CCPS 406 Project: Text Adventure

Textual Data

1) Rooms
2) Items
3) Characters
4) Messages

Group members:

- Aden Hersi (Aden.hersi@ryerson.ca)
- Charif Sukkar (charif.sukkar@ryerson.ca)
- Ervin Demnushaj (ervin.demnushaj@ryerson.ca)
- Chris Fontein (cfontein@ryerson.ca)
- Jeesoo Kim (j17kim@ryerson.ca)

Instructor: Ilkka Kokkarinen
Subject: CCPS 406 Introduction to Software Engineering
School: Toronto Metropolitan University
Date: May 15, 2022

1) Room Textual Data File Format

The rooms in our game will be represented in the data files by their description (long and short), the items inside that room, the items dropped on the floor, the characters in that room, characters that have departed while the player was in that room, and the connections of that room to others.
The longDescription will be viewed when the player first enters a room or decides to use a >look command while in that room. Otherwise, after the first visit, the shortDescription will be viewed every time the player returns. The long description will contain the detailed String description of the room's appearance while the short description will contain the room's descriptive name.
The descriptions of the characters and items in the room along with those on the floor will be synthesized with the room's long description by our source code. The short description will alternatively be built with the descriptions of the characters inside only. This way, when the player first enters a room or uses >look, they will be informed of the characters and items in plain view as well. When they return to the room, they will be informed of only the characters inside.
The items in the room that have not been dropped by the player or NPCs will be represented by itemsInRoom. The items that have been dropped on the floor either purposefully or upon character death will be represented by itemsOnFloor. The characters currently in the room are represented by charactersInRoom.
departingCharacter will represent the characters that leave the room while the player is inside. Our source code will output a message to the player when this happens. Whenever a character enters the room that the player is in, our source code will also alert the player with a message.
The entrances and exits and their connections to other rooms will be represented by connections and monsterConnections. monsterConnections refers to the paths that are only available to the monster NPC for use.
Content inside <> will be generated by code and replaced when description functionality is called in the game.

Below is a sample format for a room's representation in the textual data file using the YAML format:

- Room1 :
    - longDescription : "Long description of room. There is a table on the north wall containing <table.contents>. There are 3 barrels in the corner."
    - shortDescription : "Cave Entrance."
    - itemsInRoom :
        - item1
        - item2
    - itemsOnFloor :
        - item4
        - CharCorpse
    - charactersInRoom :
        - player
        - NPC1
    - departingCharacter:
        -NPC3
    - connections :
        - s(south) : Room2
        - n(north) : Room4
    - monsterConnections :
        - w(west): monsterRoom

2) Items Textual Data File Format

The items in our game will be represented in the data files by their description, weight, value, and type.

description represents a String description of the item's appearance and it will be viewed upon using the >look command on an item that is in plain sight or when a description of that item's container is printed. The weight of the item and its value are represented by integer values.

The type of the item represents the special class that the item belongs to. These classes can be 'Item', 'Container', 'Consumable' and 'Equipment' and each will dictate the specific ways that the item can be used during the game. For unique items with specific functionality, type will refer to the class of that unique item. All consumable items will be unique subclasses of Consumable due to the unique nature of their functionality.

If the item belongs to the class 'Container':

- content will represent the items that are currently inside that container.
- contentDescription will be a String description to be combined with the short descriptions of its contents by our code.
- emptyDescription will represent a default description if the container is empty.
- openOrClosed will represent by boolean value if the container is open or closed
- closedDescription will represent a default String description of the container if it is closed.

If the item is 'Consumable' then the effect of that item will be handled by our source code. Once that item is consumed it will no longer exist.
If the item is 'Equipment' then equipSlot will indicate where the item can be equipped and equipVal will indicate the integer value in attack or defense.

Below is a sample format of an item's representation in the textual data file using the YAML format:

Generic item Example

- Plate:
    - description : "(item description)"
    - weight : int
    - value : int
    - type : Item

Container Example

- Chest:
    - description : "(item description)"
    - weight : int
    - value : int
    - type: Container
    - content :
        - item5
        - item6
    - contentDescription : "In the chest you see <contents>."
    - emptyDescription : "The chest is empty"
    - openOrClosed : true/false
    - closedDescription : "A closed chest"

Equipment Example

- Sword :
    - description : “(item description)”
    - weight : int
    - value : int
    - type: Equipment
    - equipSlot :
        weapon
    - equipVal :
        - attack : 1

Consumable Example

- HealthPotion :
    - description : “(item description)”
    - value : int
    - weight : int
    - type: healthPotion

3) Characters Textual Data File Format

The characters in our game will be represented in the data files by their description, their equipped items, current health value, maximum health value, base attack power, the maximum weight limit of what they can carry, base durability/defense against attacks and effects that the character is under.

The description will be a String representation of the character's physical appearance. This will be used when the player enters a room that the character is already in, as the room's long and short description will contain the descriptions of the characters inside. It will also be available when a new character enters a room that the player is already in, or upon using the >look command on a character.

equipped will represent the items that the character has equipped, along with where on the body they are equipped. inventory represents the items that the character is currently carrying. lastAction will contain the String representation of the last action that the character has taken. This will be used to describe the observed actions of other characters.
effects will represent the current effect that a character is under. This can be an effect by potion, such as a temporary change of base stats, or a more complex effect such as being invisible to every other character for a few turns (execution handled by our source code).

The remaining elements (currentHealth, maxHealth, baseAttack, weightLimit, baseArmor) will all be represented by integer values to be used inside the game's calculations during gameplay.

Below is a sample format of a character's representation in the textual data file using the YAML file format:

```
- NPC1 :
        - description : "a/an (character's physical description)"
        - equipped :
                weapon : sword
                shield : buckler
                head : helmet
                body : cloak
        - inventory :
                - item1
                - item2
                - gold
        - lastAction : "equip sword"
        - currentHealth : int
        - maxHealth : int
        - baseAttack : int
        - weightLimit : int
        - baseArmor : int
        - roomVisited : {Room1, Room3}
        - effects
                - specialEffect : {value1 : 1, value2 : 2}
```

4) Program Messages Data File

The program messages in our game will be represented by String descriptions of specific messages that the player will encounter during the game. startMessage will contain a String introduction to the game that will be available when the player starts playing for the first time or uses the >intro command. endMessage will contain separate messages that pertain to the player either winning the game or losing. alerts will contain specific messages automatically printed when the player must be made aware of something. instruction will contain the verb keywords that are used in gameplay and an alert message if the wrong keyword is used.

- startMessage : “Welcome to the world of… Your goal is to… But be wary of… Best of luck Adventurer!”
- endMessage :
    - death : “You have been killed!”
    - win : “Congratulations! Let’s see how much you got. You got away with <value>!”
- alerts :
    - lowHealth : “Your current health is dangerously low.”
    - charEnters : “Someone has entered the room.”
    - charAttacks : “You have been attacked”
    - locked : “Cannot be opened this way”
- instruction :
    - naMessage: “Command is hard to understand. Please type “actions” (e.g.) to see the list of available verbs for command or ‘action, info’ for what a specific action does”
    - actionsList : [look, take, inventory, equip, open,...]