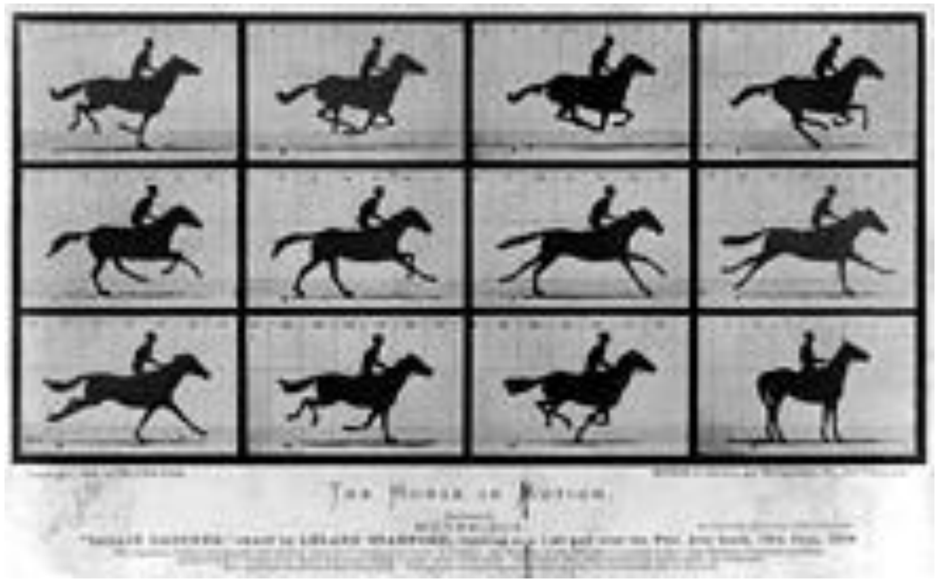
HTML 5 and JavaScript Game Programming

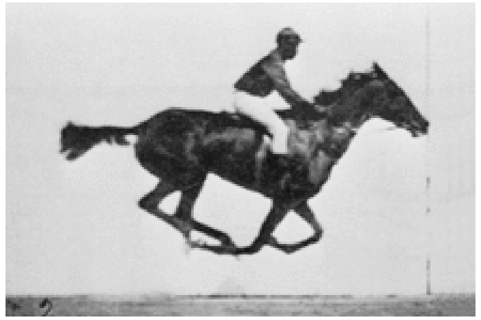
Week 7

Mario.Soflano@uws.ac.uk

Animation

Eadward Muybridge (1872)





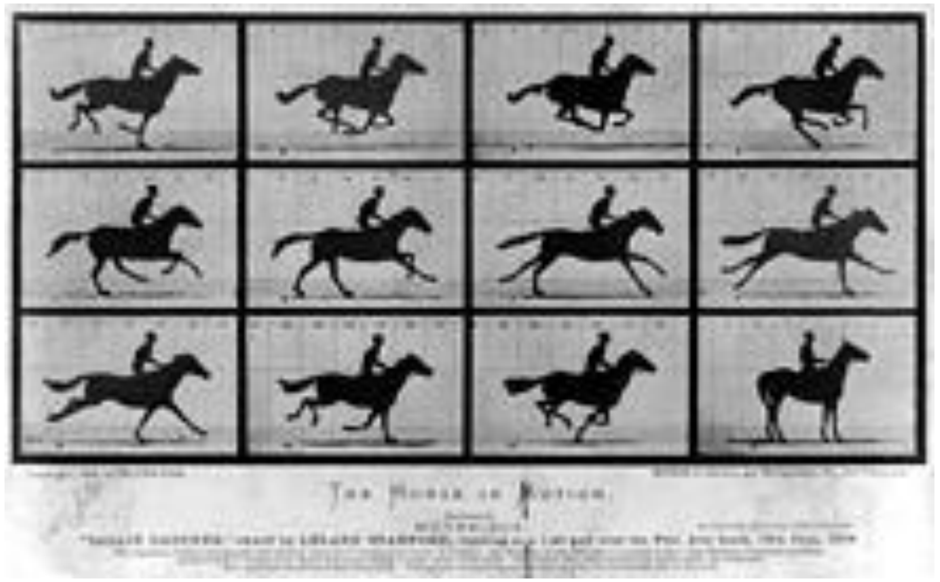
https://en.wikipedia.org/wiki/Eadweard_Muybridge#Stanford_and_horse_gaits

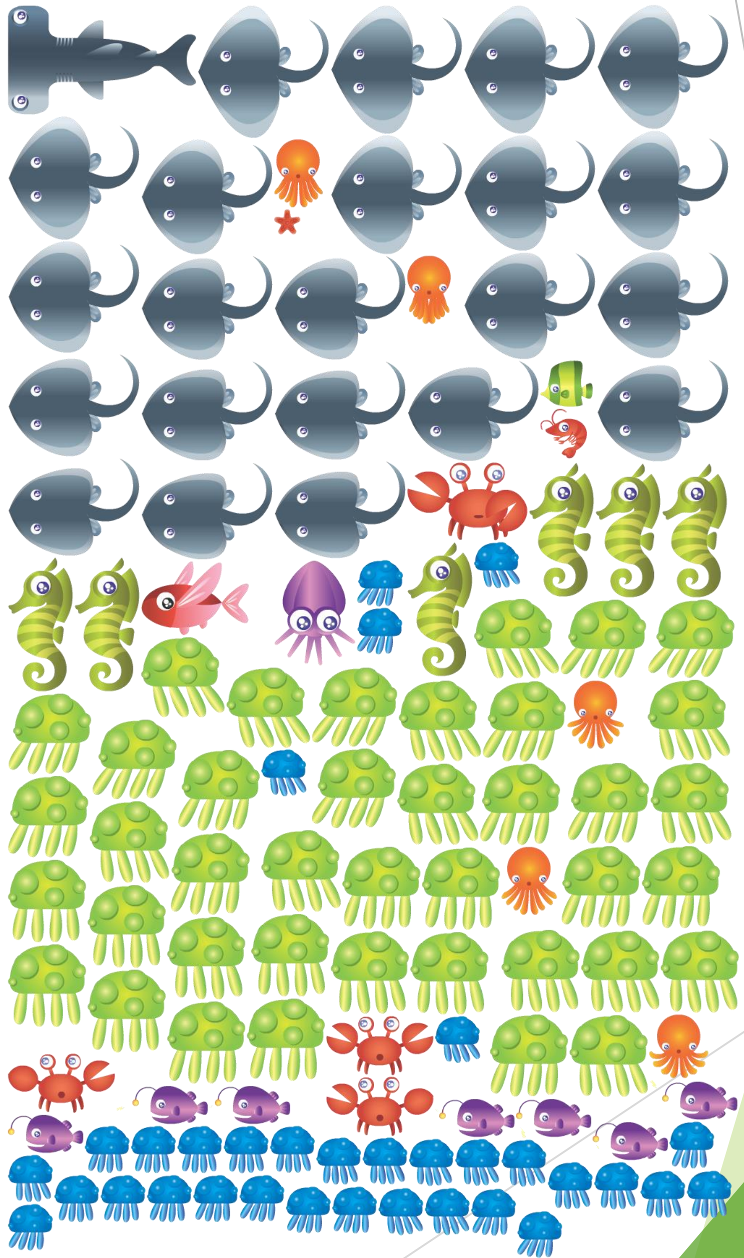
Sprite

- A single graphic image that is incorporated into a larger scene so that it appears to be part of the scene.
- Inefficient if a scene requires many sprites

Spritesheet

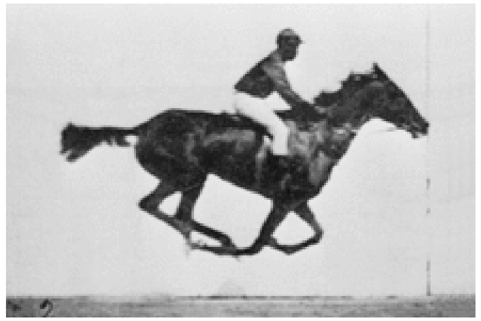
- A single graphic image that contains many sprites which are normally related
- A spritesheet contains:
  - Frames: a single image / sprite
  - Cycles: the sequence / order of the frames to form a continuous movement





Animation

- There are three parts to coding a spritesheet animation:
  - Creating the image
  - Updating the image to each frame of the animation
  - Drawing the frame to the screen

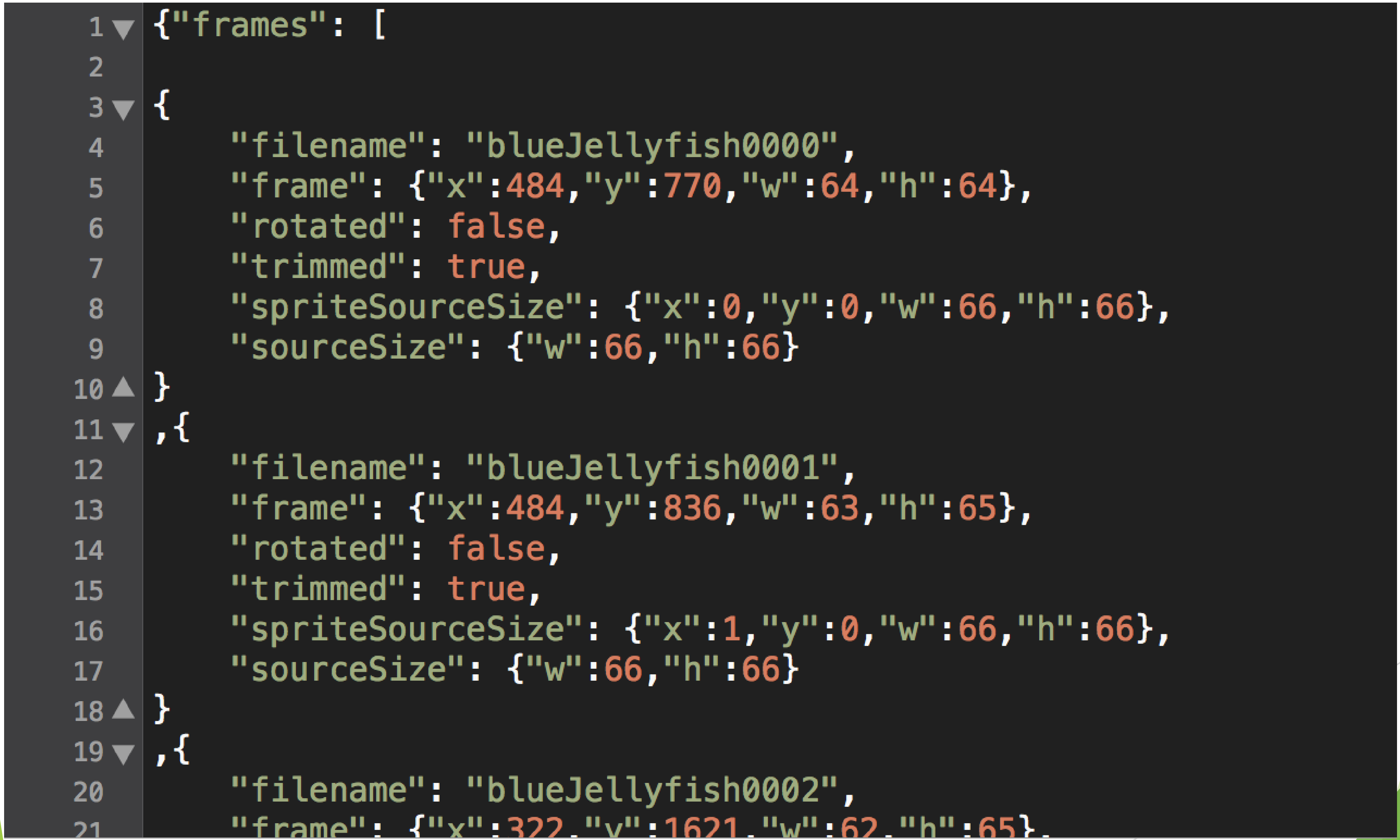


Phaser – Animation Data

Animation data in Phaser:

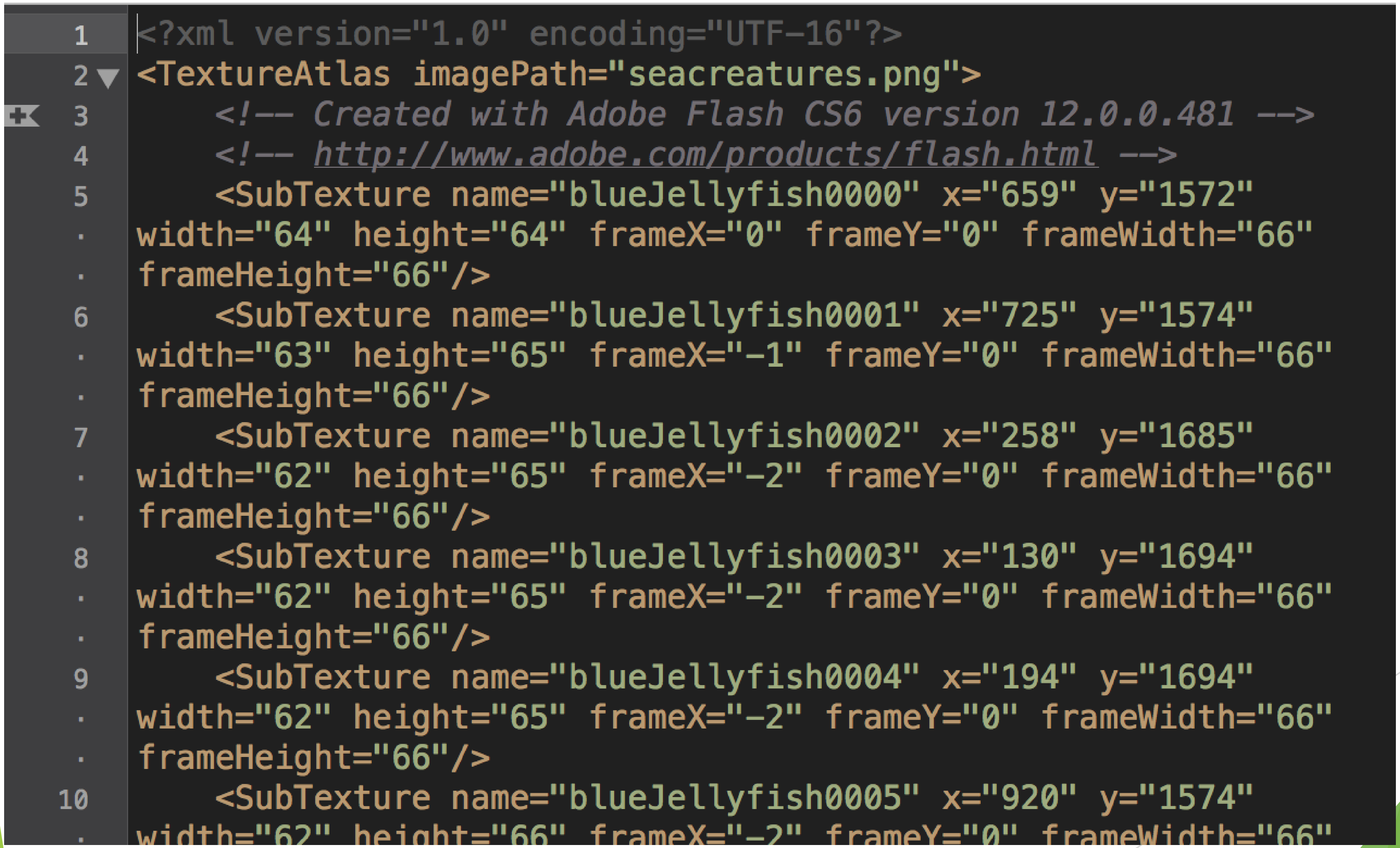
- JSON
- TextureAtlas
- Spritesheet

Phaser – Animation Data (json)



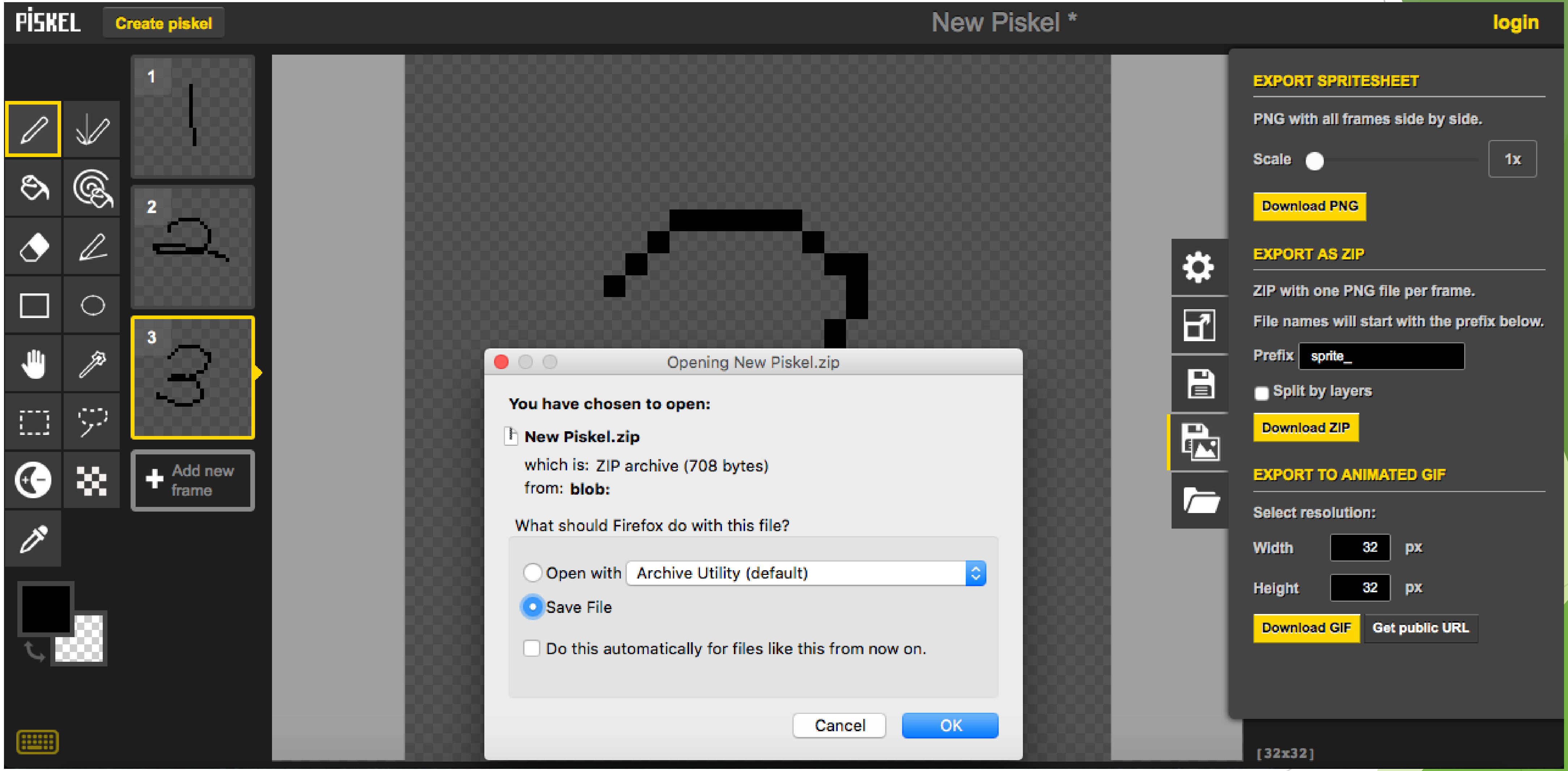
```json
{"frames": [

{
	"filename": "blueJellyfish0000",
	"frame": {"x":484,"y":770,"w":64,"h":64},
	"rotated": false,
	"trimmed": true,
	"spriteSourceSize": {"x":0,"y":0,"w":66,"h":66},
	"sourceSize": {"w":66,"h":66}
}
,{
	"filename": "blueJellyfish0001",
	"frame": {"x":484,"y":836,"w":63,"h":65},
	"rotated": false,
	"trimmed": true,
	"spriteSourceSize": {"x":1,"y":0,"w":66,"h":66},
	"sourceSize": {"w":66,"h":66}
}
,{
	"filename": "blueJellyfish0002",
	"frame": {"x":322,"y":1621,"w":62,"h":65},
```

Phaser – Animation Data (Texture Atlas)



```xml
<?xml version="1.0" encoding="UTF-16"?>
<TextureAtlas imagePath="seacreatures.png">
    <!-- Created with Adobe Flash CS6 version 12.0.0.481 -->
    <!-- http://www.adobe.com/products/flash.html -->
    <SubTexture name="blueJellyfish0000" x="659" y="1572" width="64" height="64" frameX="0" frameY="0" frameWidth="66" frameHeight="66"/>
    <SubTexture name="blueJellyfish0001" x="725" y="1574" width="63" height="65" frameX="-1" frameY="0" frameWidth="66" frameHeight="66"/>
    <SubTexture name="blueJellyfish0002" x="258" y="1685" width="62" height="65" frameX="-2" frameY="0" frameWidth="66" frameHeight="66"/>
    <SubTexture name="blueJellyfish0003" x="130" y="1694" width="62" height="65" frameX="-2" frameY="0" frameWidth="66" frameHeight="66"/>
    <SubTexture name="blueJellyfish0004" x="194" y="1694" width="62" height="65" frameX="-2" frameY="0" frameWidth="66" frameHeight="66"/>
    <SubTexture name="blueJellyfish0005" x="920" y="1574" width="62" height="66" frameX="-2" frameY="0" frameWidth="66"
```

Piskel



EXPORT SPRITESHEET

PNG with all frames side by side.

Scale 1x

Download PNG

EXPORT AS ZIP

ZIP with one PNG file per frame.

File names will start with the prefix below.

Prefix sprite_

Split by layers

Download ZIP

EXPORT TO ANIMATED GIF

Select resolution:

Width 32 px

Height 32 px

Download GIF Get public URL

Opening New Piskel.zip

You have chosen to open:

New Piskel.zip

which is: ZIP archive (708 bytes)

from: blob:

What should Firefox do with this file?

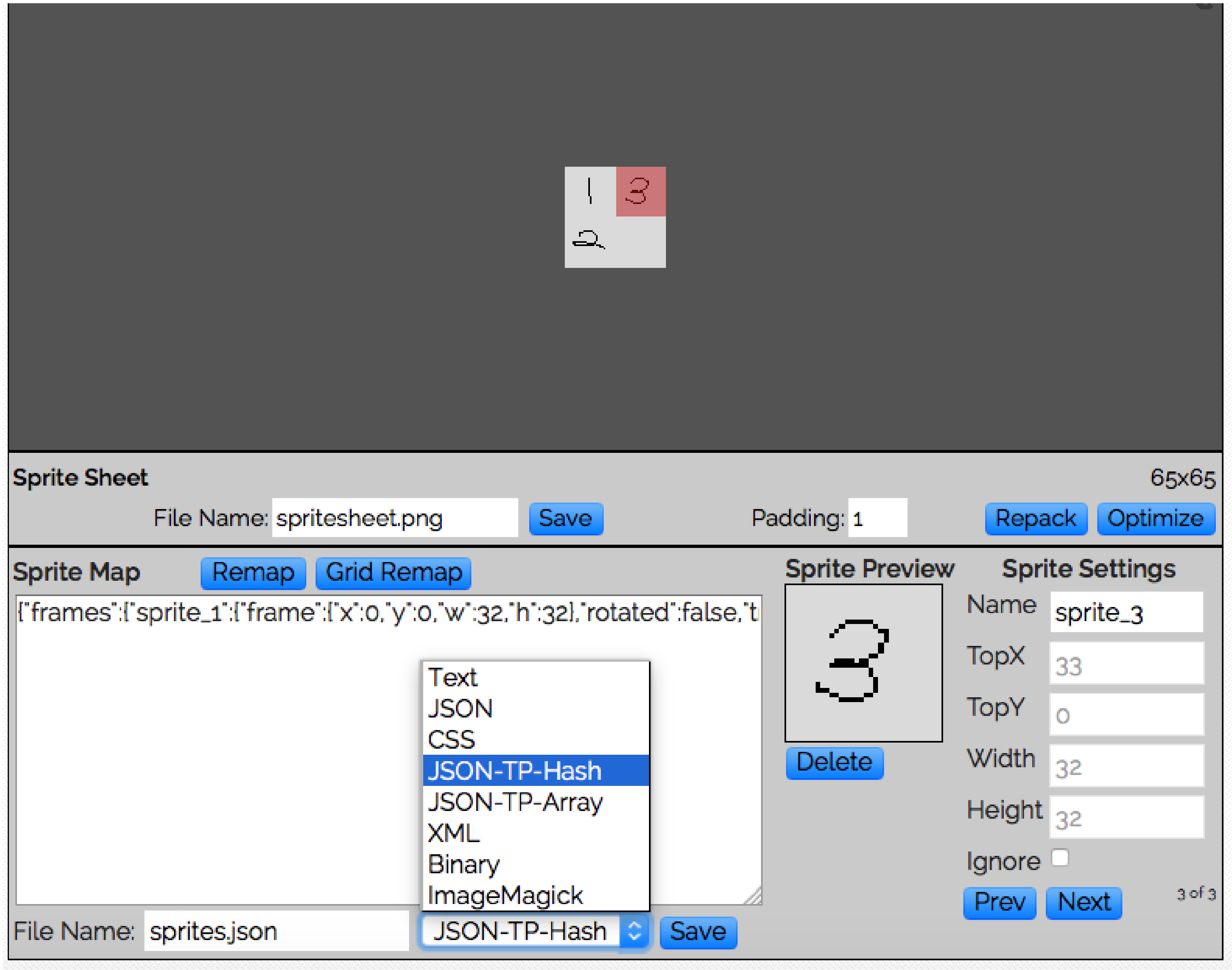
Open with Archive Utility (default) [✓ Save File]

Do this automatically for files like this from now on.

Cancel OK

Create sprites and export it to individual PNG.

Piskel normally packs the PNGs into a ZIP file



Sprite Sheet 65x65

File Name: spritesheet.png Save Padding: 1 Repack Optimize

Sprite Map Remap Grid Remap

{"frames":{"sprite_1":{"frame":{"x":0,"y":0,"w":32,"h":32},"rotated":false,"t

File Name: sprites.json JSON-TP-Hash Save

Sprite Preview

Sprite Settings

Name	sprite_3
TopX	33
TopY	0
Width	32
Height	32
Ignore	☐

http://www.leshylabs.com/apps/sstool/

Click "Save" in the SpriteSheet section to generate the new PNG

Click "Save" in the SpriteSheet section to generate the animation data as specified

Character Animation using Spritesheet

Key: Unique asset key of the sheet file

Url: URL of the sheet file

FrameWidth: Width of each single frame.

FrameHeight: Height of each single frame.

FrameMax (optional): How many frames in this sprite sheet. If not specified it will divide the whole image into frames. Default = -1

Margin (optional): Margin between frame. Default = 0

Spacing (optional): Margin between frame. Default = 0

There are 3 types of texture atlas for Phaser : jsonAtlas, jsonAtlasHash, atlasxml

Key: Unique asset key of the texture atlas file

textureURL The url of the texture atlas image file.

atlasURL (optional) The url of the texture atlas data file (json/xml). You don't need this if you are passing an atlasData object instead.

atlasData (optional) A JSON or XML data object. You don't need this if the data is being loaded from a URL.

format (optional) A value describing the format of the data, the default is Phaser.Loader.TEXTURE_ATLAS_JSON_ARRAY.

Phaser – Animation

Loading Sprite from a Spritesheet

load.spritesheet(key, url, frameWidth, frameHeight, frameMax, margin, spacing)

game.load.spritesheet('mySpriteSheet', 'assets/mySprite90x90x6.png', 90, 90, 6);

Loading Sprite from Atlas JSON / JSONHash

load.atlas(key, textureURL, atlasURL, atlasData, format)

game.load.atlas('seacreatures', 'assets/sprites/seacreatures_json.png', 'assets/sprites/seacreatures_json.json');

Loading Sprite from Atlas XML

load.atlasXML(key, textureURL, atlasURL, atlasData)

game.load.atlasXML('seacreatures', 'assets/sprites/seacreatures.png', 'assets/sprites/seacreatures.xml');

Phaser – Animation

Adding animation to game

add.sprite(x, y, key, frame, group)

X: A number representing the X position of the new sprite.

Y: A number representing the Y position of the new sprite.

Key: A text representing the image or texture used by the Sprite during rendering. Normally it should be the same key as when the asset is loaded

Frame (optional): If the sprite uses an image from a texture atlas or sprite sheet you can pass the frame here. Either a number for a frame ID or a string for a frame name.

Group (optional): Optional Group to add the object to. If not specified it will be added to the World group.

```
add.sprite(200, 360, 'mySpriteSheet');
```

Phaser – Animation

Add(name,frames,frameRate,loop,useNumericIndex)

Name: a string representing the unique name of animation

Frames (optional): An array of numbers/strings that correspond to the frames to add to this animation and in which order. e.g. [1, 2, 3] or ['run0', 'run1', run2]). If null then all frames will be used. Default: null

Frames (optional): A number that represent the framerate. Default: 60

Loop (optional): A boolean that determines whether the animation is to be played continuously or only once. Default: false

UseNumericIndex (optional): Are the given frames using numeric indexes (default) or strings? Default: true

```
animations.add('swim', Phaser.Animation.generateFrameNames('blueJellyfish', 0,
32, '', 4), 30, true);
```

Phaser – Animation

Playing the animation

play(frameRate, loop, killOnComplete)

FrameRate: A number representing the framerate per second for the animation

Loop (optional): A boolean to determine whether the animation be looped after playback. If not provided the previously set loop value of the Animation is used.

KillOnComplete (optional): A boolean which If set to true when the animation completes (only happens if loop=false) the parent Sprite will be killed.

```
play(10, true);
```

Phaser – Animation Events

Some events can be triggered during the animation

```
//call function 'functionWhenAnimStart' when the animation in object 'theObject' starts
    onStart.add(functionWhenAnimStart, theObject);
//call function 'functionWhenAnimLoop' when the animation in object 'theObject' loops
    onLoop.add(functionWhenAnimLoop, theObject);
//call function 'functionWhenAnimComplete' when the animation in object 'theObject' completes
    onComplete.add(functionWhenAnimComplete, theObject);
```

Phaser – Animation Data

To create animation data:

1)Create sprites and put it in a folder. The sprites are better to be on each individual file

2)Use sstool or texturepacker to generate animation data

Phaser – Tween

Alter one or more properties of a target object over a defined period of time.

Some basic applications of Tweens in Phaser:

- Alpha fading Sprites

- Scaling

- Motion

Phaser – Tween

Tween.from(properties, duration, ease, autoStart, delay, repeat, yoyo): Sets a target to the destination value and tweens to its current value

Tween.to(properties, duration, ease, autoStart, delay, repeat, yoyo): A tween will start from the current value and tweens to the destination value given

Properties: A Javascript object containing the properties you want to tween, such as Sprite.x

Duration (optional): Duration of this tween in milliseconds. Or if Tween.frameBased is true this represents the number of frames that should elapse.

Ease (optional): Easing function. If not set it will default to Phaser.Easing.Default, which is Phaser.Easing.Linear.None by default but can be over-ridden.

Autostart (optional): Set to "true" to allow this tween to start automatically. The default value is "false" which calls Tween.start().

Delay (optional): Delay before this tween will start in milliseconds. Defaults to 0, no delay.

Repeat (optional): Should the tween automatically restart once complete? If you want it to run forever set as -1. This only effects this individual tween, not any chained tweens.

Yoyo (optional): A tween that yoyos will reverse itself and play backwards automatically. A yoyo'd tween doesn't fire the Tween.onComplete event, so listen for Tween.onLoop instead.

Resources

Phaser.IO

http://gamemechanicexplorer.com/#easing-1

http://www.html5gamedevs.com/topic/8618-phaser-easing-effects-list/

http://www.leshylabs.com/apps/sstool/

Phaser – Input (Mouse)

enableDrag(lockCenter, bringToTop, pixelPerfect, alphaThreshold, boundsRect, boundsSprite): Allow this Sprite to be dragged by any valid pointer. When the drag begins the Sprite.events.onDragStart event will be dispatched. When the drag completes by way of the user letting go of the pointer that was dragging the sprite, the Sprite.events.onDragStop event is dispatched. For the duration of the drag the Sprite.events.onDragUpdate event is dispatched. This event is only dispatched when the pointer actually changes position and moves. The event sends 5 parameters: sprite, pointer, dragX, dragY and snapPoint.

lockCenter (optional): If false the Sprite will drag from where you click it minus the dragOffset. If true it will center itself to the tip of the mouse pointer. Default: false

bringToTop (optional): If true the Sprite will be bought to the top of the rendering list in its current Group. Default: false

pixelPerfect (optional): If true it will use a pixel perfect test to see if you clicked the Sprite. False uses the bounding box. Default: false

alphaThreshold (optional): If using pixel perfect collision this specifies the alpha level from 0 to 255 above which a collision is processed. Default: 255

boundsRect (optional): If you want to restrict the drag of this sprite to a specific Rectangle, pass the Phaser.Rectangle here, otherwise it's free to drag anywhere. Default: null

boundsSprite (optional): If you want to restrict the drag of this sprite to within the bounding box of another sprite, pass it here. Default: null

spriteName.input.enableDrag();

Phaser – Input (Mouse)

input.allowHorizontalDrag: Controls if the Sprite is allowed to be dragged horizontally.
Default: true

```
spriteName.input.allowHorizontalDrag = false;
```

input. allowVerticalDrag: Controls if the Sprite is allowed to be dragged vertically.
Default: true

```
spriteName.input.allowVerticalDrag = false;
```

isDragged: Boolean. true if the Sprite is being currently dragged.

```
if(dragablePlatform.input.isDragged) { console.log("is dragged"); }
```

input.mousePointer.isDown: Boolean. Triggered when a mouse button is clicked

```
if (game.input.mousePointer.isDown) { Console.log(“A mouse is clicked”); };
```

Phaser – Input (Mouse)

Events: The Events component is a collection of events fired by the parent game object.

events.onDragStart.add(listener, listenerContext, priority)**:** Triggered when the object starts to be dragged.

events.onDragStop.add(listener, listenerContext, priority)**:** Triggered when the object stops to be dragged.

Listener: The function to call when this Signal is dispatched.

ListenerContext (optional): The context under which the listener will be executed (i.e. the object that should represent the `this` variable).

Priority (optional): The priority level of the event listener. Listeners with higher priority will be executed before listeners with lower priority. Listeners with same priority level will be executed at the same order as they were added (default = 0)

```
spriteName.events.onDragStart.add(dragStart);

spriteName.events.onDragStop.add(dragStop);
```

Phaser – Input (Mouse)

Create bounds for drag

boundsRect: A region of the game world within which the sprite is restricted during drag.

```
bounds = new Phaser.Rectangle(100, 100, 500, 400);
spriteName.input.boundsRect = bounds;
```

//setting world bounds

```
game.world.setBounds(-1000, -1000, 2000, 2000);
```

Phaser – Input (Keyboard)

input.keyboard.createCursorKeys(): Creates and returns an object containing 4 hotkeys for Up, Down, Left and Right.

cursors = game.input.keyboard.createCursorKeys();

input.keyboard.addKey(keycode): If you need more fine-grained control over a Key you can create a new Phaser.Key object via this method. The Key object can then be polled, have events attached to it, etc. The keycode could be any keyboard buttons.

alternateUpKey = game.input.keyboard.addKey(Phaser.Keyboard.W);

input.keyboard.removeKeyCapture(keycode): Removes an existing key capture.

game.input.keyboard.removeKeyCapture(Phaser.Keyboard.W);

input.keyboard.addKeyCapture(keycode): By default when a key is pressed Phaser will not stop the event from propagating up to the browser. There are some keys this can be annoying for, like the arrow keys or space bar, which make the browser window scroll. You can use addKeyCapture to consume the keyboard event for specific keys so it doesn't bubble up to the the browser. Pass in either a single keycode or an array/hash of keycodes.

game.input.keyboard.addKeyCapture([Phaser.Keyboard.LEFT, Phaser.Keyboard.RIGHT, Phaser.Keyboard.SPACEBAR]);

Phaser – Input (Gamepad)

Input.gamepad.start(): Starts the Gamepad event handling. This MUST be called manually before Phaser will start polling the Gamepad API.

```
game.input.gamepad.start();
```

input.gamepad.supported: Boolean. Whether or not gamepads are supported in current browser.

```
 if (game.input.gamepad.supported) { console.log(“gamepad is supported”); };
```

input.gamepad.active: Boolean. If the gamepad input is active or not

```
 if (game.input.gamepad.active) { console.log(“gamepad is active”); };
```

input.gamepad.pad1.connected: Boolean. Whether or not this particular gamepad is connected or not.

```
 if (game.input.gamepad.pad1.connected) { console.log(“gamepad pad1 is connected”); };
```

Phaser – Events

Events (normally on the update function state):

Phaser.Pointer.isDown: Boolean. If the Pointer is touching the touchscreen, or the mouse button is held down, isDown is set to true.

```
if (game.input.activePointer.isDown)   {       console.log(“mouse clicked”);    }
```

Phaser.Key.isDown: Boolean. The "down" state of the key.

```
if (cursors.up.isDown)    { console.log(“button up is pressed”); };
```

Phaser.Gamepad.isDown(buttonCode): Returns true if the button is currently pressed down, on ANY gamepad. buttonCode is the button to check for and it returns True if a button is currently down.

```
if (pad1.isDown(Phaser.Gamepad.XBOX360_DPAD_LEFT))  { console.log(“Gamepad
    DPAD left is pressed”); };
```

Phaser – Events

Events (can be declared on the create function state):

```
Phaser.Input.onTap.add(listener, listenerContext, priority)
Phaser.Input.onDown.add(listener, listenerContext, priority)
Phaser.Key.onDown.add(listener, listenerContext, priority)
Phaser.GamepadButton.onDown.add(listener, listenerContext, priority)
```

```
game.input.onTap.add(theTapFunction, this);
game.input.onDown.add(anyInputFunction, this);
alternateUpKey.onDown.add(theAlternateUpKeyFunction, this);
gamepadButtonKey.onDown.add(theGamepadButtonFunction, this);
```

Phaser – Events

Events for button (can be declared on the create function state):

```
Phaser.Button.onInputOver.add(listener, listenerContext, priority)
Phaser.Button.onInputOut.add(listener, listenerContext, priority)
Phaser.Button.onInputUp.add(listener, listenerContext, priority)
```

```
button.onInputOver.add(overFunction, this);
button.onInputOut.add(outFunction, this);
button.onInputUp.add(upFunction, this);
```

Phaser – Camera

Phaser.Camera.follow(target,style): A Camera is your view into the game world. It has a position and size and renders only those objects within its field of view. The game automatically creates a single Stage sized camera on boot. Move the camera around the world with Phaser.Camera.x/y

Target: The object you want the camera to track. Set to null to not follow anything.

Style: Leverage one of the existing "deadzone" presets. If you use a custom deadzone, ignore this parameter and manually specify the deadzone after calling follow(). Options are:Phaser.Camera.FOLLOW_LOCKON, Phaser.Camera.FOLLOW_PLATFORMER, Phaser.Camera.FOLLOW_TOPDOWN, Phaser.Camera.FOLLOW_TOPDOWN_TIGHT);

```
game.camera.follow(ufo, Phaser.Camera.FOLLOW_TOPDOWN_TIGHT);
```

Phaser – Camera

Setting boundary for camera

Phaser.Camera.deadzone: The deadzone is a Rectangle that defines the limits at which the camera will start to scroll. It does NOT keep the target sprite within the rectangle, all it does is control the boundary at which the camera will start to move. So when the sprite hits the edge, the camera scrolls (until it reaches an edge of the world)

```
game.camera.deadzone = new Phaser.Rectangle(100, 100, 600, 400);
```

Resources

http://phaser.io/examples/v2/category/input

http://phaser.io/examples/v2/category/buttons

http://phaser.io/examples/v2/category/camera

http://phaser.io/learn/chains