Easing Functions from TweenJS library

I showed you in class many *normalized easing functions* used in a similar fashion we used `Mathf.PingPong` and `Mathf.Sin` functions, to implement quickly many of the Scott Rogers' mechanics. They came from this reference:

Ref: TweenJS Demonstration Page

You can see all of them in the appendix below. I am collecting them in a table for your convenience below. Before that, a little discussion on the general form of these functions, and the *normalized* form used in this library (and many others like this in other languages, like Tween.cs etc):

- The general form of these functions is $f = f(t)$, whereby *normalized* means that both $t \in [0, 1]$ and $f(t) \in [0, 1]$ (at least the starting and ending points and many points — see the **backXYZ** and **elasticXYZ** examples whith the points outside of this box in the list and the demonstartions).
- They either start at point $(0, 0)$ and end at point $(1, 1)$, or,
- They start at point $(0, 1)$ and end at point $(1, 0)$.
- The **in** refers to the starting point (on the left)
- The **out** refers to the ending point (on the right)
- The **ease** refers to the movement being slower that usual in the respective end(s). (In the graphs, the *dots* are spready evenly in time, so the part of slow movement shows the dots more frequent. The **in**-s have more frequent dots in the start, **out**-s in the end, **InOut**-s in both.
- The **InOut** refers to both starting and ending points of movements.
- Notice that the **InOut** form always is built by *piecing* together the **In** form with the **Out** form in the point $(1/2, 1/2)$.
- The transition is always smooth (without a kink in $(1/2, 1/2)$).
- This can give an idea how to build an arbitrary easing function:
  - Start with the **In** form. Any function from $(0, 0)$ to $(1, 1)$ will do, say $f_i(t)$.
  - Next, build the **Out** form. Again any function can do, *provided* that the slope at $(0, 0)$ is equal with the slope at $(1, 1)$ of the **In** form, say $f_o(t)$.
  - Then, the function $f_{io}(t) = (t \leq 1/2) \cdot \frac{f_i(2 \cdot t)}{2} + (t > 1/2) \cdot (1 - 0.5 \cdot f_o(2 - t))$ is of **InOut** form.
- Other considerations: see the Reference above.

Method Name	$f_{In}(t)$	$f_{Out}(t)$	$f_{InOut}(t)$
Linear	t	t	t
Power_k	t^k	$1-(1-t)^k$	$(t <= 0.5) \cdot (0.5 \cdot (2t)^k) + (t > 0.5) \cdot (1 - 0.5 \cdot ($
Quad=Power_2			
Cubic=Power_3			
Quart=Power_4			
Quint=Power_5			
Sine			
getBack_k	$t^2 \cdot ((k+1)t - k)$		

```
// static methods and properties
  /**
   * @method linear
   * @param {Number} t
   * @static
   * @return {Number}
   **/
  Ease.linear = function(t) { return t; };

  /**
   * Identical to linear.
   * @method none
   * @param {Number} t
   * @static
   * @return {Number}
   **/
  Ease.none = Ease.linear;

  /**
   * Mimics the simple -100 to 100 easing in Adobe Flash/Animate.
   * @method get
   * @param {Number} amount A value from -1 (ease in) to 1 (ease out) indicating the
strength and direction of the ease.
   * @static
   * @return {Function}
   **/
  Ease.get = function(amount) {
    if (amount < -1) { amount = -1; }
    else if (amount > 1) { amount = 1; }
    return function(t) {
      if (amount==0) { return t; }
      if (amount<0) { return t*(t*-amount+1+amount); }
      return t*((2-t)*amount+(1-amount));
    };
  };
```

```
/**
 * Configurable exponential ease.
 * @method getPowIn
 * @param {Number} pow The exponent to use (ex. 3 would return a cubic ease).
 * @static
 * @return {Function}
 **/
Ease.getPowIn = function(pow) {
  return function(t) {
    return Math.pow(t,pow);
  };
};

/**
 * Configurable exponential ease.
 * @method getPowOut
 * @param {Number} pow The exponent to use (ex. 3 would return a cubic ease).
 * @static
 * @return {Function}
 **/
Ease.getPowOut = function(pow) {
  return function(t) {
    return 1-Math.pow(1-t,pow);
  };
};

/**
 * Configurable exponential ease.
 * @method getPowInOut
 * @param {Number} pow The exponent to use (ex. 3 would return a cubic ease).
 * @static
 * @return {Function}
 **/
Ease.getPowInOut = function(pow) {
  return function(t) {
    if ((t*=2)<1) return 0.5*Math.pow(t,pow);
    return 1-0.5*Math.abs(Math.pow(2-t,pow));
  };
};

/**
 * @method quadIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quadIn = Ease.getPowIn(2);
/**
 * @method quadOut
 * @param {Number} t
 * @static
 * @return {Number}
```

```
 **/
Ease.quadOut = Ease.getPowOut(2);
/**
 * @method quadInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quadInOut = Ease.getPowInOut(2);

/**
 * @method cubicIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.cubicIn = Ease.getPowIn(3);
/**
 * @method cubicOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.cubicOut = Ease.getPowOut(3);
/**
 * @method cubicInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.cubicInOut = Ease.getPowInOut(3);

/**
 * @method quartIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quartIn = Ease.getPowIn(4);
/**
 * @method quartOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quartOut = Ease.getPowOut(4);
/**
 * @method quartInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quartInOut = Ease.getPowInOut(4);
```

```
/**
 * @method quintIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quintIn = Ease.getPowIn(5);
/**
 * @method quintOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quintOut = Ease.getPowOut(5);
/**
 * @method quintInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.quintInOut = Ease.getPowInOut(5);

/**
 * @method sineIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.sineIn = function(t) {
  return 1-Math.cos(t*Math.PI/2);
};

/**
 * @method sineOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.sineOut = function(t) {
  return Math.sin(t*Math.PI/2);
};

/**
 * @method sineInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.sineInOut = function(t) {
  return -0.5*(Math.cos(Math.PI*t) - 1);
};
```

```
/**
 * Configurable "back in" ease.
 * @method getBackIn
 * @param {Number} amount The strength of the ease.
 * @static
 * @return {Function}
 **/
Ease.getBackIn = function(amount) {
  return function(t) {
    return t*t*((amount+1)*t-amount);
  };
};
/**
 * @method backIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.backIn = Ease.getBackIn(1.7);

/**
 * Configurable "back out" ease.
 * @method getBackOut
 * @param {Number} amount The strength of the ease.
 * @static
 * @return {Function}
 **/
Ease.getBackOut = function(amount) {
  return function(t) {
    return (--t*t*((amount+1)*t + amount) + 1);
  };
};
/**
 * @method backOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.backOut = Ease.getBackOut(1.7);

/**
 * Configurable "back in out" ease.
 * @method getBackInOut
 * @param {Number} amount The strength of the ease.
 * @static
 * @return {Function}
 **/
Ease.getBackInOut = function(amount) {
  amount*=1.525;
  return function(t) {
    if ((t*=2)<1) return 0.5*(t*t*((amount+1)*t-amount));
    return 0.5*((t-=2)*t*((amount+1)*t+amount)+2);
  };
```

```
};
/**
 * @method backInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.backInOut = Ease.getBackInOut(1.7);

/**
 * @method circIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.circIn = function(t) {
  return -(Math.sqrt(1-t*t)- 1);
};

/**
 * @method circOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.circOut = function(t) {
  return Math.sqrt(1-(--t)*t);
};

/**
 * @method circInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.circInOut = function(t) {
  if ((t*=2) < 1) return -0.5*(Math.sqrt(1-t*t)-1);
  return 0.5*(Math.sqrt(1-(t-=2)*t)+1);
};

/**
 * @method bounceIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.bounceIn = function(t) {
  return 1-Ease.bounceOut(1-t);
};

/**
 * @method bounceOut
 * @param {Number} t
```

```
 * @static
 * @return {Number}
 **/
Ease.bounceOut = function(t) {
  if (t < 1/2.75) {
    return (7.5625*t*t);
  } else if (t < 2/2.75) {
    return (7.5625*(t-=1.5/2.75)*t+0.75);
  } else if (t < 2.5/2.75) {
    return (7.5625*(t-=2.25/2.75)*t+0.9375);
  } else {
    return (7.5625*(t-=2.625/2.75)*t +0.984375);
  }
};

/**
 * @method bounceInOut
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.bounceInOut = function(t) {
  if (t<0.5) return Ease.bounceIn (t*2) * .5;
  return Ease.bounceOut(t*2-1)*0.5+0.5;
};

/**
 * Configurable elastic ease.
 * @method getElasticIn
 * @param {Number} amplitude
 * @param {Number} period
 * @static
 * @return {Function}
 **/
Ease.getElasticIn = function(amplitude,period) {
  var pi2 = Math.PI*2;
  return function(t) {
    if (t==0 || t==1) return t;
    var s = period/pi2*Math.asin(1/amplitude);
    return -(amplitude*Math.pow(2,10*(t-=1))*Math.sin((t-s)*pi2/period));
  };
};
/**
 * @method elasticIn
 * @param {Number} t
 * @static
 * @return {Number}
 **/
Ease.elasticIn = Ease.getElasticIn(1,0.3);

/**
 * Configurable elastic ease.
 * @method getElasticOut
```

```
   * @param {Number} amplitude
   * @param {Number} period
   * @static
   * @return {Function}
   **/
  Ease.getElasticOut = function(amplitude,period) {
    var pi2 = Math.PI*2;
    return function(t) {
      if (t==0 || t==1) return t;
      var s = period/pi2 * Math.asin(1/amplitude);
      return (amplitude*Math.pow(2,-10*t)*Math.sin((t-s)*pi2/period )+1);
    };
  };
  /**
   * @method elasticOut
   * @param {Number} t
   * @static
   * @return {Number}
   **/
  Ease.elasticOut = Ease.getElasticOut(1,0.3);

  /**
   * Configurable elastic ease.
   * @method getElasticInOut
   * @param {Number} amplitude
   * @param {Number} period
   * @static
   * @return {Function}
   **/
  Ease.getElasticInOut = function(amplitude,period) {
    var pi2 = Math.PI*2;
    return function(t) {
      var s = period/pi2 * Math.asin(1/amplitude);
      if ((t*=2)<1) return -0.5*(amplitude*Math.pow(2,10*(t-=1))*Math.sin( (t-
s)*pi2/period ));
      return amplitude*Math.pow(2,-10*(t-=1))*Math.sin((t-s)*pi2/period)*0.5+1;
    };
  };
  /**
   * @method elasticInOut
   * @param {Number} t
   * @static
   * @return {Number}
   **/
  Ease.elasticInOut = Ease.getElasticInOut(1,0.3*1.5);
```