First of all, we adopted the Model-View-ViewModel(MVVM) as our design pattern. That mean for the view is control by the ViewModel and there is a Model to control the data.

In “ShopViewModel.cs”, inside the method “RandomPokemon()”, there is a line “PokemonModel pokemon = RandomPokemonMethod();”. The RandomPokemonMethod() is a factory method to random generate and initialize the Pokemon to the field “pokemon” without having logic inside the method “RandomPokemon()”.

Inside all the field of each of the ViewModel is all private with a properties to access it, it can prevent and easily control which field can be control and which field can be access.

Inside the MainWindowViewModel which is the “most important” ViewModel mainly is to control the which view is display and which ViewModel is used. Therefore, most of the GoTo\_\_\_ViewModel (except the BattleViewModel since a timer need to be set), all the GoTo\_\_\_\_ViewModel only have two line, which is only control what is the CurrentViewModel and what is the CurrentView. When we need to Update the ViewModel, we will update inside other ViewModel. For example, when we need to update the Player in the GymViewModel, we will called (GymViewModel)MainWindowViewModel.GymViewMolde.UpdatePlayer(Player), to update the Player inside the GymViewModel.

By our MVVM design, when a View needed to be added, we only need to initialize it inside the MainWindowViewModel and if needed initialize the ViewModel inside the MainWindowViewModel. All the logic can be implement inside the new ViewModel and draw the View. This design allows us to update and add the View and ViewModel easily.