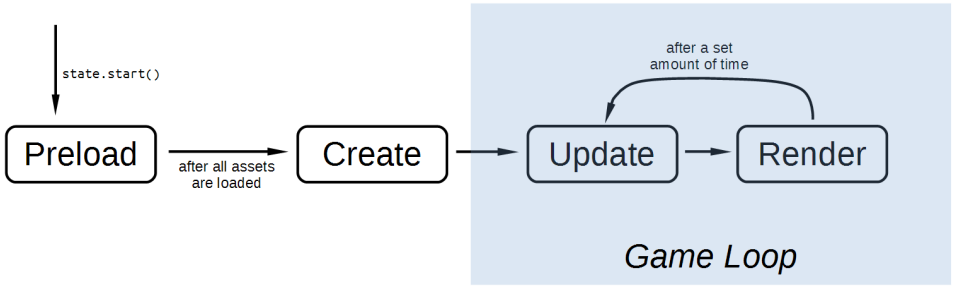
Week 5: How Phaser.js Works

How a Game Engine Works

Last week we saw some of the basic control structures in JavaScript, those instructions that provide for:

1. **if ... else:** the if statement gives us the opportunity to introduce decision points and conditional logic into our code
2. **for:** the for statement gives us the chance to repeat blocks of code more than once
3. **events like onclick**: events provide ways to trigger the execution of blocks of our code

As we begin using the thousands of lines of JavaScript code written inside of the Phaser.js game engine, let's step back and take a look at the way the engine is set up for powering gameplay.



States

Phaser operates on the principle of states. The game and other objects each have states of being that can be performed, completed, and restarted. The basic flow of states from one to another is modeled in the diagram above.

Phaser runs through a series of methods – JavaScript functions associated with objects – in order to process one single update, including a pre-update, update, and post-update model:

- this.debug.preUpdate();
- this.world.camera.preUpdate();
- this.physics.preUpdate();
- this.state.preUpdate(timeStep);
- this.plugins.preUpdate(timeStep);
- this.stage.preUpdate();
- this.state.update();
- this.stage.update();
- this.tweens.update(timeStep);
- this.sound.update();

- this.input.update();
- this.physics.update();
- this.particles.update();
- this.plugins.update();
- this.stage.postUpdate();
- this.plugins.postUpdate();

Finite-State Machine (FSM)

Phaser is written to enforce the rule that only one single state can be running at any time for any particular object. States can be started up, run, and concluded, with each stage in that sequence expressed as a block of code and available as a Phaser object method call. We call this a Finite-State Machine, or FSM for short, because only one state is ever running at a time, so there is no way for an infinite number of states to be propagated.

Display List

Phaser manages a list that contains all of the elements that may be visible within your game – player characters, score text, etc. The sequence of the list determines how the elements are visibly composed. Graphics at the start of the list are drawn first so that anything drawn afterwards may appear visibly over – or on top of – the previously-drawn graphics.

World

The game world in Phaser is where everything takes place. The world can be of larger dimensions than the actively visible stage area. When a world is bigger than the game's stage, only a portion of the world will be visible. The dimensions of the stage and the world in Phaser are both expressed in pixel units.

Camera

The camera in Phaser is conceptually based on the camera used in 3D-modeling. However, in our two-dimensional Phaser world, camera movement is simplified since we do not have to worry about a Z-axis. Phaser's camera has a basic horizontal and vertical coordinate-based position. It is possible to instruct Phaser to lock the camera to another object's movement – such as a player's character.

Asset Cache

Because Phaser-based games are presumed to running over the Internet in a Web browser, all game asset files must be preloaded to avoid delays and glitches in gameplay timing. The asset cache is where Phaser keeps images, sounds, and other memory-consuming assets. We always preload any asset into

the cache before using it. Phaser automatically tracks the load progress for assets loaded, making it possible to give players feedback about the loading status of a game's assets.

Image Assets

Bitmap graphic images are the primary asset type utilized in a Phaser game. Generally, the PNG file format is preferred for image files, since PNG supports alpha channel transparencies, allowing pixels in the image to "show through" the graphics that appear behind the image. Images must be preloaded into the asset cache before they can be assigned for display within the game.

Sprites

Most of the visible elements in a Phaser game are called sprites. Sprites represent the characters of any game, can have physics applied to them, and allow for animated or still images to be used. We saw sprites in our first class when we looked at the Scratch environment to model a simple pong game. See this article for more on Sprites: **https://en.wikipedia.org/wiki/Sprite_(computer_graphics) (https://en.wikipedia.org/wiki/Sprite_(computer_graphics))**