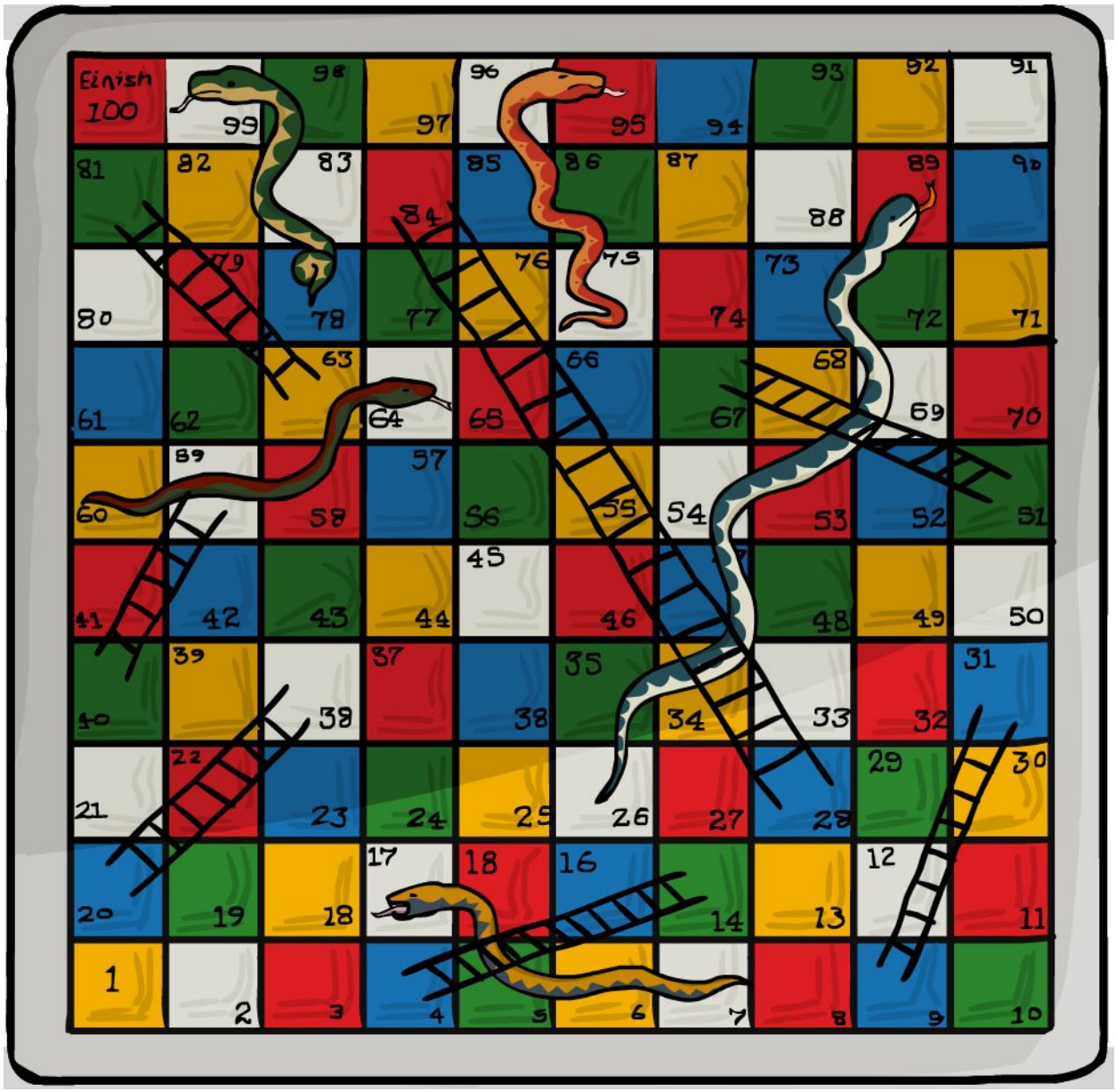
# Snake & Ladders



> **Note:** We recommend that you attempt the *Maths* assignment first before doing *Snakes & Ladders*.

# Rules of the Game

Snakes and Ladders is a simple children's dice game. The goal of the game is to reach the finish square (100) before anyone else.

If a player lands on the bottom of a ladder, they move to the top of the ladder. Bonus!

But, if a player lands on a snake's head, the player moves back to the snake's tail. Bummer!

# Part 1 Learning Outcomes

- Can use `NSNumber` to wrap integers for inclusion in an `NSDictionary`
- Can compare `NSNumber` values
- Can use an `NSDictionary` for modeling moving squares when landing on ladders or snakes
- Can override the default initializer `init`

# Part 1 Goals

- To create a single player command line game called *Snakes & Ladders* that models the kid's game
- The app will wait for the user to type `roll` (or `r` for short) and will roll a random dice value between 1 and 6

- Once the user rolls the app will move the user to a new square
- The user will jump forward or go back depending whether they land on a ladder or snake
- The app will inform the user which square they have landed on and whether they hit a snake or ladder along the way
- When the user passes square 100 the game is over and the game and congratulate them.

# Starter Instructions

Create a command line app called *Snakes & Ladders*. Create a `while` loop inside `main.m` to keep the game going.

Outside the `while` loop let's welcome the user and give them some simple instructions. For now just tell them to type "roll" or "r".

Using `fgets` parse the user's input as we did in the *Maths* app.

# Create a Player Class

Before we get going we need to pause to think about how we should architect this app. To do this it helps to simplify the problem as much as possible.

Let's simplify by setting aside the winning state. We can also set aside the problem of moving squares when the player lands on ladders or snakes.

Let's concentrate on just generating a random dice value. Once we have that we can handle moving from the current square to a new square. We can start

with a simple class structure. Later we can refactor if we find it helpful to split classes further.

Let's start with a simple `NSObject` subclass that we can call "Player".

# Setting Up the Player Class

The Player class needs at least 2 things. An `NSInteger` property called something like `currentSquare` and a `roll` method.

`roll` should return void. To start, let's just create a random value between 1 and 6 in `roll` using `arc4random_uniform()` to do this. Log this dice value to the console.

We only need to create 1 player. So we should do this outside the `while` loop in `main.m`

Check the parsed input string for "roll" or "r" from `main.m`. If the user inputs one of these strings then call the `roll` method on the `Player` instance. Test to see whether roll works.

# Set Up the Player's Basic Movement

We have to set the initial `currentSquare` property to 0. We do this by overriding the `Player`'s `init` method.

When the user types "roll" increment the `currentSquare`'s value inside `roll`. Do this by adding the random value to `currentSquare`. To test this log out the currently rolled value to the console and the incremented value.

# Set Up the Game Logic

The essence of this game is simple. The player rolls a dice. They go to a square whose value is the sum of their current square plus the rolled value. A player can land on certain "special" squares. If they do, they move to a new square, either greater or lesser than the square they just landed on. If they don't land on a special square they stay where they just landed.

We have already modelled the first part in our `Player` class. We just need to model the behaviour of these "special" squares. How do we do this? Spent a couple moments thinking what data stucture (a class, array, dictionary, or set) you would use to store information on snakes. It should also know what square they are on and what square they land on the head. Then read below.

All we need to do is associate one number with another. If we look at the image of the board we see, for example, that there is an association between 4 and 14. So, if the player lands on square 4, they move to square 14.

The best way to model this is a dictionary. In this dictionary we can represent the special squares as keys. The *associated value* of these keys represent the new squares the player moves to.

So, inside our Player's `init` method set an `NSDictionary` property called, something like, `gameLogic`. Set it using the `_` to be sure you are setting the backing store . Do not call `self` inside `init`. Create a key for the foot of each ladder and a key for each snake head. Make each key's associated value correspond to the top of the ladder or the snake tail. Use the *Snakes & Ladders* image (above) to get the required keys and values.

> **Note:** Objective-C collections can only have objects as members. So you will have to wrap these numeric values inside the `NSNumber` class. Use the `NSNumber` literal syntax to do this.

> **Tip:* Once you create the dictionary it is never changed (mutated). Dictionaries and arrays that are not mutated after creation should always be *immutable* types. Use Objective-C's dictionary literal syntax.

# Handle Landing on Snakes or Ladders

Check whether the player has landed on a square corresponding to a key in the dictionary. Do this inside the implementation of `roll`. If the roll matches a key then set the `currentSquare` property to the associated value. If there is no match set `currentSquare` to the new rolled value.

> **Comparing with `NSNumber`:** If you use `==` when comparing object references you will be comparing pointers. This is rarely what you want. You have 2 choices when doing a comparison with `NSNumber`. You can convert both values to primitives and then compare the primitives using `==`. Or you can use `NSNumber`'s method `isEqualToNumber:`. But to use `isEqualToNumber:` both values will have to be NSNumbers.

# Handle the Console Output After Rolling

To inform the user in a console log we need a string to represent what happened as a result of the player's roll.

There are a few ways to handle this. My suggestion is to create a property in `Player` called something like `output`. This should be an `NSString*` type. Set this property from inside the `roll` method.

The output should include the following information: the dice value and the current square. You should also report whether they stepped on a ladder or snake. The output should look something like this:

```
2016-09-05 12:32:29.020 Snakes&Ladders[24090:424191] WELCOME TO SNAKES & LADDERS
2016-09-05 12:32:29.021 Snakes&Ladders[24090:424191] Please type "roll" or "r"
r
2016-09-05 12:32:31.732 Snakes&Ladders[24090:424191] You rolled a 2
2016-09-05 12:32:31.732 Snakes&Ladders[24090:424191] You landed on 2
r
2016-09-05 12:32:34.476 Snakes&Ladders[24090:424191] You rolled a 6
2016-09-05 12:32:34.477 Snakes&Ladders[24090:424191] You landed on 8
r
2016-09-05 12:32:36.348 Snakes&Ladders[24090:424191] You rolled a 3
2016-09-05 12:32:36.348 Snakes&Ladders[24090:424191] You landed on 11
r
2016-09-05 12:32:41.930 Snakes&Ladders[24090:424191] You rolled a 5
2016-09-05 12:32:41.930 Snakes&Ladders[24090:424191] You landed on 16
r
2016-09-05 12:32:44.917 Snakes&Ladders[24090:424191] You rolled a 2
2016-09-05 12:32:44.917 Snakes&Ladders[24090:424191] You landed on 18
r
2016-09-05 12:32:47.451 Snakes&Ladders[24090:424191] You rolled a 1
2016-09-05 12:32:47.451 Snakes&Ladders[24090:424191] You landed on 19
r
2016-09-05 12:32:50.939 Snakes&Ladders[24090:424191] You rolled a 3
2016-09-05 12:32:50.939 Snakes&Ladders[24090:424191] You landed on 22
r
2016-09-05 12:32:52.082 Snakes&Ladders[24090:424191] You rolled a 6
2016-09-05 12:32:52.083 Snakes&Ladders[24090:424191] Stairway to heaven!
You jumped from 28 to 84
```

# Handle the Game Over State

The game over state is either true or false. So, we should add a `BOOL` property to the `Player` called `gameOver`. If unset, a `BOOL` value is set to `NO`. But it's not a

bad idea to make our intention explicit. Therefore, set our `gameOver` property to `NO` in the `Player`'s `init` override.

Inside `roll` we are going to want to check to see if our random dice value takes us up to, or past, square 100. If it does, we can set the `output` property in `Player` to something appropriate.

In `main.m` we can check to see if, after our roll, `gameOver` is `YES`. If it is, we can output the message and `break` out of our `while` loop.