Hello and welcome to this presentation on the Model View Controller Pattern (MVC). I, James Moran, will be giving this presentation, let’s begin.

What is the MVC? It separates modelling of the logical-framework, visual-presentation and handling user-input, into three separate design aspects for a project. The Model handles the business-logic of the application, responding to requests for information from the View, given the current state, as well as any commands to change state (such as the user clicking on a button in the View), from the Controller. The View is the visual aspect for the application, in the case of the Game Café, showing a form with various controls, that show information to the user, garnered from the Model (such as Membership Information) and allow traversal through the application (via form navigation buttons, to access a certain part of the system).

Considering the Game Café as an example, the Model is initiated via the main entry-point into the application (initialising the root-form of the system), as well as having a representation of the Database and its tables, for the information that is stored about certain aspects of the Game Café (e.g. for Members, Bookings and Hardware available at the Game Café). These are represented as classes in the application, with the details in respect to the class name, stored in the class, along with another class (IDUpdateSystem), to handle updating the ID for each of these classes, whenever a new entry is added to the database. Next comes the View of the system, with the example of the GameCafeRootForm shown here, that is displayed to the User, to allow them to interact with the system (in this case, there is a title and three buttons that allow them to traverse to the form for viewing the respective Member, Booking or eSports Event Information on that form). This leads into the Controller design-aspect, with the logic for the form visual-aspect, being auto-generated by Visual Studio, along with event-handlers for certain controls on this form, (e.g. for when any button on the form is pressed), as well as event-handlers for the form itself (such as when the form is closing). The logic for these events, is contained within a class for the form (with the same name as the form), for what occurs when these events are raised (whether that is hiding the root form and showing another form, or checking to see if the User is closing this form, asking them to confirm their choice).

The advantages of Use-Case Modelling, are detailed here, with the Advantages being that of a User-Centred Technique (ensuring that the correct system is developed for the User), easy for the Customer/User to understand (as they are composed in a natural-language form, providing an excellent way to communicate with them). As well as providing an Objective Means to track the project (by deriving earned value from the implementation and delivery of them). The potential disadvantages of their use though, are that of Potential Redundant Classes (as a Use-Case can span across the use of multiple classes, given Object-Orientated-Design), the use of a different paradigm to the Object-Model (with unclear mapping between the Use-Case Model’s structure and the Object-Model’s network structure) and Poor Scalability (not providing enough Use-Cases for an adequate specification or using too many, leading to the functional decomposition of objects and classes).

This slide details the references used in this presentation, thank-you for watching and goodbye for now.