SDV602 Assessment Milestone 1

Tim Gentry

4/09/2017

Contents

[1. Summary 3](#_Toc492331239)

[2. Game Concept Design 4](#_Toc492331240)

[2.1 Design Considerations 4](#_Toc492331241)

[2.2 Design Development 4](#_Toc492331242)

[3. Game Interface Design 5](#_Toc492331243)

[4. Game System Design 7](#_Toc492331244)

[4.1 Use Case Diagram 7](#_Toc492331245)

[4.2 Functional Test Cases 10](#_Toc492331246)

[4.3 Architectural Pattern 12](#_Toc492331247)

[4.4 Analysis Class Diagram 13](#_Toc492331248)

# 1. Summary

This report covers the work done for Milestone One, which covers three areas of the game’s development.

The first section covers development of the game’s concept. How the game will play, what the goal is and what the player will be expected to do.

The second section covers development of the game’s user interface. How the player will see the game and how they will be able move between different views within the interface.

The third section covers the development of the game’s system. Covering the identification of functionality required for the player to be able to play and then brief design of the system’s structure, examining the individual components of the system and how they all relate to each other.

The project files used to develop the application for this project should appear alongside this report in the file directory or are alternatively available from the GitHub repository found at

**https://github.com/tgentrynz/SDV602-Assessment**

# 2. Game Concept Design

## 2.1 Design Considerations

The basic idea behind the game is a text based adventure game. Through research of games that fit similar descriptions the following key points were identified as the focus of this game’s system.

* The user will control the game through text input.
* The game should simulate an environment and allow the user to move between locations within this environment.
* The game should allow for interactions between the player and objects within the environment to create some form of puzzle mechanic.

## 2.2 Design Development

Based on these considerations and research of similar game concepts, it was decided that the game for this project would focus on an “escape room” scenario.

The environment for this kind of game would take the form of some indoor space with an exit that is sealed based on some condition. The players’ goal would be to move around the environment and find the correct key item to unseal the exit and escape the enclosed environment.

Despite the name, “escape room”, the environment for this game should simulate a building or a dungeon, some larger structure that can be made of more than a single room. This allows for more player movement, creating multiple stages for puzzles which reduces the chance of them becoming cluttered and keeps the objectives clearer.

The following section provides an example of how this game concept could play.

The player could start in a room directly facing the exit, which takes the form of a locked door. Other than the door, there is a lever in the room, but its handle is missing. There is an existing exit, which leads to a side room.

Being unable to use the locked door or broken lever, the player will move to the side room, which contains a lever handle. Returning to the first room, the player can use the lever handle with the lever. This doesn’t unlock the exit, instead opening a secret doorway.

Entering through the secret doorway, the player enters a room that contains a key. After returning to the first room, the player can use the key on the exit door to unlock it. Allowing them access to the victory area.

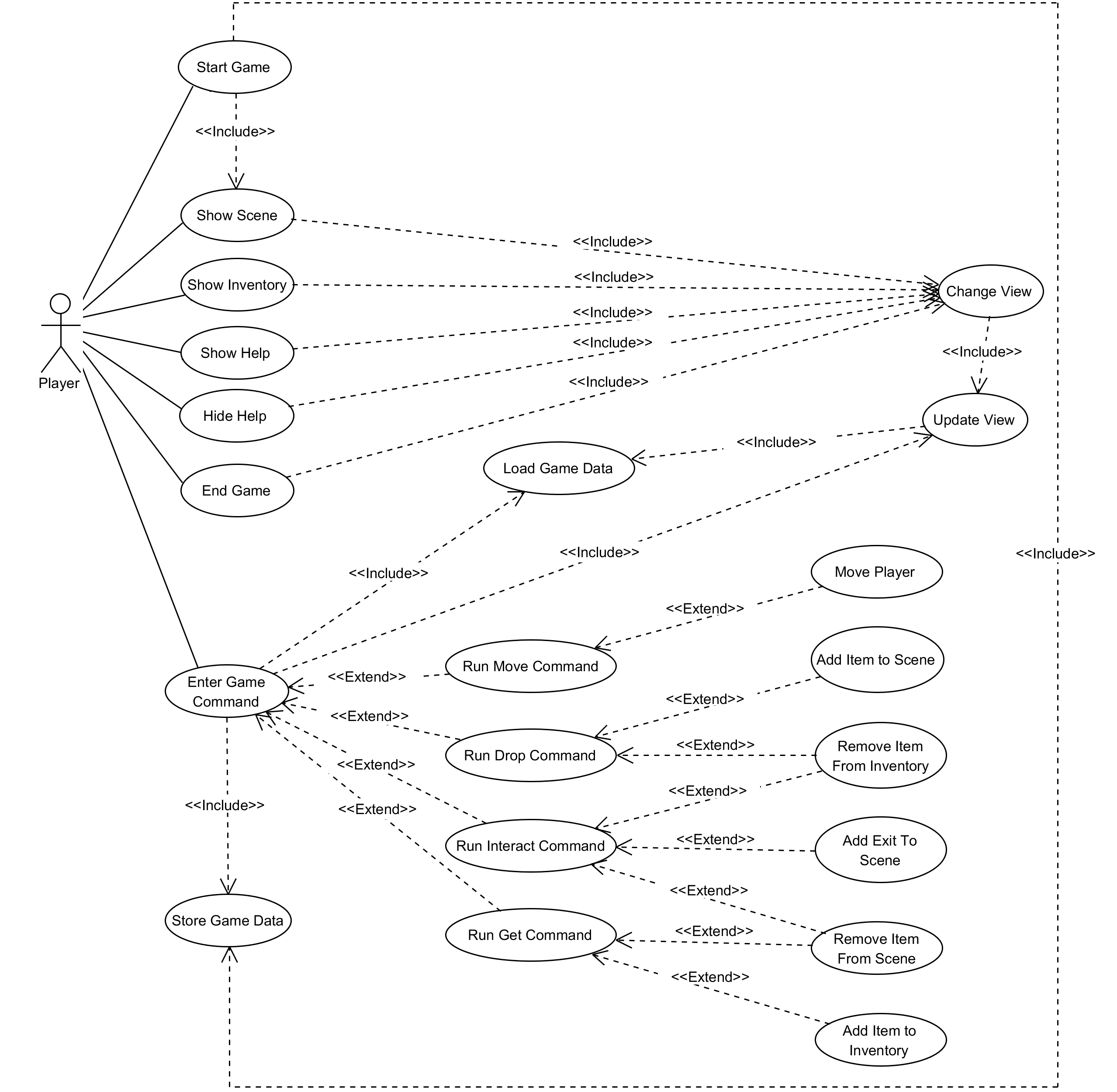
# 3. Game Interface Design

|  |  |
| --- | --- |
| C:\Users\TimTen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\titleScreen.png | **Board 1 – Title Screen**  This is the first screen visible to the player upon opening the game.  The main content on the screen shows the name of the game and provides a list of commands the player can enter to start the game, receive further instruction or end the game.  The bottom of the screen holds two text boxes, one is read only and displays feedback to the player’s inputs and the other is the text box the player uses to enter inputs.  Typing “start” will cause the application to proceed to **Board 2**, typing “help” will have the application proceed to **Board 4** and typing “exit” will close the application. |
|  |  |
| C:\Users\TimTen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\SceneScreen.png | **Board 2 – Scene Screen**  This screen is how the player views the game environment.  The main content on the screen provides a description of the location, a list of items present in the location and a list of exits to other locations in the game environment. It is backed by an image the represents the game environment.  The bottom of the screen holds two text boxes, one is read only and displays feedback to the player’s inputs and the other is the text box the player uses to enter inputs.  Typing “show items” will cause the application to proceed to **Board 3**, typing “help” will have the application proceed to **Board 4** and typing “exit” will return to **Board 1**. |
| C:\Users\TimTen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\InventoryScreen.png | **Board 3 – Inventory Screen**  This screen displays the items that a player has in their inventory.  The main content shows a list of items that are in the player’s inventory backed by an image of a bag.  The bottom of the screen holds two text boxes, one is read only and displays feedback to the player’s inputs and the other is the text box the player uses to enter inputs.  Typing “show room” will cause the application to return to **Board 2**, typing “help” will have the application proceed to **Board 4** and typing “exit” will return to **Board 1**. |
|  |  |
| C:\Users\TimTen\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HelpScreen.png | **Board 4 – Help Screen**  This screen displays a list of commands the player can use to operate the game.  The main content shows a list of how each command is typed and what it does.  The bottom of the screen holds two text boxes, one is read only and displays feedback to the player’s inputs and the other is the text box the player uses to enter inputs.  Typing “close” will cause the application to return to **Board 1, 2 or 3**, depending on the screen they were on prior to this one. |

# 4. Game System Design

## 4.1 Use Case Diagram

To operate the game, users need to be able to switch between views, enter game commands and see updates to the game information. These broad use cases were further developed into the following use case diagram.



The following section contains a description for each use case presented in the diagram.

**Start Game**

This use case is accessible from the main menu view. A user types “start” into the input field and a new game instance is created, so the **Store Game Data** use case is included. After a game instance is created, the UI changes so that is now displaying the information about the game scene, this requires that the **Show Scene** use case is included.

**Show Scene**

This use case is directly accessible to the player from the inventory view. A user types “show scene” into the input field and the view changes to the scene view by including the **Change View** use case.

**Show Inventory**

This use case is accessible to the player from the scene view. A user types “show items” into the input field and the view changes to the inventory view by including the **Change View** use case.

**Show Help**

This use case is accessible to the player from all of the other views. A user types “help” into the input field and the view changes to the help view by including the **Change View** use case.

**Close Help**

This use case is accessible to the player from the help view. A user types “close” into the input field and the view returns to the view the user was on before accessing the help menu by including the **Change View** use case.

**End Game**

This use case is accessible to the player from the scene view or the inventory view. A user types “exit” into the input field and the display is changed back to the main menu view by including the **Change View** use case.

**Change View**

This use case is accessible to the player from one of the graphic user interface focussed use cases, **Show Scene**, **Show Inventory**, **Show Help**, **Close Help** or **End Game**. The graphic user interface changes to show the view that the user requested. To make sure that the screen is updated with the most recent information, the **Update View** use case is included by this one.

**Update View**

This use case is accessible to the player from use cases that change information being viewed by the player, **Change View**, or use cases that need to display changes to the game data, **Enter Game Input**. By including the **Load Game Data** use case, it retrieves the most up to date game data and then changes the information presented on the user interface elements.

**Enter Game Command**

This use case is accessible to the user from the scene and inventory views. A user types one of the game environment manipulation commands into the input field. The system then loads the data that needs to be changed through the inclusion of the **Load Game Data** use case. Following this, the specific command entered is performed by one of the extending use cases, **Run Move Command**, **Run Drop Command**, **Run Interact Command** or **Run Get Command**. After the game data is changed the display is updated to show the changes by including the **Update View** use case and then saved through the inclusion of the **Store Game Data** use case.

**Run Move Command**

This use case is accessible to the user by extension of the Enter Game Command use case. The system checks that the game scene the user would like to move to is accessible and, provided it is, the game system updates the player’s position through extending to the **Move Player** use case.

**Run Drop Command**

This use case is accessible to the user by extension of the Enter Game Command use case. The system checks that the item the user would like to drop is in their player inventory and, provided it is, the game system removes the item from the player’s inventory through extending to the **Remove Item from Inventory** use case and places it in the current game scene through extending to the **Add Item to Scene** use case.

**Run Interact Command**

This use case is accessible to the user by extension of the Enter Game Command use case. The system checks the door item the user would like to interact with is present in the current scene and the key item required by it is present in the player’s inventory then, provided these conditions are met, the game system removes the key item from the player’s inventory through extension to the **Remove Item from Inventory** use case, removes the door item from the game scene through extension to the **Remove Item from Scene** use case and finally adds a new exit in the game scene through extension to the **Add Exit to Game Scene** use case.

**Run Get Command.**

This use case is accessible to the user by extension of the Enter Game Command use case. The system checks that the item the player wants to pick up is present in the current game scene and, provided it is, the game system removes the item from the game scene through extension to the **Remove Item from Scene** use case and then add that item to the player’s inventory through extension to the **Add Item To Inventory** use case.

**Move Player**

This use case is accessible to the user by extension of the **Run Move Command** use case. It changes the game scene that the player is positioned in within the game environment.

**Add Item to Scene**

This use case is accessible to the player by extension of the **Run Drop Command** use case. It adds an item to the current game scene.

**Remove Item from Scene**

This use case is accessible to the player by extension of the **Run Interact Command** and **Run Get Command** use cases. It removes an item reference from the current game scene.

**Add Item to Inventory**

This use case is accessible to the player by extension of the **Run Get Command** use case. It adds an item reference to the player inventory data.

**Remove Item from Inventory**

This use case is accessible to the player by extension of the **Run Drop Command** and **Run Interact Command** use cases. It removes an item reference form the player inventory data.

**Add Exit to Scene**

This use case is accessible to the player by extension of the **Run Interact Command** use case. It adds a reference to another game scene to the current game scene.

**Load Game Data**

This use case is accessible to the player through inclusion by the **Enter Game Command** and **Update View** use cases. It retrieves the game’s information from the data storage.

**Save Game Data**

This use case is accessible to the player through inclusion by the **Start Game** and **Enter Game Command** use cases. It saves game information to the data storage.

## 4.2 Functional Test Cases

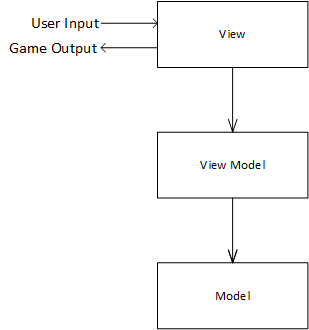
To develop a better understanding of the functionality behind the use cases, some initial test cases were created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Tested** | **Use Case** | **System State** | **Input** | **Output** |
| Player starts game. | Start Game | Game is on the title screen. | Player types “start” into input field and presses the enter key. | The game starts and the view changes to the scene screen. |
| Player views the inventory. | Show Inventory | Game is on the scene screen. | Player types “show items” into the input field and presses the enter key. | The game view changes to the inventory screen. |
| Player views the scene. | Show Scene | Game is on the inventory screen. | Player types “show room” into the input field and presses the enter key. | The game view changes to the scene screen. |
| Player views the help screen. | Show Help | Game is on any of the non-help screen screens. | Player types “help” into the input field and presses the enter key. | The game view changes to the help screen. |
| Player closes the help screen. | Close Help | Game is on the help screen. | Player types “close” into the input field and presses enter | The game view changes to the screen prior to the help screen being opened. |
| Player moves. | Run Move Command | Game is running and player is not looking at the help screen. | Player types “goto” and an exit name into the input field and presses enter. | The current scene is changed and the scene screen is updated. |
| Player picks up an item. | Run Get Command | Game is running and player is not looking at the help screen. | Player types “get” and an item name into the input field and presses enter. | The item is removed from the list of items in the scene and added to the list of items in the player’s inventory. |
| Player drops an item. | Run Drop Command | Game is running and player is not looking at the help screen. | Player types “drop” and an item name into the input field and presses enter. | The item is removed from the list of items in the player’s inventory and added to the list of items in the scene. |
| Player uses an item. | Run Interact Command | Game is running and player is not looking at the help screen. | Player types “use” and an item name into the input field and presses enter. | The door item is removed from the list of items in the scene, the door’s key item is removed from the list of items in the player’s inventory and the door’s referenced game scene is add to the list of exits in the current scene. |

## 4.3 Architectural Pattern

To help organise the system behind the game, some elements of the **Model-View-View Model** pattern were used. This involves separating different areas of functionality into individual sections of the codebase. This allows for the different functions of the application to be developed independently of each other. The specific functions that are separated are the **Model,** which represents functionality related to storing and retrieving data; the **View**, which represents functionality related to user interactions; and the **View Model,** which represents the communication functionality between the view and the model.

The relationship between these three sections of the codebase works as follows.

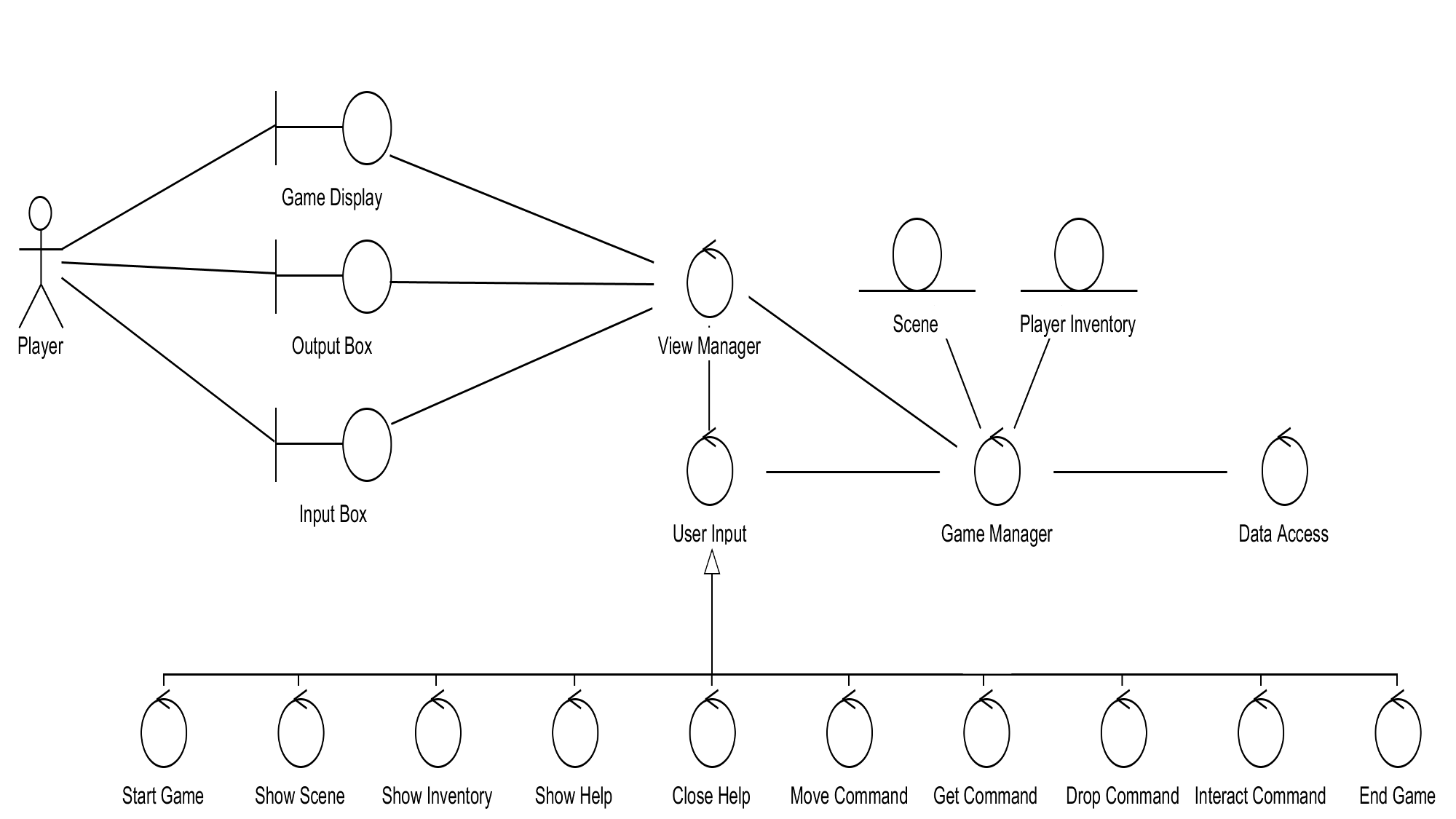


As shown here, the **View** is the aspect of the program that the user interfaces with. It will cover use cases related to the user interface, such as Start Game, Show Scene, Show Inventory, Show Help, Close Help, Change View, Update View and Enter Game Command. The view communicates with **View Model**, which is responsible for the game manipulation use cases, such as Run Move Command, Run Get Command, Run Drop Command, Run Interact Command, Move Player, Add Item to Scene, Add Item to Inventory, Add Exit to Scene, Remove Item from Scene and Remove Item from Inventory. The view model communicates with the **Model**, which handles the data access use cases, Load Game Data and Store Game Data.

These relationships are unidirectional so that the **Model** does not need information about the **View Model** and the **View Model** does not need information about the **View**. It should also work so that the **View** only needs to know enough about the **View Model** to request data from it and shouldn’t need to know anything about how the **Model** operates.

## 4.4 Analysis Class Diagram

Building on the use cases, a map of the potential system was created to help develop the separation of responsibilities and relationships between system components.



The following section details each element presented in this diagram.

**Game Display**

This is a boundary class that represents the user interface elements that the player uses to view information about the game environment. It is updated by the **View Manager**.

**Output Box**

This boundary class represents the user interface elements that provide feedback to the player’s inputs. It is updated by the **View Manager**.

**Input Box**

This is a boundary class that represents the user interface elements that the player uses to enter input into the system. It updates the **View Manager**.

**View Manager**

This is a controller class that manages the passing of input and output between the user interface classes and the game manipulation classes. It has direct access to the **Game Display**, **Output Box** and **Input Box** classes for user interface processes and direct access to the **Game Manager** class for game information processes. It is also in charge of the creation of **User Input** instances for game manipulation processes.

**User Input**

This is a controller class that manages the changes to the system made by user inputs. It can update both the **View Manager** and **Game Manager** classes. The different realisations of this class, **Start Game**, **Show Scene**, **Show Inventory**, **Show Help**, **Close Help**, **Move Command**, **Get Command**, **Drop Command**, **Interact Command** and **End Game** relate to specific use cases and alter the data in different way. However, the way they communicate with the other components remains the same.

**Game Manager**

This controller class manages changes to the game data. It is updated by the **User Input** class, can send updates to the **View Manager** class and request updates from the **Data Access** class. It can also read and write **Scene** and **Player Inventory** classes.

**Scene**

This entity class holds information about a game scene.

**Player Inventory**

This entity class holds information about items a player has collected.

**Data Access**

This controller class handles requests from the **Game Manager** class for retrieval of game data from the data store.