Architectural Design

The general structure for the implementation of the solution has been based on MVC. The different files has been stored in the folder (/Controller, /Model, /View).

- **On the Model side:**
  - We have a ScriptableObject called GameDataModel to keep the game's information. The use of ScriptableObject is recommended when storing and managing data.
  - To translate the number to text we define an extensible class *(meeting open-closed SOLID principle)* with an interface *(meeting interface segregation and dependency inversion principles)*. That to that structure the rest of the system won't need to know what is the provider since its communication is through the interface:
    - Class *NumberToTextProviderEnglish*: Base behaviour that transform a number to english text. (dependency inversion principle)
    - Class *NumberToTextProviderSpanish*: Extends the previous behaviour to deal with the particular cases of Spanish languages like (quinientos, setecientos, etc…) (open-closed principle)
- **On the Controller side:**
  - General controller called **MainAppController.cs** that will be responsible to run the different game states and offer access to the multiple resources.
  - For the game states we use the command design pattern.
    - Each the commands deal with a *single responsibility* (SOLID principle)
    - All these commands share the same minimal interface meeting the *interface segregation* (SOLID principle). This way the MainAppController.cs only will need that interface with only the essential functionality.
    - Class *GameStateMenuCommand*: Only responsible of managing the state "Menu"
    - Class *GameStateShowNumberCommand*: Only responsible of managing the state "Show Number"
    - Class *GameStateAnswersCommand*: Only responsible to manage the state "Select Answer".
- **On the View side:** We have a collection of view responsible to display the information and collect the interactions that will be reported to the Controllers.
  - Class *ScreenMainMenuView*: View responsible to display and collect the interaction of the "Menu" state.
  - Class *ScreenShowNumberView*: View responsible to display and collect the interaction of the "Show Number" state.
  - Class *ScreenAnswerNumberView*: View responsible to display and collect the interaction of the "Select Answer" state.
  - Class *HUDPlayGameView*: View responsible to display the information about the game's score.
  - Class *ItemNumberView*: View that represent a number that can be interacted with.

Conclusions:

With the MVC structure we have a system flexible enough to deal with different requirements. So far the application can be customized:

- Total of number of the numbers to find (to a maximum of 3)
- Total possible answer to show (to a maximum of 7)
- Max value to generate a number (to a maximum of 100000000)
- Language used in the system ("es" (spanish), "en" (english))