**Game Architecture Overview**

The game is segmented into four main groups

* The entities in the game
* The components used by the entities
* The entity managers
* The state machines

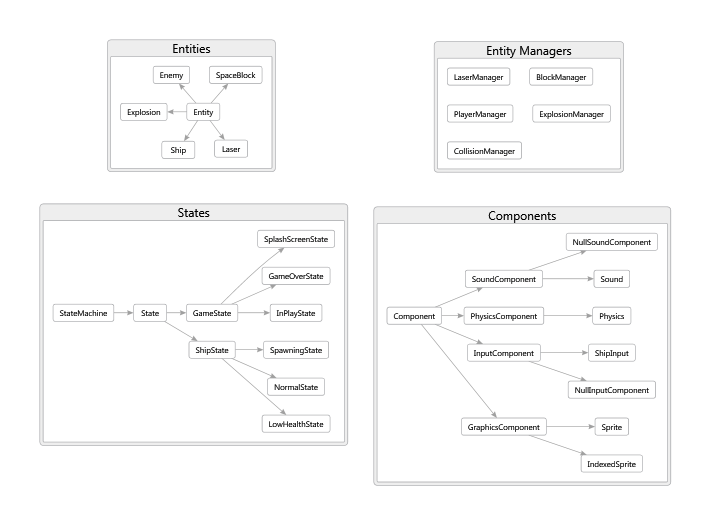


Figure 1 The groups and the classes they contain

**The Entities**

The entity classes represent the moving visible objects on the screen. The entity classes are Ship, SpaceBlock, Laser, Enemy and Explosion. These classes are subclasses of the Entity class.

Information common to all entities is stored in the Entity class. This is information such as:

* Velocity
* Position
* Height
* Width

The Entity class is extended / subclassed by every entity. The Entity class constructor takes several abstract components for drawing, playing sound, handling user input, etc. The Entity base class calls update on all of these components when its Update method is called.l It calls draw on its graphics component when its Draw method is called.

The specific components to be used by the base class are created by the entity subclass.

An entity subclass fills in the components used by the Entity base class. An entity subclass builds itself by creating components and passing them to the base class constructor.

One way to look at it, is that Entity is a shell of a car, just the frame. The actual entity creates a new frame and populates it with features (doors, windows, etc).

**The Components**

The abstract component types are:

* GraphicsComponent (draws the entity)
* InputComponent (handles input)
* PhysicsComponent (updates the position of the entity)
* SoundComponent (plays sounds (if any) for the entity)

In otherwords, the entities are composed of data and the components which operate on that data.

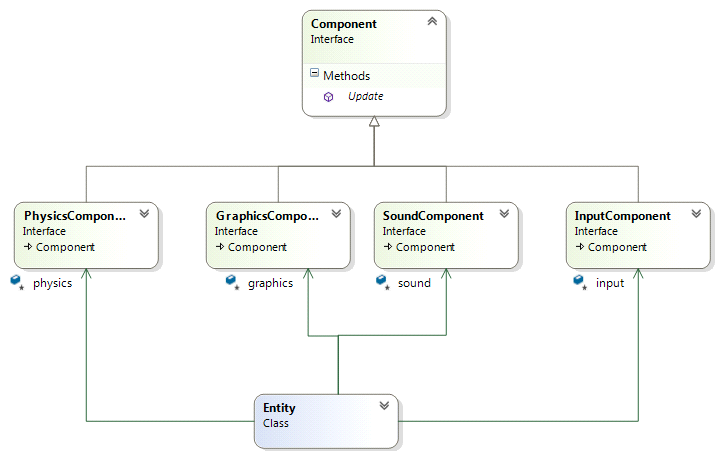


Figure 2 An Entity is Composed of Components

This approach is called the Component pattern and it has become popular in recent years for game programming.

See <http://gameprogrammingpatterns.com/component.html> for more information about this pattern.

**The Entity Managers**

To avoid having lots of code in the main Game class dealing with explosions and lasers and ships, each entity has an associated class called a manager. Instead of looping through each of these entities in the update and draw methods, the game calls Update and Draw on each manager.

The manager class keeps track of instances of its associated entity.

ExplosionManager for example has methods for adding and removing explosions from the screen. The block manager can spawn a number of blocks to the screen.

These managers are used to decouple the main Update method from all of the individual entities.

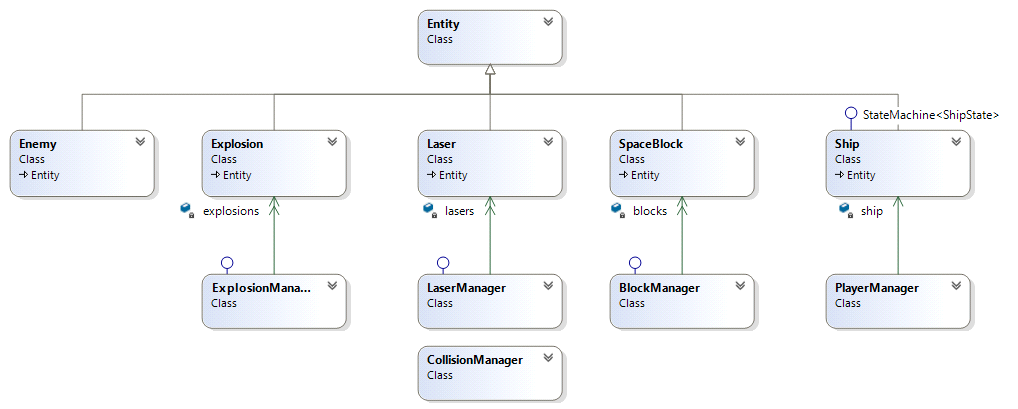


Figure 3 The Managers and the Entities they Manage

The managers are an example of the "update method" pattern: <http://gameprogrammingpatterns.com/update-method.html>

**The State Machines**

The state machines are currently Game and Ship. Game uses states to transition between the splash screen, the main game, and the game over screen.

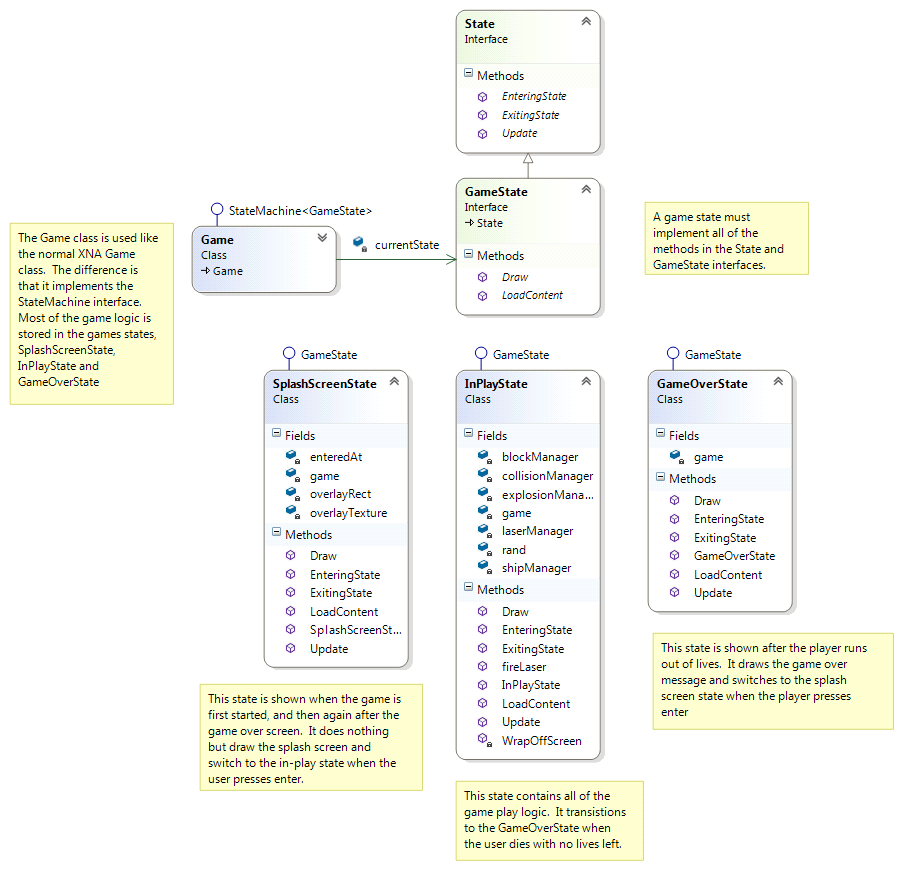


Figure 4 Game States

The ship uses states to transition between spawning, having its normal appearance, and being near death (blinking red).

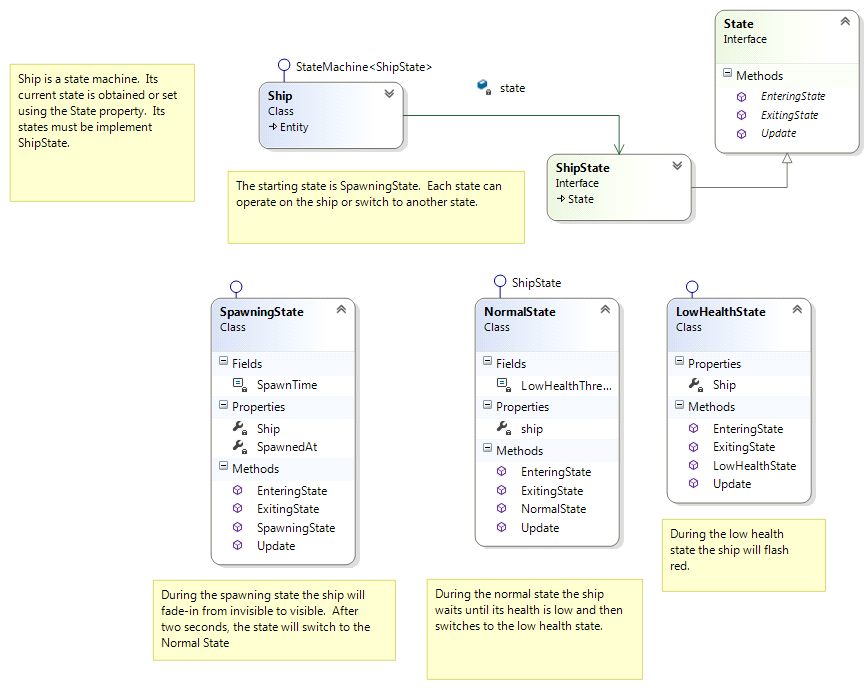


Figure 5 Ship States

This is an example of the State pattern.

<http://gameprogrammingpatterns.com/state.html#the-state-pattern>

**The StateMachine and State Interfaces**

StateMachine and State are interfaces which are only used when creating a new state machine or adding a new state to an existing state machine.

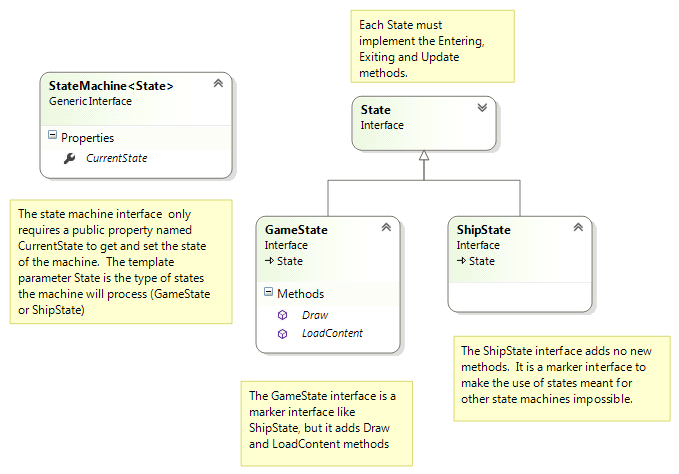


Figure 6 State / StateMachine Interfaces

**Design Philosophy**

* Many small files are better than several large files
* Having many specific interfaces is better than having a couple of very general interfaces.
* Composition is better than inheritance.
* Use patterns where appropriate (this project uses the component, state and update method patterns)