Snakes & Ladders - Mulitplayer Support

Part 2 Learning Outcomes

- Can refactor and move the input logic to an InputManager class
- Can understand and implement a model manager class for handling an array of multiple player instances
- Can expose methods and properties to do stuff with multiple player instances
- Can keep track of the currently active player using an index property and the modulo operator
- * Can use an `NSMutableString` to build up a complex string
- Can use a BOOL flag
- Can sanitize user input by lowercasing

Part 2 Goal

- Add multi player support
- Player's should be able to see a score output that summarizes the score of all players after each roll
- Show the final score when a player wins and start the game again
- Add quit functionality

Add a `name` Property to the `Player` Class

To handle multiple players we need to start by making a small alteration to the `Player` class. We need a `name` property so that we can identify the various players in our inputs and outputs to the console. So, go ahead and add an `NSString*` property called `name`.

Create a `PlayerManager`

It is best to create a manager class to handle multiple player instances. Let's call this `NSObject`subclass `PlayerManager`. `PlayerManager` will need an `NSMutableArray` property to hold the player instances. We can call this `players`. We will need to initialize this array in our `init` override.

We can initialize a single instance of `PlayerManager` in `main.m`. Do this outside the `while` loop. Remove the old `Player` import and instantiation from `main.m`. `PlayerManager` will handle player creation.

Implementing Player Creation in `PlayerManager`

`PlayerManager` is going to need a method to create players. It is going to need to take a parameter for the number of players to create. Let's call this `createPlayers:`. This should be void since it should set the `players` property.

Fill out the implementation of the method by writing a for loop. Instantiate the number of players according to the passed in parameter value. Make sure you give each player a name like "player1", "player2", etc. As you create each player don't forget to add it to the `players` array.

Prompting the User for the Number of Players

In `main.m` we need to start the app with a log message to the user. This message should prompt them to input the number of players they wish to have.

If no players are created, then we need to prompt the user again to input the number of players. Prevent them from continuing until they input a numeric value.

One way to do this is to use the count of the `players` property on `PlayerManager`. Check the count of `players` to see whether any players exist. Once the `players` property does have a count greater than 0, we can use this to skip the creation state.

We can handle the game over state by removing all players from the `players` property. We can then call `continue` to trigger another iteration of the while loop. This should prompt the user to input the number of players again. We will return to this in the final section. So, leave it for now.

We can also use the user's parsed input to test to see whether a numeric value was correctly inputted. Calling `if ([someString intValue]) { }` will return `nil` if `someString` is not a numeric value.

So check the user's input for a numeric value with this technique. If what they input is not a numeric value, trigger another iteration of the while loop. You should include a message to the user to input a valid numeric value.

If `[someString intValue]` is not `nil`, then we can call the manager's `createPlayers:` method. Pass the manager the valid numeric value as a parameter.

`PlayerManager` Should Handle All Communication

`PlayerManager` should handle all communication between `main.m` and the `Player` instances.

So create a `roll` and `output` method on `PlayerManager`. Make sure `main.m` calls these manager versions and not the `Player` ones. The implementation of these methods on `PlayerManager` will call the `Player`'s implementation. To do this we need to track which player is currently active in the `PlayerManager`.

Tracking the Active Player

Let's create a property on `PlayerManager` called, something like, `currentIndex`. Make it an NSInteger. Let's set it to 0 in `PlayerManager`'s `init` override method.

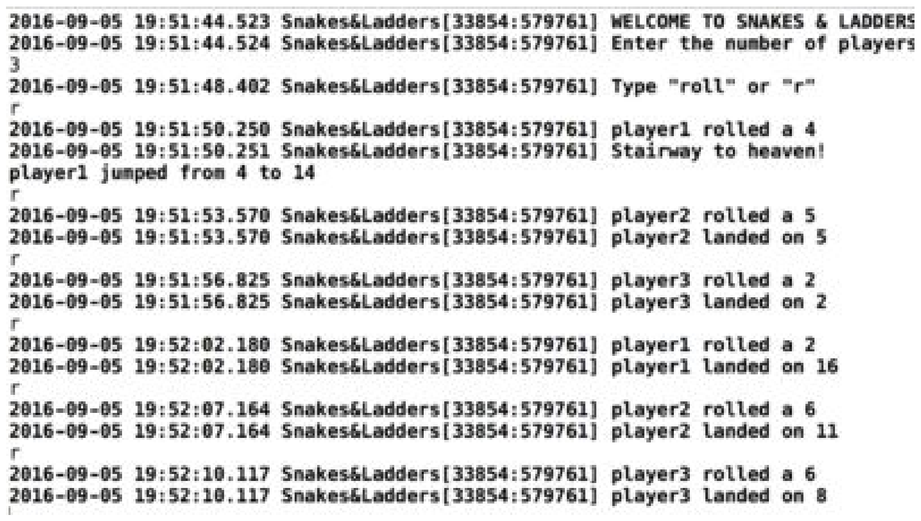
The `PlayerManager`'s `roll` method will have to increment this value. Several methods in `PlayerManager` will need to know the currently active `Player` instance. So we should create a method that returns the current player instance. This method will return a `Player*` computed from the `currentIndex`.

To do this I would like you to use the modulus operator. `currentIndex` should just keep incrementing by 1 for each roll. The method `currentPlayer` will compute the array index using 3 things. The modulus operator, the `currentIndex` and the count of the `players` array.

Put pieces together

Again, make sure you remove your import of `Player` in `main.m`. `PlayerManager` should handle *all* communication between `main.m`. That is, `PlayerManager` should handle the creation of players. It should also handle all interaction between `main.m` and these players.

We need to refactor our outputs to include the player's name. Your console should look something like this.



```
2016-09-05 19:51:44.523 Snakes&Ladders[33854:579761] WELCOME TO SNAKES & LADDERS
2016-09-05 19:51:44.524 Snakes&Ladders[33854:579761] Enter the number of players
3
2016-09-05 19:51:48.402 Snakes&Ladders[33854:579761] Type "roll" or "r"
r
2016-09-05 19:51:50.250 Snakes&Ladders[33854:579761] player1 rolled a 4
2016-09-05 19:51:50.251 Snakes&Ladders[33854:579761] Stairway to heaven!
player1 jumped from 4 to 14
r
2016-09-05 19:51:53.570 Snakes&Ladders[33854:579761] player2 rolled a 5
2016-09-05 19:51:53.570 Snakes&Ladders[33854:579761] player2 landed on 5
r
2016-09-05 19:51:56.825 Snakes&Ladders[33854:579761] player3 rolled a 2
2016-09-05 19:51:56.825 Snakes&Ladders[33854:579761] player3 landed on 2
r
2016-09-05 19:52:02.180 Snakes&Ladders[33854:579761] player1 rolled a 2
2016-09-05 19:52:02.180 Snakes&Ladders[33854:579761] player1 landed on 16
r
2016-09-05 19:52:07.164 Snakes&Ladders[33854:579761] player2 rolled a 6
2016-09-05 19:52:07.164 Snakes&Ladders[33854:579761] player2 landed on 11
r
2016-09-05 19:52:10.117 Snakes&Ladders[33854:579761] player3 rolled a 6
2016-09-05 19:52:10.117 Snakes&Ladders[33854:579761] player3 landed on 8
```

Stretch Goals

Adding Score Functionality to `PlayerManager`

Let's start by adding a `score` method to `PlayerManager` that returns an `NSString*`.

Implement this method by looping through the array of players. Call the `score` method on each `Player` instance. The `Player`'s `score` method should also return an `NSString*`.

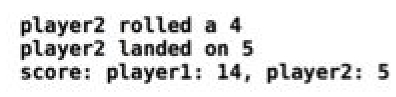
Notice how the manager passes the responsibility of generating the player output string to the `Player`instance.

The manager assembles these score output pieces into a final output string. The player instance's `score` method should return the player's `name` and `currentSquare`.

The `score` method inside `PlayerManager` will create an empty `NSMutableString` before the for loop. Inside the for loop, each player instance's `score` method will return its score string. Append this to the mutable string. Return the mutable string as an immutable string. You should do this because once you generate the final output string you will never need to mutate it.

Let's call `score` on the `PlayerManager` instance inside `main.m` after each roll.

Your output should now resemble this:



```
player2 rolled a 4
player2 landed on 5
score: player1: 14, player2: 5
```

Restarting a Won Game

Add some logic to `main.m` that will start the game again once there is a winner. You can remove all players from the `PlayerManager`'s `players` property to do this. Once there is a winner, trigger another iteration of the while loop using `continue`.

Add Quit

If the user inputs “quit” then exit the app and thank the user. To do this you can call `break` to break out of the while loop. You could also use a BOOL flag value called something like `gameOn`. Use this flag as a parameter in the `while` loop . Change the flag to `NO` if the user types “quit” into the console. Then trigger another iteration of the while loop with `continue`. Test your code.

Make User Input Case Insensitive

Make sure the input you accept from the user is case insensitive. Users should be able to input "Quit" "QUIt", etc and "Roll", "ROLl", etc.

Look at the `NSString` documentation on how to get a lower case version of the input string. When you compare input strings to the expected input, compare a lower case version of the string. Test your work.

Allow the User to Restart

Let's make our quit functionality a bit more fine grained. When the user inputs “quit”, instead of just exiting, ask if they would like to "quit" or "restart".

If they want to restart, remove all objects from the `PlayerManager`’s `players` property. Then call `continue`. This should trigger a request for the number of players in a new game.

If they say "quit" rather than "restart", then the app can call `break` to break from the while loop. Thank them.