

CS 329E: Bulko

Semester Project (Part 2)

1 Problem Definition

In this programming project, you are creating your own simple adventure game. In Part 1, you designed data structures, created a few basic functions, and included some test code to verify that your framework is correct.

In Part 2 of the project, you will:

- modify your data structure to include items that can be found in a room, picked up by the player, carried around in the player's inventory, and dropped in another room.
- remove the test code hardcoded into `viewDidLoad()`, and create an iOS user interface where the player can type in simple one-word or two-word commands to navigate the map, interact with items, and see results.
- add code to save the state of the game, so that if the user terminates the app and relaunches it, the game is restored to the last state.
- replace the practice map with a custom map of your own design!

2 Storyboard:

Create a simple user interface on the storyboard. You only need three Views:

- a `UIImageView` with an outlet called `imageView`. For now, you can either leave this empty, or you can add a random image or photo to your app to fill it; you won't use it till later.
- a `UITextView` with an outlet called `outputArea`.
- a `UITextField` with an outlet called `commandField`. Give it placeholder text "Enter command".

You will no longer be printing anything to the debug area! You will be sending output to the `UITextView`. Replace any print statements in your code with statements where you instead set the value of `outputArea.text` to the desired output.

3 Room contents:

- Add a property to the `Room` class called `contents`, an array of strings that will store the list of objects currently located in that room. Modify the `init()` method for class `Room` to add a parameter for `contents`.
- For testing purposes, modify `loadMap()` by adding objects to several rooms.

- Add "plate" to the end of room2 to indicate there is a plate in the Dining Room.
 - Add "cup", "fork", and "knife" to the end of room3 to indicate there are a cup, a fork, and a knife in the Kitchen.
 - Add "magazine" to the end of room5 to indicate there is a magazine in the Bathroom.
 - Add "pajamas" and "slippers" to the end of room7 to indicate there are pajamas and slippers in the Master Bedroom.
- Modify the `createRoom()` method to handle the new `contents` property. If there are objects in the room (as specified in `loadMap()`), `contents` should contain an array of those strings. If there are no objects in the room, `contents` should be set to "None".
 - Write a function `getContentsOfRoom()`, which takes a `Room` object as a parameter, and returns a string. If the room contains a cup, fork, and knife, this function should return a string that looks like the following:

Contents of the room:

```
cup
fork
knife
```

- Modify `look()` to include a call to `getContentsOfRoom()` so that it outputs the contents of the room along with the name of the room.

4 Inventory:

Create a global variable, an array of strings called `inventory`. This will represent the items currently being carried by the user. Initially, this variable should be set to an empty array.

- Write a function `pickup()` that takes a string as a parameter. If the contents of the current room contains that string, add it to the user's inventory and remove it from the room's contents. `outputArea` should display either "You now have the ITEMNAME." or "That item is not in this room."
- Write a function `drop()` that takes a string as a parameter. If the user's inventory contains that string, remove it from the inventory and add it to the room's contents. `outputArea` should display either "You have dropped the ITEMNAME." or "You don't have that item."
- Write a function `listInventory()` that displays what is currently in the user's inventory. If the user is currently carrying a cup, a fork, and a knife, `outputArea` should display:

You are currently carrying:

```
cup
fork
knife
```

If there is nothing in the user's inventory, then it should display:

You are currently carrying:
nothing.

5 The command interpreter:

Write a command interpreter to allow the user to type in simple one- or two-word commands into the text field `commandField`. First, set up the text field.

- Make the class `ViewController` conform to the `UITextFieldDelegate` protocol. (This does not require any methods or properties to be defined.)
- In `viewDidLoad()`, set `commandField.delegate` to `self`.
- Define the method `textFieldShouldReturn()`. (Use autocomplete to help you with the parameters and return value.) This method is called whenever the text field passed as a parameter has focus, and the user hits the return key.

The code for `textFieldShouldReturn()` is your command interpreter. When the user enters a string and hits "return", retrieve the string and break it into "words". *Hint:* there is a string method called `components(separatedBy:)` that works like the Python string method `split()` you might find useful.

Write a `switch/case` structure that specifies what to do based on the word or words you extract from the text field. For example, if the user types in the word `look` and hits the return key, you want to call the function `look()`. Or if the user types `north`, call `move(direction: "north")`.

After the command is executed, clear the text field so that the placeholder text reappears and is ready to accept a new command from the user.

The list of commands you are required to implement includes:

```
look
north
east
south
west
up
down
inventory
exit
get ITEM
drop ITEM
help
```

Note that there is one more command you must implement: `help`. This command simply outputs the following text:

```
look:           display the name of the current room and
                its contents
north:          move north
east:           move east
south:          move south
west:           move west
```

up:	move up
down:	move down
inventory:	list what items you're currently carrying
get item:	pick up an item currently in the room
drop item:	drop an item you're currently carrying
help:	print this list
exit:	quit the game

6 Customize your app:

Customize the app to make it *your* unique game.

- Throw away the map defined in `loadMap()`, and replace it with a map of your own choosing. This could be your own home, a business or restaurant, the apartment that the characters in your favorite TV show live in, a castle or dungeon, part of a building on the UT campus - be creative! You can choose whatever you want, but try to ensure that the map is interesting. Ensure it has *at least* ten locations (rooms, hallways, balconies, etc.) that can be easily navigated with compass directions and stairs. (*Hint:* if you have several rooms along a hallway, such as the hallway containing our classroom, you can break the hallway down into "West End of Hallway", "Central Part of Hallway", and "East End of Hallway", with doors on the north and south side of each part of the hallway.)
- Add images to your app representing each of the locations in your map. Add code to your app so that whenever you move from one location to another, the image in `imageView` is updated to reflect the new location. To get these images, you could:
 - take photos of real rooms in your home.
 - make simple drawings on paper, and take photos of those drawings.
 - download screenshots from movies or TV shows.

Be creative!

- Place appropriate objects in some of the locations.
- Add images of your objects, and add code to display those images in `imageView` in place of the current location whenever you interact with one of the objects.

Now comes the fun part! Think of interesting ways you can use the objects to interact with the game, and create new intuitive commands for using them in the command interpreter. (Note that this will break the data-independence requirement enforced in Part 1. This is acceptable and expected.)

- *Example:* In the description for the kitchen, say "There is a table here." Modify the `pickup()` function so that if the user enters the command `get table`, it outputs "The table is too big to get. But there is a note on the table." The user is allowed to get the note. Now add a new command `read ITEM`. (Don't forget to add it to your help text.) If the user enters `read note`, output the message, "The note says, 'See you when I get home. I left you a surprise in the refrigerator! XOXO'". Of course, this suggests you want to add a refrigerator object to the kitchen, add a command `open ITEM` that applies to the refrigerator, and place some gettable treat in the refrigerator.

- *Example:* Place a key in a particular room that the user can get. In another room, mention that there is a closed door on the east wall. If the user tries to move east from that room, output the message, "The door is locked." Add a command `unlock door`. (Don't forget to add it to your help text.) If the key is in the user's inventory, output the message "The door is unlocked.", and moving east will now succeed. However, if the user does not have the key, output the message, "You don't have the key." Of course, this interaction is more fun if the key is not easy to find, and if opening the door leads to something interesting, such as more rooms with items in them. It also makes sense to add a `lock door` command while you're doing this.
- *Example:* When the user enters a room, you could output a message saying the room is dark, and it's too dangerous to enter without a light. The `look` command says you can't see anything. You could either have the user search for a light switch on the wall and turn it on first, or you could have a flashlight or candle object in another room that they would have to find first.

Include at least four interesting interactions of your own design in your final project.

Finally, customize the UI. Add a launch screen. Use non-default colors and/or fonts. Add borders around the text field and/or text view. **This step is optional**, and you will not be penalized if you don't do it, but it's an opportunity for you to showcase what you've learned in this course and make this app something you can be proud to show off to your friends and relatives, or to demo during a job interview.

7 Enable the user to save progress:

One final technical requirement for this project: add code that saves the state of the game, so that if the user terminates the app mid-game and relaunches it, the game will be restored to the last state. You have several options of where you can store your project state: we've learned about Firebase/Firestore, Core Data, and User Defaults. Choose one. Some things to consider:

- What are the implications of this decision? What advantages and disadvantages do each of these options have?
- What values define the "state" of the game? In other words, what things do you need to save in order to restore the game to the last known state?
- When should saves occur? Remember that the user can terminate the app at any time.

8 Grading criteria

1. Basic function and gameplay: the room contents, inventory mechanism, and command interpreter work as expected. (50%)
2. Customization: the app uses an interesting custom map and objects, and includes at least four interesting interactions. (50%)
3. **Note that if the app does not build and run, ZERO points will be given.**
4. The Coding Standard is followed. One point deducted for each violation.

9 General criteria

1. I will be looking for good documentation, descriptive variable names, clean logical structure, and adherence to all coding conventions expected of an experienced programmer, as well as those outlined in the Coding Standard document. There will be penalties for failure to meet these standards.
2. Xcode will automatically generate standard headers to your .swift files. Add lines to each Swift file so that the header includes the following:

```
// Project: LastnameFirstname-Project  
// EID: xxxxxx  
// Course: CS329E
```