CS 329E: Bulko
Semester Project (Part 1)

1 Problem Definition

In video game culture, an adventure game is a video game in which the player assumes the role of a protagonist in an interactive story driven by exploration and puzzle-solving. Early adventure games from the 1970s and early 1980s were text-based, using text parsers to translate the player's input into commands. As personal computers became more powerful, the games were augmented with graphics, and later moved into point-and-click interfaces.

Adventure, or "Colossal Cave", was released in 1976, and is widely accepted to be the first adventure game. Other notable adventure game series include Zork, King's Quest, The Secret of Monkey Island, and Myst. Many of these are now available to play for free online, such as here.

In this programming project, you will create your own simple adventure game. The project will be completed in two parts. In Part 1, you will create a simple "practice" map for an adventure game. You will create objects representing the rooms of a house, along with a method to display the features (properties) of a room. You will also create a function that allows you to move from room to room in the house. Your goal in Part 1 is simply to design data structures, create a few basic functions, and include some test code to verify that your framework is correct.

The practice map for our game will be a symbolic representation of the house shown in Figure 1. Later this semester, in Part 2 of the project, you will add a user interface, add objects to the game that a user can interact with, and then replace the practice map with one of your own design.

For our compass directions, "north" means towards the top of this webpage, "east" means to the right, etc. So we would say you could go through the door from the Living Room to the Dining room by going north, and from there, pass through the door to the kitchen by going west. Although there is no label on the floor plan, there is a "room" called the Upper Hall on the second floor representing the region connecting the Bathroom, Small Bedroom, and Master Bedroom by doors, and connected to the Living Room via the stairs. You would take the stairs by moving in the directions "up" or "down".

2 The `Room` class:

Your first task will be to create a Swift class called Room. This class must contain at least the two following methods:

- `init()` takes seven arguments:
  - `name`, a string representing the name of the room.

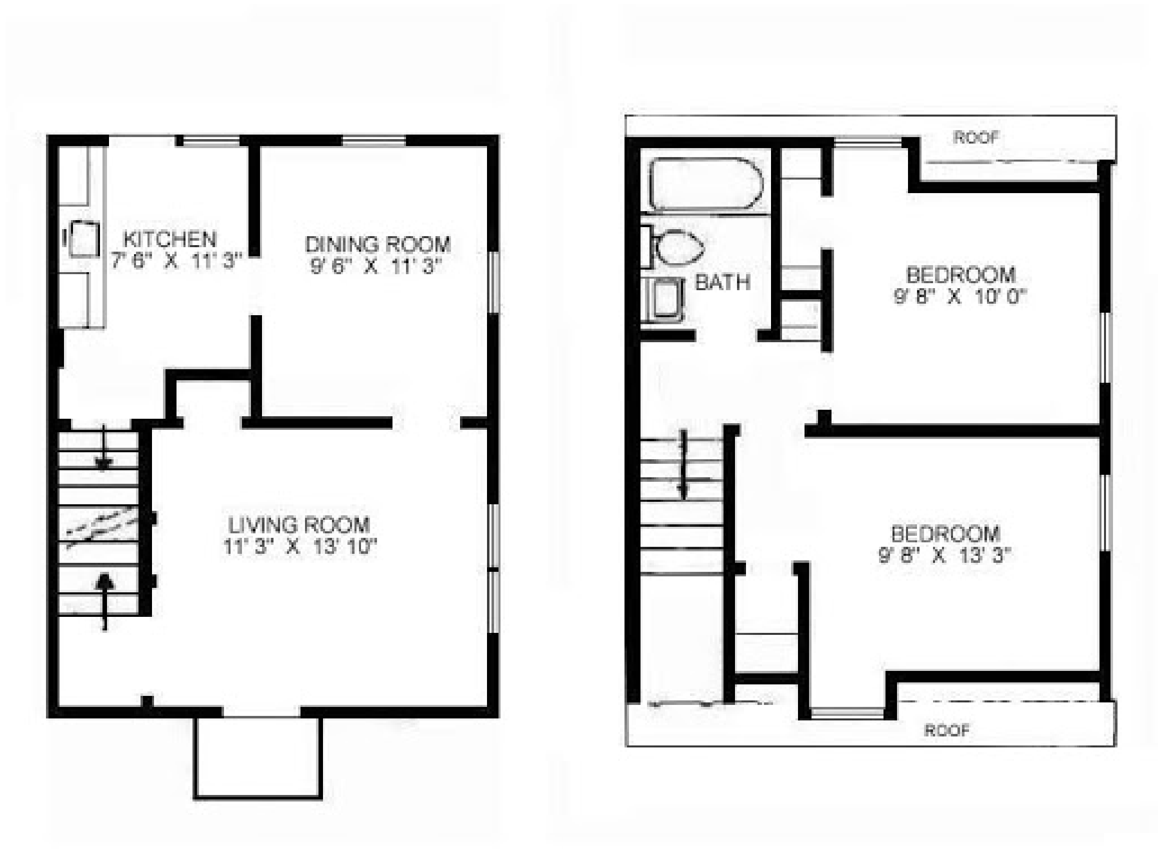

Figure 1: Floor Plan for Practice Map

  - `north`, a string representing the name of the room you would reach if you went north from this room.
  - `east`, a string representing the name of the room you would reach if you went east from this room.
  - `south`, a string representing the name of the room you would reach if you went south from this room.
  - `west`, a string representing the name of the room you would reach if you went west from this room.
  - `up`, a string representing the name of the room you would reach if you went up a flight of stairs from this room.
  - `down`, a string representing the name of the room you would reach if you went down a flight of stairs from this room.

  A value of `"None"` for one of these arguments means you can't travel in that direction from this room (because there is no door or stairs leading to another room in that direction).

- `displayRoom()`
  - displayRoom takes no arguments. It tells the `Room` object to display its name, along with the names of all of the rooms it is adjacent to. For example, if `myRoom` were a pointer to the Upper Hall, then `myRoom.displayRoom()` would print out:

```
Room name:  Upper Hall
   Room to the north:  Bathroom
   Room to the east:   Small Bedroom
   Room to the south:  Master Bedroom
   Room below:         Living Room
```

followed by a blank line. Note that if the value of a parameter is `"None"` (which is the case for moving west from the top of the stairs), nothing should be printed for that direction.

3 The data structure:

Figure 2 illustrates the data structure you must build to represent the house.

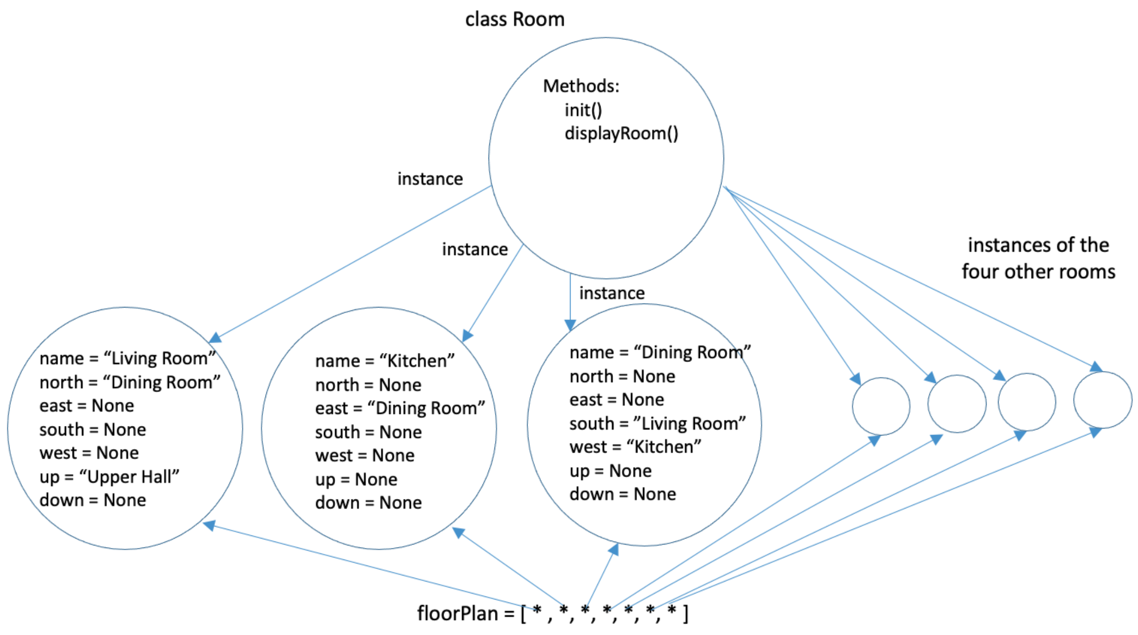

Figure 2: Class Diagram

4 Writing the program:

It is important that you create the data structures, functions, and variables *exactly* as described below. If you digress from the specifications too much, you will make completing Part 2 of the project more difficult for yourself.

Create two global variables:

- `floorPlan` is a global array of `Room` objects that contains pointers to all of the rooms.
- `current` contains a pointer to the `Room` you’re currently located in.

`loadMap()`

- define seven local variables `room1`, `room2`, `room3`, ..., `room7`. These are arrays of strings that contain data for each of the seven rooms in our house. Each array contains the following strings in this order:
  - the name of this room
  - the name of the room you would reach by traveling north from this room
  - the name of the room you would reach by traveling east from this room
  - the name of the room you would reach by traveling south from this room
  - the name of the room you would reach by traveling west from this room
  - the name of the room you would reach by traveling up from this room
  - the name of the room you would reach by traveling down from this room
- calculate the value of `floorPlan` by calling `createRoom()` seven times to create seven pointers to `Room` objects, and collecting the objects created into a list.

`createRoom()`

- `createRoom()` is a utility function that takes one argument, one of the arrays of strings created in `loadMap()`. It uses the data in that list to create a new `Room` object with all of its properties defined according to the contents of the list. This will be used by `loadMap()` to create all of the `Room` objects in `floorPlan`.

`look()`

- `look()` takes no arguments and returns no values. It simply prints out the message, "You are currently in the CURRENTROOMNAME."

`getRoom()`

- `getRoom()` is a utility function that takes one argument, the name of a room (a string), and returns the corresponding `Room` object. You will find it useful to call this in writing `move()` below.

`move()`

- `move()` takes one argument: a string representing the direction the player wants to move from the current room ("north", "west", "up", etc.).
- `move()` should return a `Room` object, which represents the room the player ends up in after the move. This can be used by the calling program to update the value of `current` when the move is complete.
- If there is no room in the indicated direction, `move()` should print "You can't move in that direction." Otherwise, you should print "You are now in the NEWROOMNAME."

`displayAllRooms()`

- For each room in `floorPlan`, use the method `displayRoom()` to print out the properties of that room. You will find this useful for testing your code.

`viewDidLoad()`

- call `loadMap()` to create the data structure.
- call `displayAllRooms()` to print out the data structure, so we can verify that it was created correctly.
- set `current` to point to the Living Room, our starting point for testing your code.
- execute a series of calls to `move()` and `look()` to test your code.

These are the calls you should include at the bottom of `viewDidLoad()`. (The comments indicate the expected result of each call.)

```
        current = move(direction:"south")     // can't move this direction
        current = move(direction:"west")      // can't move this direction
        current = move(direction:"north")     // Dining Room
        current = move(direction:"south")     // Living Room
        current = move(direction:"up")        // Upper Hall
        look()                                // Upper Hall
        current = move(direction:"east")      // Small Bedroom
        current = move(direction:"east")      // can't move this direction
        current = move(direction:"west")      // Upper Hall
        current = move(direction:"south")     // Master Bedroom
        current = move(direction:"north")     // Upper Hall
        current = move(direction:"north")     // Bathroom
        current = move(direction:"south")     // Upper Hall
        look()                                // Upper Hall
        current = move(direction:"west")      // can't move this direction
        look()                                // still in the Upper Hall
        current = move(direction:"down")      // Living Room
        current = move(direction:"north")     // Dining Room
        current = move(direction:"west")      // Kitchen
        current = move(direction:"north")     // can't move this direction
```

Important: your code MUST be data-independent.

What does this mean? Your code should work regardless of what the map looks like. That means if you make references to "Living Room" or "Kitchen" anywhere in your code other than in the assignment statements in `loadMap()`, you are doing it wrong, and you will be penalized 20% on your grade for this assignment. If you add a new room to your map, change a room's name, or change the connectivity between two rooms, you want `loadMap()` to be the only function you need to edit.

The idea is you are writing a framework for a game that should work no matter what map is provided. If we wanted to change the setting of this game from the house plan shown above to, say, your apartment or home, Carlsbad Caverns, or Hogwarts, we should just have to change the data provided in `loadMap()`. If you have code anywhere else in your program that refers to a specific room by name, it will not work with the new map without extensive rewriting.

5 Expected output:

Recall that `viewDidLoad()` calls `displayAllRooms()`, and then makes a series of calls to `move` and `look()`, each of which produce output. Your output should look like the following:

```
Room name:  Living Room
   Room to the north:  Dining Room
   Room above:         Upper Hall

Room name:  Dining Room
   Room to the south:  Living Room
   Room to the west:   Kitchen

Room name:  Kitchen
   Room to the east:   Dining Room

Room name:  Upper Hall
   Room to the north:  Bathroom
   Room to the east:   Small Bedroom
   Room to the south:  Master Bedroom
   Room below:         Living Room

Room name:  Bathroom
   Room to the south:  Upper Hall

Room name:  Small Bedroom
   Room to the west:   Upper Hall

Room name:  Master Bedroom
   Room to the north:  Upper Hall

You are currently in the Living Room.
You can’t move in that direction.
You can’t move in that direction.
You are now in the Dining Room.
You are now in the Living Room.
You are now in the Upper Hall.
You are currently in the Upper Hall.
You are now in the Small Bedroom.
You can’t move in that direction.
You are now in the Upper Hall.
You are now in the Master Bedroom.
You are now in the Upper Hall.
You are now in the Bathroom.
You are now in the Upper Hall.
You are currently in the Upper Hall.
You can’t move in that direction.
You are currently in the Upper Hall.
You are now in the Living Room.
You are now in the Dining Room.
You are now in the Kitchen.
You can’t move in that direction.
```

6 Grading criteria

1. The data structures (`Room` objects and `floorPlan`) are created correctly. (35%)
2. Each of the functions and global variables listed above are created and meet the specifications. (35%)
3. The program produces the correct output. (30%)
4. **Note that if the app does not build and run, ZERO points will be given.**
5. The Coding Standard is followed. One point deducted for each violation.

7 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