Application Design

The *Chevalier* application has been organized into four distinct groups of objects, Game Objects, Presentation Objects, Game State Objets, and Rules Objects. For a System Diagram expressed using *Unified Modeling Language*[4] of objects in *Chevalier* and how they interact within the application see Figure 1.0.

The Game Objects, labeled in gray on the System Diagram, are controller objects, the most important of which is the Chevalier Object, which may be thought of as the main controller that is the first created object and from which all other objects for the application are generated. In particular, two Player Objects are created, one for each player. Each Player Object has four array lists with references to the created Elements pertaining to that player's army. There is a `left` list, containing all the Elements in the player's left command; a `right` list, for the right command; a `center` list, for the center command; and a `dead` list, where all Element references from any command are moved to once they are designated as removed from the game. Every Element Object has references to 9 Footprint Objects. The first eight Footprints are pre-generated templates representing the Element in every facing, one for each spoke of the compass, these eight pre-generated templates facilitate faster game response. The ninth Footprint, represents the current element position. There are also two sets of ten MoveType Objects that are used to describe types of moves Element Objects may make.

[4] For information on *Unified Modeling Language* see www.uml.org.

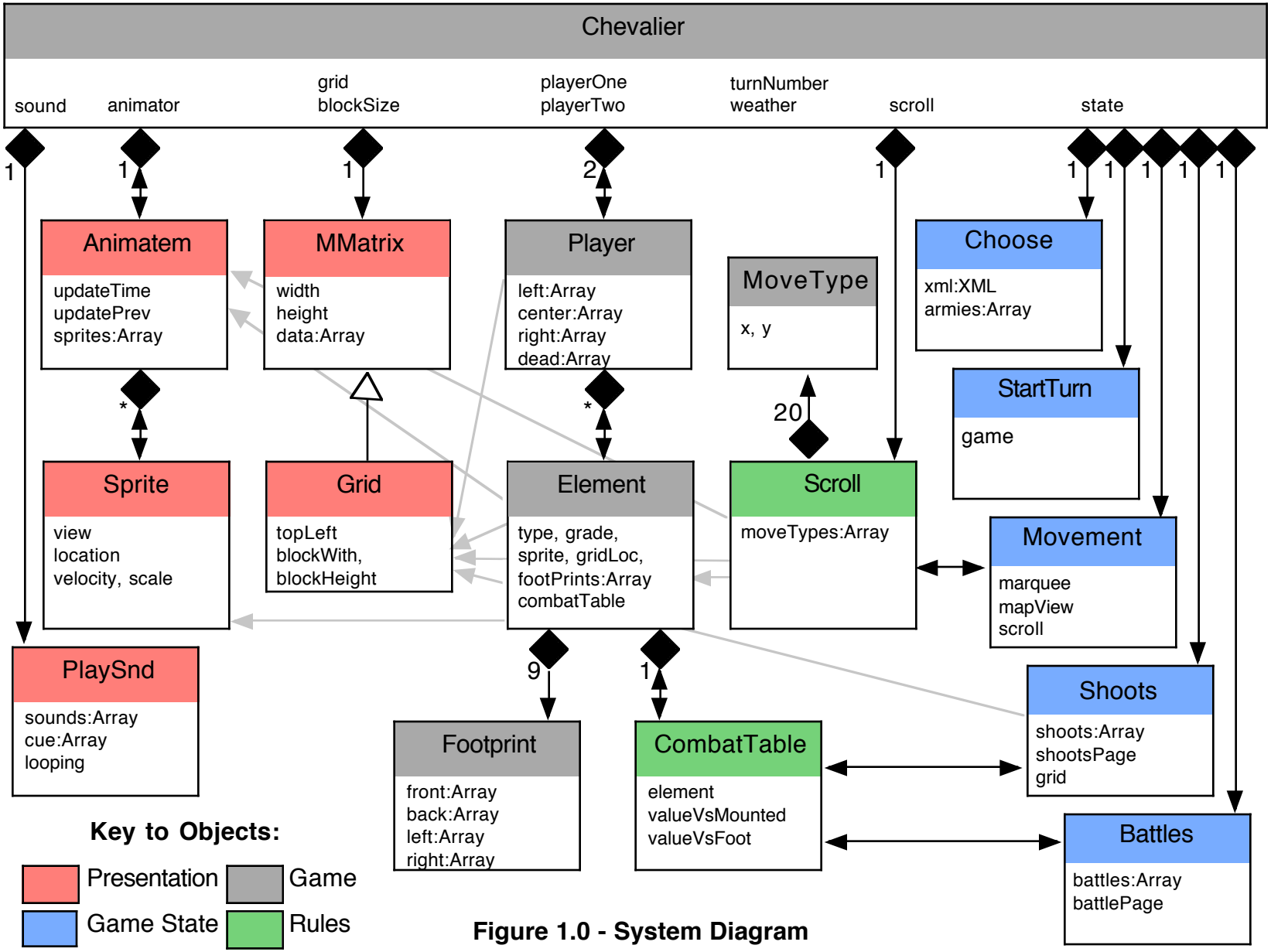

Figure 1.0 - System Diagram

The Game State Objects, labeled in blue on the System Diagram, are those objects that represent a specific state the *Chevalier* game is in. All Game State objects, implement the interface `IGameState`, which ensures that each State Object is prepared to accept various update calls, start state and end state calls, and standardized input from the keyboard and mouse.

There are five game states; "Choose," for choosing an army or battle scenario at the beginning of the game; "StartTurn," for displaying and initializing a new turn; "Movement," for moving elements around on the map; "Shoots," for conducting the resolution of distant shooting; and "Battles," for conducting close combat. The last four states each represent a game phase and are called cyclically each turn for each player.

The Rules Objects, labeled in green on the System Diagram, have been intentionally encapsulated away from the Game Objects to allow for future interchangeability of game systems created by Wargames Research Group, namely *DBA*, *DBM*, *DBR* and *DBMM*. It is theoretically possible to cater for each *DB* rule set by creating each its own customized instance of Rules Objects. At the beginning of *Chevalier* the player could potentially choose a rule set by which to play the game and the appropriate Rules Objects be loaded. There are two Rules Objects, the CombatTable Object, which contains all the rules and tables for conducting combat, and the Scroll Object, which is partly a controller object, but has all the specifics of game movement.

The Presentation Objects, labeled in red on the System Diagram, are those that handle screen display and screen management. They are all generic objects that are encapsulated away from the main body of code and are in no way specific to *Chevalier*. The most important Presentation Objects are the Animatem and Sprite Objects, which together comprise the *Animatem* engine. This engine is essentially a velocity engine that handles the timed animation of multiple Sprites. When Sprites collide or reach a destination they send message back to the controlling *Chevalier* Object which deals with the situation appropriately. Every Element Object is assigned a Sprite that is used to display the state and position of the corresponding Element. There is also a Sprite assigned to the game map, allowing it to be moved, scaled, and rotated easily via the *Animatem* engine. A generic PlaySound Presentation Object is used specifically to handle game audio, and there are MMatrix and Grid Objects that deal with map locations, terrain, and locations of Elements on the gaming map.