# Backbone Stickit Homework

## Introduction

In this exercise, we will make a hangman game. The player is given a random word and has to guess the letters that make up the word. After five wrong guesses, the game is over.

The fully functional game can be found here here. Play around to get a feel with what you will be building.

**Fully functional version (in minified JavaScript)** http://jsfiddle.net/xe9mzvd6/6/

For this exercise, templates and styling have already been created for you, as well as the model and a skeletal view, where you will apply Stickit bindings.

**Your starting point:** http://jsfiddle.net/xe9mzvd6/5/

The model describes the state of the game, which is dictated by the following variables:

- The current word being guessed
- The sequence of guessed letters
- Whether the game is finished (either by too many incorrect guesses, on word completion, or when the player has decided to reveal the word).

## Question 1

We use a string rather than a character array to represent the sequence of guessed letters. Why is this advantageous?

*Hint: consider the identities of `a` and `b` in the following cases*

```
// Case 1
var a = 'some string';
var b = a;
b = 'other string';     // What is the value of b == a?

// Case 2
var a = [ 'first', 'second', 'third' ];
var b = a;
b.push( 'fourth' );     // What is the value of b == a?
```

## Question 2

In `template__game-view`, notice that `.js-display` is currently an empty `div` element. We will render the characters for the word with styling that reflects the character's state ( `unguessed`, `guessed`, `revealed` ) here.

**(a)** Study the Game model. Which attributes must `.js-display` observe?

**(b)** The template `template__characters` has been prepared for you to render the characters. What object structure would you need to pass to a template function derived from this template?

**(c)** Create a binding for `.js-display`, to observe the attributes you need and implement `onGet` to render the template correctly.

- Each character element's class list should be `'correct'`, `'revealed'` or `''`.
- The display character should be `'?'` if neither the character is correctly guessed nor is the game finished.

*Hint: study the model's `getCharacterStates()` function.*

**(d)** In addition to `observe` and `onGet`, there is another binding parameter that is needed for the correct rendering. What is it?

## Question 3

In `template__game-view`, `.js-guess-history` is also an empty `div` element. We want to render the characters that have been guessed with the styling that reflects the correct and incorrect guessing.

Create a binding for `.js-guess-history` to observe the attributes you need (which?) and implement `onGet`. Each character element's class list should be 'correct' or 'incorrect'.

*Hint: study the model's `getGuessStates()` function.*

## Question 4

`.js-guess` is the textbox that allows you to enter a character to guess. It clears when you type a character and press `enter`. The character typed, if a valid character (from `a` to `z`), will be appended to the sequence of guessed characters.

**(a)** Do we need a uni-directional or a bi-directional binding?

**(b)** We don't want the model to update until the player focuses out of the textbox or hit `enter`. What extra parameter must be defined in the binding for this to happen?

**(c)** We don't want the model to update unless a valid character is given. How do we ensure this?

**(d)** Create a binding for `.js-guess` using the answers you have given for parts **(a - c)**

*Hint: study how the model reacts to the change in the guess sequence.*

## Question 5

At this point, we should have a fully functional application. However, there are certain user actions that can cause buggy behaviours:

- If the Internet connection is slow, it is possible to type guesses into the textbox while the word is being

- The player can keep guessing even if the game is finished.
- The reveal button is clickable even after the game is finished.

**(a)** Bind the visibility of `.js-container`, which is the container of the entire application, and make it hide while the word is being fetched.

*Hint: The current word is reset to* `''` *while a new word is being fetched*

**(b)** Bind the `disabled` attribute of `.js-new-word` and `.js-reveal` so that the elements cannot be interacted with when the game is finished.

## Solution

http://jsfiddle.net/xe9mzvd6/3/