1 My Framework

This game is built over GFFramework, my own framework that I have been developing for the past few weeks. It works as a layer over Unity to speed up the games development process, it's still a work in progress.

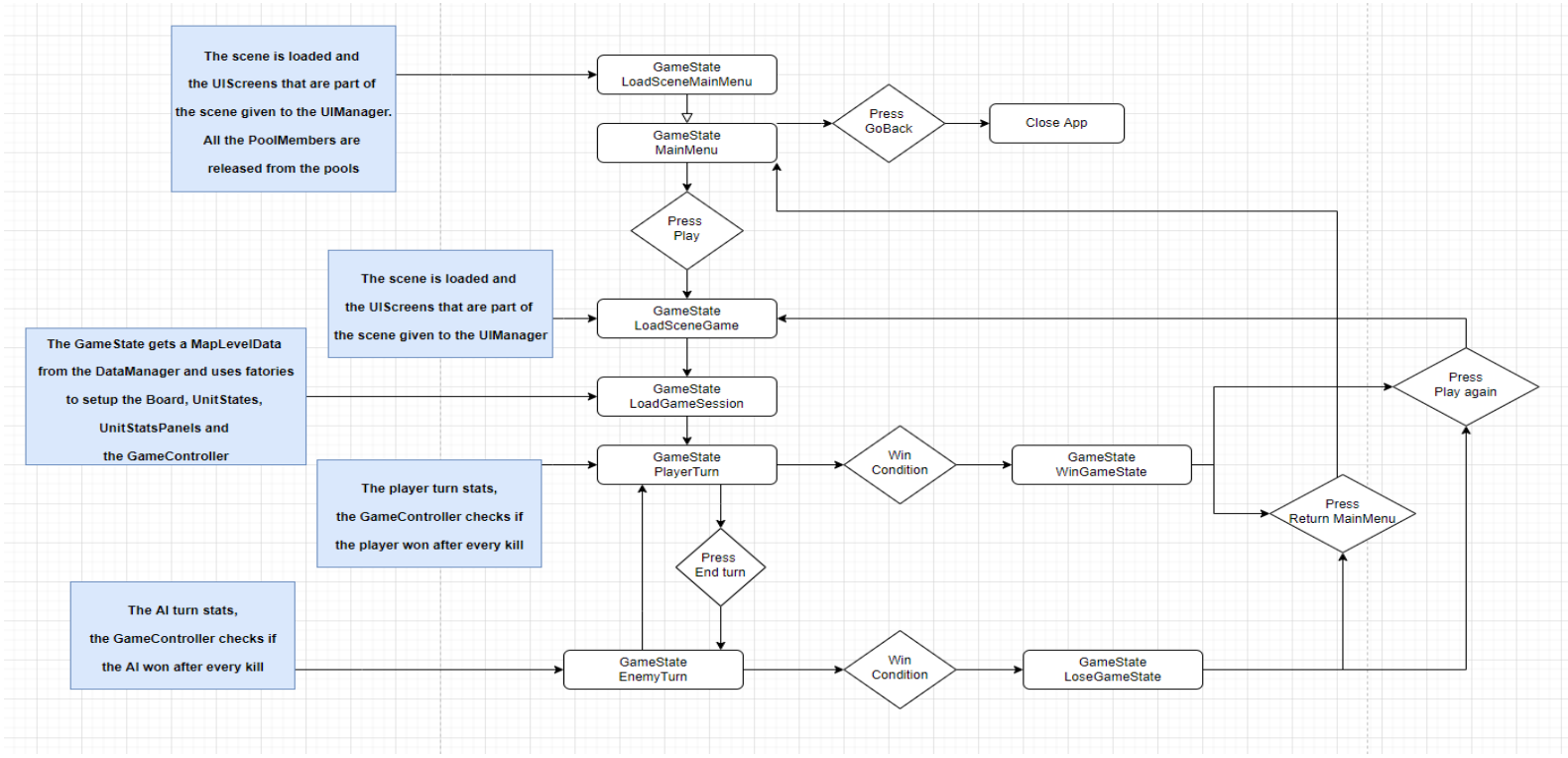
It's designed to use the GameInitializer to load only the managers needed in a similar way to how Unreal modules work. Only the GameStateManager or other implementation of IGameStateProvider is required by the GameInitializer.

All the parts of the framework are going to be decoupled, you don't have to use the whole framework or all the managers. The GameStates are the glue that makes all the systems work, but no manager (UIMan ,InputMan, CameraMan...) is going to be required by others.

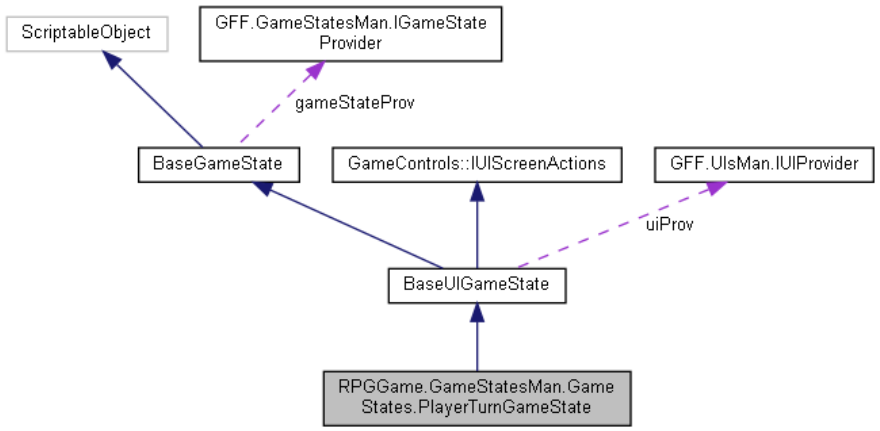
Right now there are some dependencies between the UIManager and the GameState manager. In the next few weeks, I´m going to create an Assembly definition for each manager folder and complete the decoupling of the managers from each other.

1.1 GameStates system

The game's flow is a State pattern. Each state of the game is a ScriptableObject, which helps to create as many states as you want easily.



Here you can see the GameStates flow (you can find the PNG in the same folder).



Here you can see an example of a GameState, the PlayerTurnGameState.

Each GameState has this important methods:

Setup(): Entry method where the dependencies of the GameState components are resolved (Dependency Injection Composition root), the components can also start to listen to the events that they need here.

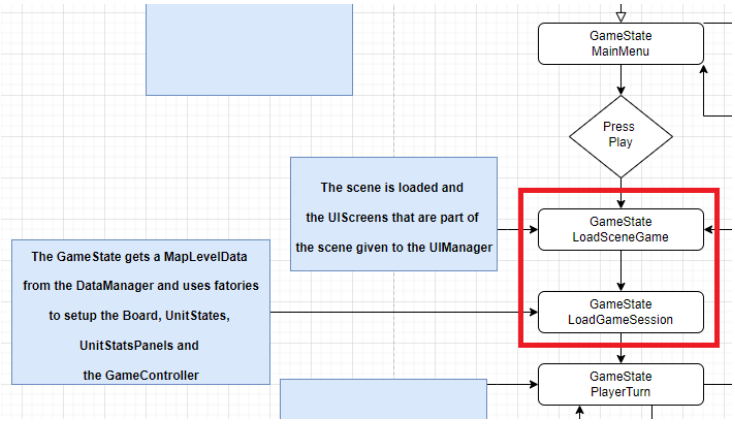
Unsetup(): Exit method where all the resources taken by the game state are disposed and stop listening to events.

The **GameStateManager** is responsible for loading and unloading each GameState.

BaseUIGameState is an important GameState that is worth mentioning. It's the base class that gets a UIScreen (Prefab [Canvas +UI components]) from the UIManager and shows it to the player.

2 The Game

2.1 The Game Initialization



These two states handle the game initialization.

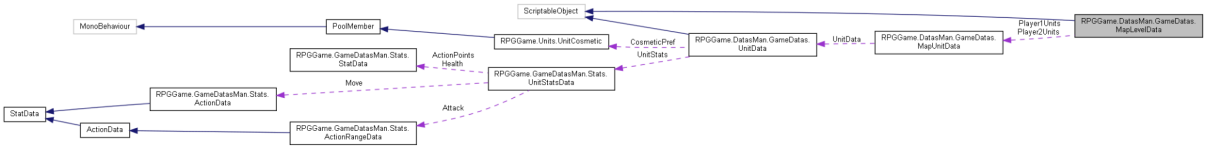
2.1.1 LoadSceneGameState

Loads a Scene, sends the UIScreens that are part of this scene to the UIManager and moves to the next GameState.

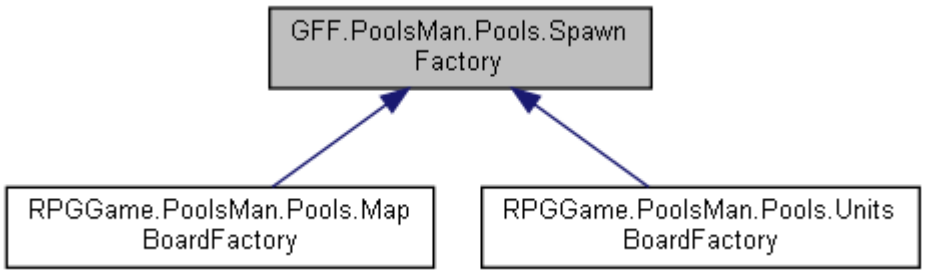
2.1.2 LoadSessionGameState

Builds the play session/game match. It gets a MapLevelData and creates:

- The **Board** and its **Cells** using a MapBoardFactory.
- The **UnitStates** (soldiers and monsters) using a UnitsBoardFactory.
- The **PlayerControllers** that handle the state of a player (e.g: soldiers alive).
- The **GameController** that handles the match state which player wins.



Here you can see the MapLevelData dependencies (you can find the PNG in the same folder).

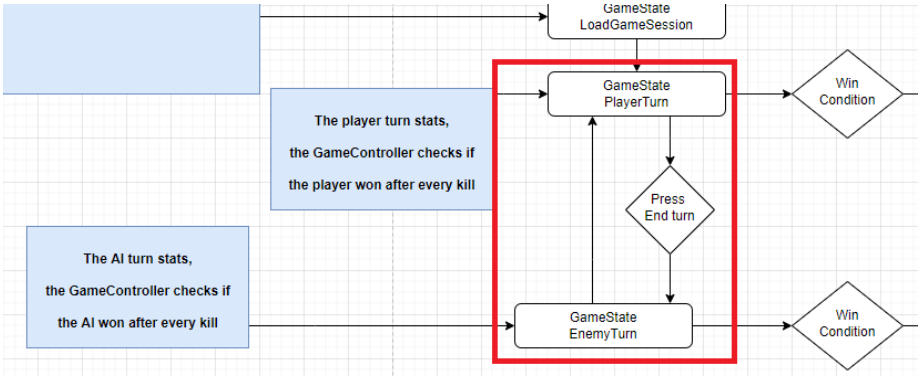


The Factories use the PoolManager to get the cells, UnitStates, UnitConsmetics and UIPanelUnitStats. Then everything is injected, wired and returned.

After this, the GameController is ready and it's given to the **SessionManager** that starts the game session.

2.2 The Game Loop

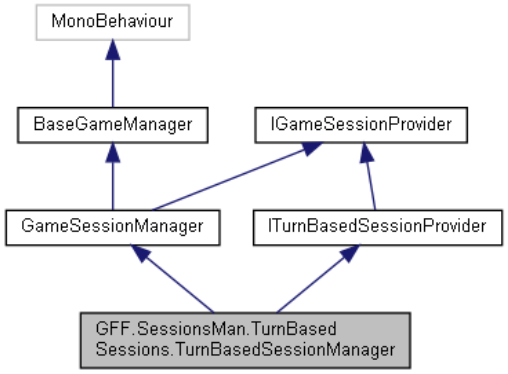
**Any arrows that looks like Diamond Inheritance are just Interfaces*
**After the* LoadSessionGameState *is done injecting dependencies, every class is referenced using Interfaces.*



These two states handle the game loop.

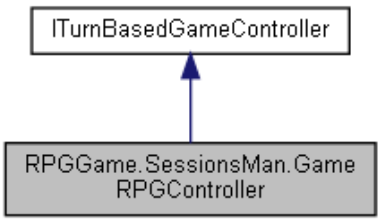
2.2.1 TurnBasedSessionManager

Acts as connection point between the players and the GameController and provides some utilities like pause the game.



2.2.2 GameRPGController

Responsible for the rules, actions, who wins and state of a PlayerRPG vs PlayerRPG match. It handles the interactions between the PlayerRPG UnitStates and the Board.

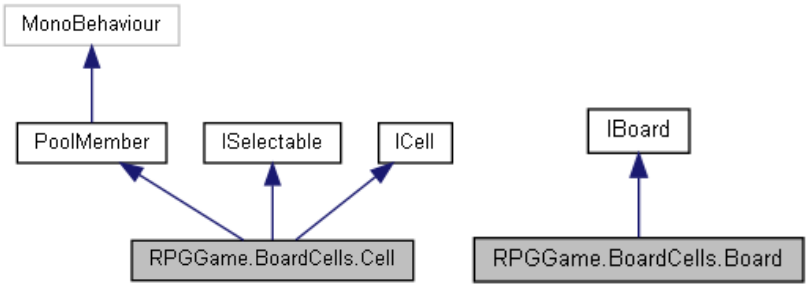


Public Member Functions

	GameRPGController (**IBoard** board, **IPlayerRPG** player1, **IPlayerRPG** player2)
void	**StartGame** ()
void	**EndGame** ()
void	**OnSelectCell** (**ICell** cell)
bool	**StartTurn** (bool isPlayer1, Action onWinGameCallback)
bool	**EndTurn** (bool isPlayer1)

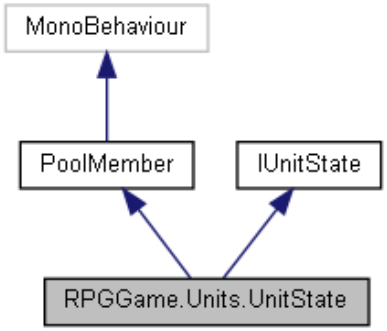
2.3 game Board and Cells

The Board provide utilities to remove and add IUnitState from the Cells



2.3 UnitStates

Current state of a Unit (e.g: Soldier, Monster), it acts as a facade (pattern) to connect the Cosmetic, Transform and UnitStatsState (life, attack, actionPoints...)

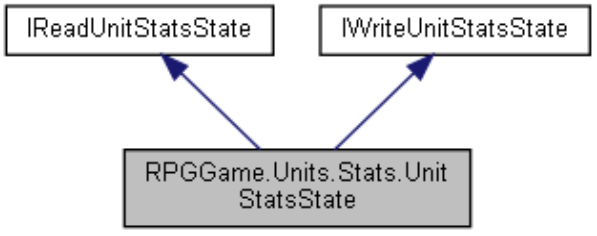


2.4 UnitStatsStates

Current state of the Unit stats, their initial values are loaded from a **UnitStatsData**.

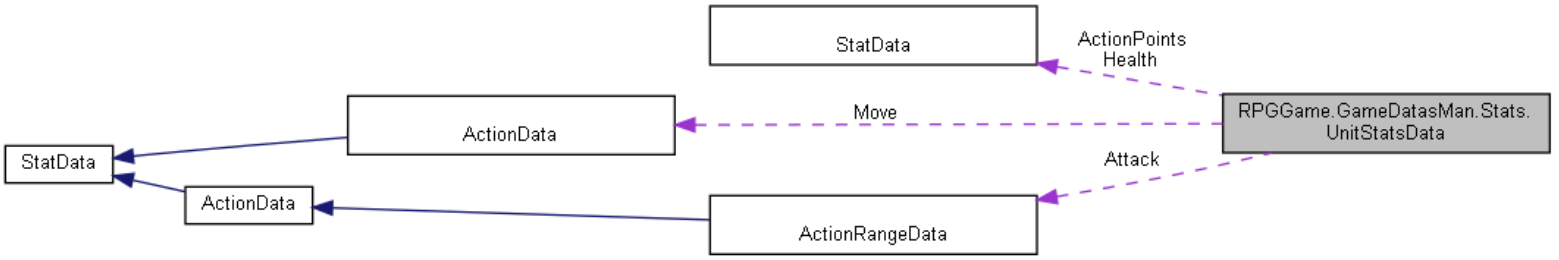
IReadUnitStatsState provides to the UI (or any other listeners) only methods that don't change the stats state.

IWriteUnitStatsState on the other hand provides to the UnitState just the methods to change the state of the stats.



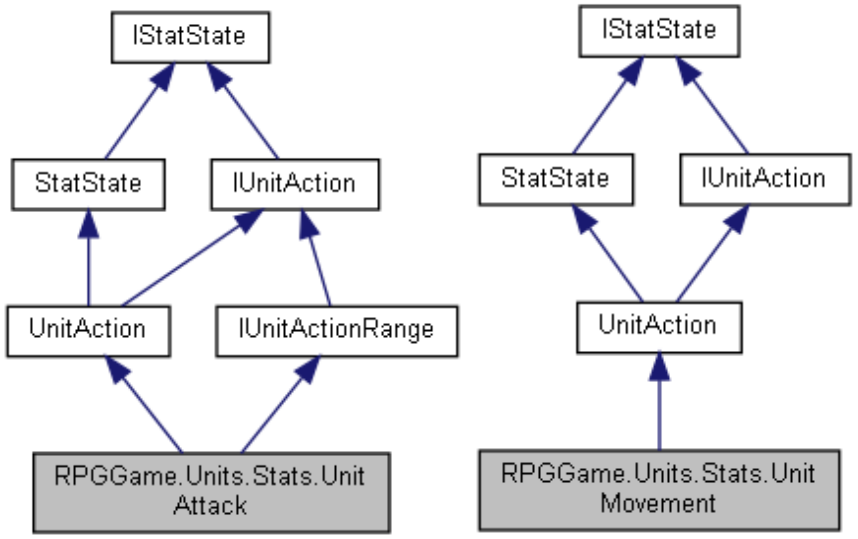
2.4.1 UnitStatsData

The UnitStatsData is obtained from the list of UnitPositions of each Player in the MapLevelData:



2.4.2 UnitActions

The health and the ActionPoints of each unit are plain StatStates, the attack and movement of the units on the other hand are UnitActions since they have a value and a cost to perform them.



2.4.2 Final words

In a nutshell, the game dependencies are designed this way:

- The **UnitState** has a **UnitCosmetic** and **StatsState.**
- The **PlayerRPG** has **UnitStates** and can have an **AIController.**
- The **Board** has n **Cells.**
- The **GameRPGController** has a **Board** and two **PlayerRPG.**
- The **GameSessionManager** has a **GameRPGController.**
- The **PlayerTurnStates** interact with the **GameSessionManager.**

Everything is loaded using the **MapLevelData** that contains the UnitPositions and Board size. All the Datas are ScriptableObjects or serialize classes inside them.