Julia Franco
Software Development I
Wednesday 5:00-9:15pm

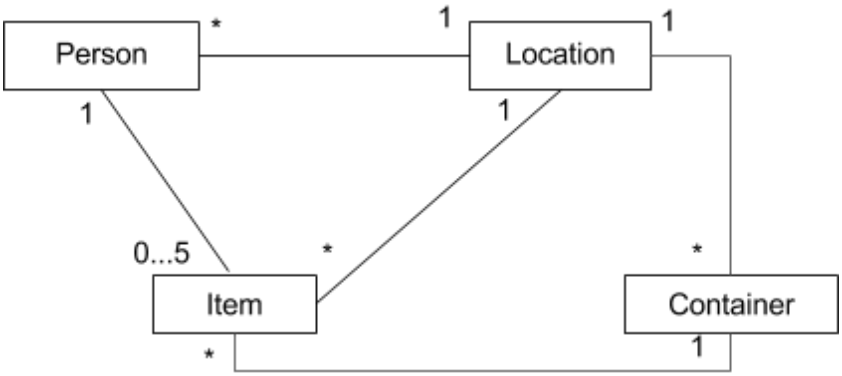
Abstract: Cove Wrench Drop: The Game is a text based adventure game in which the user navigates around the map to pick up items to combine in hopes of retrieving the last wrench from where it has been dropped.

Introduction: This game was developed to provide a challenge to the developer as well as to create a quirky and fun game for someone to play. It could potentially be defined for a small scale release on the internet if the files can be packaged in such a way suitable for distribution, or simply to exist for future amusement. This paper will include a detailed description of how users interact with it, the requirements it addresses,a literature survey of other works, a user manual, a final summary and references consulted.

Detailed System Description. Describes what the system does and how specific users interact with it. It also describes how classes interact (in UML).: This system only has one user at a time- the player. The player interacts with the system by entering commands. The game is driven by a while loop that runs while true. The first thing the system does upon launch is create all of the items in the game- locations, items, containers, and the player object itself. It then renders the description of the opening location, which is set by the no-args constructor that creates the player object. The system asks the user for a command. The user can enter any number of commands that must match exactly what is specified in a series or if/else if/else statements inside the game loop. The user can also enter a name of an item or a container if the command lends itself to it (ie, take, drop, transfer, etc). The system reads the input as the first word, then the rest of the line, to account for multiple word names. These inputs are stored in an array that the getInput function returns. The system then looks at the first argument for the command. If the command requires secondary input, it first looks to see if the user has supplied any information. If it has not, it then calls the function to get the secondary input. Along with this, it usually prints out the user's options for secondary input (ie, items in the location for the player to take, containers in the location to rummage through, etc). If the user has entered a command requiring secondary input, the system attempts to execute the command. It takes the second input argument as a string and then checks to see if that string matches the name attribute of any known object in a the applicable category. This method either returns an object of applicable type. If it is null, it skips the rest of the function and prints out the correct message. Otherwise, it checks to see it the object of applicable type is in the user's location. If so, it performs the requested action, otherwise displaying an error message. These functions are driven by loops to loop over all the elements in a list of containers and objects, and if/else statements. The player's inventory is an ArrayList of items, making the methods to add and subtract items from the list very simple. If an item is added to the person's inventory, another function is called to remove it from the location. Some containers can be hidden- meaning that if a player examines the location for items and containers, the hidden container will not show up in the list. If a player tries to rummage through this hidden container, it will say it does not exist. A hidden location is another Container object created via an overloaded constructor that contains an argument to set the "hidden" boolean data field to true. Then when displaying location contents and other methods, there is an if/else statement to see if "hidden !=true". If it is true, it will skip over whatever other statements are nested beneath it. The player enters navigational commands, and depending on the command, passes a specific value to a function to determine where to go. The passed argument specifies the column in a 2D matrix in the location class. The rows represent the locations, and the number at naMatrix[x][y] represents the index of the element in the list Locations.places is in that specific column of the value of the data field of Person.player.locationIndex to determine the row to look in. This function, Person.moveLocation, sets the value of Person.player.locationIndex to the number at this intersection. The system then attempts to render the location. In most circumstances, the location renders fine and returns as true, which sets the data field in Person renderLocation as true, and allows the system to go back to the top of the loop. There are two

exceptions at this time- due to the nature of Location.place[5], it immediately causes the player to lose when they go there. The system prints the description of the place, and then prints the appropriate ending from the MainFile.DisplayQuit(int) switch statement that handles what ending is displayed. This method returns the renderLocation data field false. At the top of the loop, there is an if statement to break out of the loop when Person.player.renderLocation is false. The other instance of Person.renderLocation returning false is when the user goes to Location.places[6]. As a joke, the player can randomly be forcibly ejected from the game. When the system attempts to render this location, it runs a method to determine if it can. It calls the MainFile.getRandomNumber method and if that method returns 12, this method returns true which causes the render method to return false and display the appropriate ending message. When the user leave the game, either willingly, victoriously, or forcibly, MainFile.DisplayQuit(int) is called, with the integer value corresponding to cases in a switch statement to print the correct ending message before asking the user if they would like to restart.

UML Diagram of Class Interaction:



A person can only have one location, but a location can have many people. A location can have many items and many containers, but each container and item can only have on location. A person can have up to 5 items, but an item can only be owned by one person. A container can hold unlimited items, but an item can only be in one container at a time. Really, an item can only be in one of those places at one time.

Requirements: At the current moment in time, the following proposed requirements have been met. The user can collect items and hold them in an inventory. However, the inventory has to be enabled by finding an appropriate item that can hold an inventory. The user can outright discard items, or put them in a secondary inventory. There is also a map the player can find and display to visually see the layout of the world. The player has specific attributes, including a name, an inventory, and a location. There is a map the a person can navigate, with the option to go in 6 directions- up, down, left, right, forward, and “backward” (acts as a turn around and go forward.) The player can rummage through containers to find other items. There are a number of proposed requirements still to be fulfilled. One is the ability to combine the items together and eventually deploy these combined objects. If the objects deployed have the correct attributes, there is a chance the user will retrieve the wrench that it is supposed they dropped. There also may be other people introduced into the game that the user can interact with. Depending on how development continues, other requirements may be introduced and met.

Literature Survey: There are many well known text based adventure games. Titles include Zork, The Hitchhiker’s guide to the Galaxy, and Spider in Web. Many of these games include commands used here- basic navigation, the ability to hold and use items. These games have unique stories and features that shape the game play- the ability to dig and use swords, dependent upon the items held and the scenario the player faces. This system employs many of the same functionality and styles of these games, in the descriptions and commands usable in the games.

User Manual: The user inputs commands to move the player around the map, and to collect and drop items and hold them in an inventory. The user can also rummage through various containers for additional items.

Since the inventory can only hold 5 items, there is an unlimited secondary inventory that the user can transfer items to. Many commands can be used in conjunction with an item name to perform specific actions.The user can also type help for a list of valid commands and pointers for their use, as well as quit out of the game. It will ask the player if they want to reset. If they choose yes, the game will begin again, Otherwise, it will print the credits ans terminate the execution. In the future the user will also be able to combine items, and "use" them to attempt to accomplish the goal of the game. Another feature for future consideration will be adding in people for the player to interact with. The user input is not overly user friendly at the current moment- the input must exactly match the specified names and commands- if one wants to pick up an item called "A BUNCH OF PARCANS", the user must type "take A BUNCH OF PARCANS", "take parcans" or "take a bunch of parcans" will result in a message that says that item does not exist. This hopefully will be simplified going forward. The user will also be able to acquire money and buy items that could be potentially useful.

Conclusion: The system is currently on track for completion on time. The ground work has been laid, allowing the player to navigate around and interact with items, and seems as if many of the potential kinks have been worked out. The next phase of the project will be mainly involved with implementing features that allow the user to accomplish the purpose of the game, as well as introducing an unpredictable element of danger.

References/Bibliography: Y. D. Liang's Introduction to Java Programming, Comprehensive Ver., 10th Ed., StackOverflow.com, as well as Zork and Spider and Web.