional features for this system will fit into two categories: the internal aspects of the game (moves, score, inventory, etc.) and the exchange of data between the user and the game (input-box, text/image display, etc.). These features draw heavy inspirations from other text gamed adventure games such as *Zork*[1].

For our system, we shall maintain the functional features found in many other text-based adventure games. The reasoning behind this being that these functional features are synonymous with the subgenre of games. Though there is much room for innovation and improvement, these features are core to the genre and should remain in this system to still be considered a text-adventure game. Moreover, the familiarity of these features shall help users with experience of similar systems. These will include features such as: an input-box to receive commands by the user, a text-display box to convey information to the user and saving the user's progress.

Furthemore, our system shall include additional features to hopefully innovate on the complexity and enjoyability of text-based adventure games. These features should add some noverly to the system for users familiar to this genre of games while also including quality-of-life features for newer users. The definition in use for text-based games is that the input by the user shall only be in the form of text. This definition however, does not limit the system in how information shall be presented. Adding features such as an image display box and sound effects to the system should serve for more creativity and complexity for the games played on this system, while other features such as autocompleting commands and shorthand should offer a smoother experience for the users.

[1] https://en.wikipedia.org/wiki/Zork

ID	Short name	Description
F1	Location	This will display the current location of the player. This should help them be able to navigate throughout the game. For extension, we could also display the available/visited locations around the current one.
F2	Input box	The text input box which will be the place for the user to interact with the game.
F3	Commands	Commands will be the user's way of interacting with the game environment.
F4	Text display box	The text display box will display information about locations, characters and dialog.
F5	Autocomplete/ shorthand	Autocomplete and/or use shorthand for common commands which will be used while interacting with the system.
F6	Image display box	Show graphical information that is related to the current game state. Since the game is played on a terminal, these images will have to be in the form of ASCII art or other text based images.
F7	Save game state	Should be able to save and load the current game state to allow players to play in more than one sitting.
F8	Cheat codes / Easter eggs	Typing in special (hidden) commands to apply additional effects to the game state.
F9	Sound effects	Short sound effects played through the terminal when relevant to the current game state
F10	Certificate	Present a PDF certificate at the end of the game stating that the player successfully completed the game

Quality requirements:

Author: Donal Shortt

Our main focus when deciding on quality requirements for our project were project maintainability. We felt like this was the most important aspect to address for two reasons.

Firstly we believe that the most we can get out of this course are the skills to be able to design projects that are easy to maintain for more than one person.

Secondly, up until now we have been to a large extent programming by ourselves. The limited experience each of us has had on group projects have often ended up in a jumbled mess of code that no one really understands and with one person doing the majority of the work as they were seen as the “best programmer”. Given this, we have four quality requirements that focus on project maintainability.

Usability and reliability were our next main concern. We were dedicated to making our game easy to use as it is after all a game and the learning curve for a game should not be too steep else the developer risks players losing interest. Creating a reliable game was again motivated by a focus on maintaining player interest, as if a game is super buggy player’s interest will not last long. (There are certain exceptions to this rule).

Lastly we wanted the game to be available on a wide variety of platforms. Given that our target audience is computer science students, we assumed that most would be familiar with the bash shell and as such we decided to run our game from that.

Responsiveness and security were relatively low priority for us. We decided responsiveness was relatively unimportant because we were running the game out of a terminal and if the game became unresponsive at any point it would most likely be due to an infinite loop somewhere rather than poorly optimised code.

Security was another aspect that we didn’t place too much importance on. We would welcome any exploits people find in our game as we think it is part of the charm of the genre.

ID	Quality Attribute	Description
QR1	Reliability	Sanitise user input to a certain predefined format
QR2	Reliability	Game state change shall be triggered only by user input
QR3	Maintainability	States/Levels/Areas shall be defined in a certain way such that adding new ones is as simple as filling out a template
QR4	Maintainability	All code shall conform to a unified style
QR5	Maintainability	Maven shall be used to handle project dependencies
QR6	Maintainability	All game data should be stored on JSON files, except for game engine provided info, such as the player's character name
QR7	Usability	All text displayed by the game shall be comic sans, a universally revered font
QR8	Usability	The user should be able to begin a game within 5 keystrokes, excluding character name input
QR9	Availability	The game shall run from a BASH terminal

Java libraries

Author: Nariman Gasimov

Lanterna - For the graphical user interface terminal of the game. Lanterna can be used to create more video game-like feel while playing the text adventure game.
Fastjson - Simple and fast json processor library. Will be used to keep the data of the game for future use.
Flying Saucer - To render a given html/css markup page to a pdf file. Will be used in our game to give the winning certificate in the end.
Lombok - We will use this library because it reduces verbosity in the java code. Instead of writing many lines of getters and setters in a class, this library will provide all of it. This will make the code look nicer and will as well save us some time.
JUnit - For unit testing purposes. We will use this framework because it provides test runners, assertion, assumption and exceptions, which can be useful while developing the game.
Time4J - This library gives useful time tricks that can be used for calculating and measuring the game duration and possibly identifying the fastest player for ranking purposes.
JLine - We will use this library specifically for the autosuggestions functionality. It will be very useful to let the player know what are the possible options to write on the prompt for user-friendliness.