**Readme:**

**Demo:**

To launch the particle system demo, simply open “index.html” in any javascript enabled web browser.

The demo uses several different implementations of the same basic particle system to create a variety of background effects:

* Rain – Many large particles fall at an 85 degree angle to imply the presence of an ambient, gentle breeze. The system also utilizes an “oscillating wind” force object to slightly stagger the direction in which the particles fall.
* Snow – “Snow” is remarkably similar to “Rain”, but incorporates a much stronger and more lopsided “oscillating wind” force. This combined with a reduced downward velocity makes the snowflakes appear to ‘float’ through powerful gusts of wind.
* Bug Swarm – This effect is created by applying a force in the, slightly offset, direction of some user-specified point to each particle in the system. This causes the ‘bug’ to chaotically orbit said point in manner reminiscent of clouds of mosquitoes, gnats, and flies.
* Star Field – This effect users two different particle controllers to create a scrolling, parallax starfield. One controller emits larger particles that fall faster and the other emits smaller particles that fall slower.
* Explosion – Many particles are created at a user specified point and immediately launch upwards in a parabolic arch.

**Technical Details:**

The particle system consists of five major components:

* Controllers – Particle controller objects are used to emit particles exhibiting certain properties. Each controller is assigned a ‘Particle Properties’ object and zero or more force applicators. A controller must be provided a properties object before it can be instantiated, but force applicators can be added on the fly
* Properties – Properties are used to configure particle controllers. This reduces the number parameters needed by a controller and allows the user to instantiate several particle controllers with the same properties object.
* Forces – Forces are used to manipulate particles by applying ‘force’ to them. The exact meaning of this is implementation dependent and a force object is free to manipulate its controller’s active particles in any way the user sees fit. However, particles have specific fields and methods to standardize the force application process (the specifics of this are outlined in the “Contracts” section).
* Graphics Interfaces – Graphics interfaces are used to abstract the drawing functions used by the particle system and further decouple the individual components. Users can then interface custom or third-party graphics libraries by extending the default class and overriding its methods (the demo’s graphics controller, for example, does this for HTML canvas functions). Again, specific requirements are detailed in the “Contracts” section.
* Sprites – Particle controllers use sprites to abstract graphics code from the individual particles. This allows the user to add more complex visual effects to the particle system without having to modify the particle class.

**Scripts:**

* “.\js\forceapplicators\GenericApplicator.js” – Defines the generic force applicator
* “.\js\forceapplicators\BugApplicator.js” – Defines the “Bug Swarm” force applicator. This force applies a directional “wind” pointing towards some predefined point. It can be configured to add random ‘jitter’ to the vector’s direction creating a random wobble effect as the particle moves
* “.\js\forceapplicators\OscillatingWind.js” – Defines the “Oscillating Wind” force applicator. This force creates a global wind that oscillates between some minimum and maximum at a predefined rate.
* “.\js\graphics\GenericGraphic.js” – Defines the generic sprite class
* “.\js\graphics\GenericCircle.js” – Defines a sprite that draws a primitive circle
* “.\js\graphics\GenericImage.js” – Defines a sprite that draws an image
* “.\js\graphicscontrollers\GraphicsController” – Defines the generic graphics interface
* “.\js\graphicscontrollers\HTMLCanvasController” – Defines the HTML Canvas graphics interface used in the demo
* “.\js\main\DemoMain.js” – implements the core functionality of the demo and implementations of the demoed particle systems.
* “.\js\particlecontrollers\ParticleController.js” – Defines the generic particle controller
* “.\js\particlecontrollers\DemoParticleController.js” – Defines the particle controllers used in the demo (these exhibit the same functionality of the default particle controllers, but remove particles that exit the drawing area)
* “.\js\particleproperties\ParticleProperties.js” – Defines the generic particle properties class
* “.\js\particles\Particle.js” – Defines the generic particle
* “.\ParticleSystem.js” – Defines the generic particle system

**Implementation Details:**

To interface the particle system with your custom projects you first need to port the graphics code you are using by creating a custom graphics controller that extends the generic controller (see the HTML Canvas controller for an example of this).

Particle systems contain a list of particle controllers, and each controller contains a list of force applicators and a properties object. Properties define the spawning areas of particles, their default velocities, and graphics. Particle systems are updated using the step function and rendered using the draw function. This allows you to update and render the system at different frequencies depending on the setup of your project.

**Contracts:**

* Graphics Contract – An implementation of the graphics controller must implement each of the prototyped functions. Additionally, any particle system class can assume that each of the default methods are defined.
* Force Contract – A force applicator must be given the entire list of its controller’s active particles each step event. Likewise, any force applicator can assume that it has access to a list of active particles in its step function. Although force applicators are permitted to manipulate a particle in any way the user wishes, they should only use the “applyForce” method defined in the generic particle class.