Introduction to the Ultimate Game Parts Template

Over the years I've made several templates for Phaser and while I've been satisfied with all of them, I felt it was time for a new one. Why? Mostly because I've learned a lot of techniques and shortcuts in the last several years. One of the most popular templates that I have is the utility template which works a lot with rescaling games.

And while that's great, I found that I was using the same combinations of code over and over again especially when it came to placing game objects like text or images onto the canvas. So I have combined these steps into several common functions that will allow you, the developer to perform a lot of these tasks in one line of code.

Another reason is the rise of the Node Package Manager for use on JavaScript Projects. I wanted something that could meet the needs of the modern programmer.

This particular template is especially good for casual games that need just a start screen and a game over screen and the main playing screen.

I plan on publishing modules later to be able to expand the template to make it more versatile to other games.

Below is all the documentation that you need to get started and if you have any questions please don't hesitate to contact me. I love talking about code!

It is important to know that this template is built on top of another one made by Richard Davey the creator of Phaser. My template is code that supplements his.

If you haven't already used git or NPM, there are some prerequisites that you need first.

Step 1. Install the node package manager

What is the node package manager(NPM)?

According to the official website:

npm makes it easy for JavaScript developers to share and reuse code, and makes it easy to update the code that you're sharing, so you can build amazing things.

For if it makes things easier or not, is debatable, however it is now being widely adopted

as standard.

Install here

Step 2. Install Git

What is git?
Git is a free and open source distributed version control system

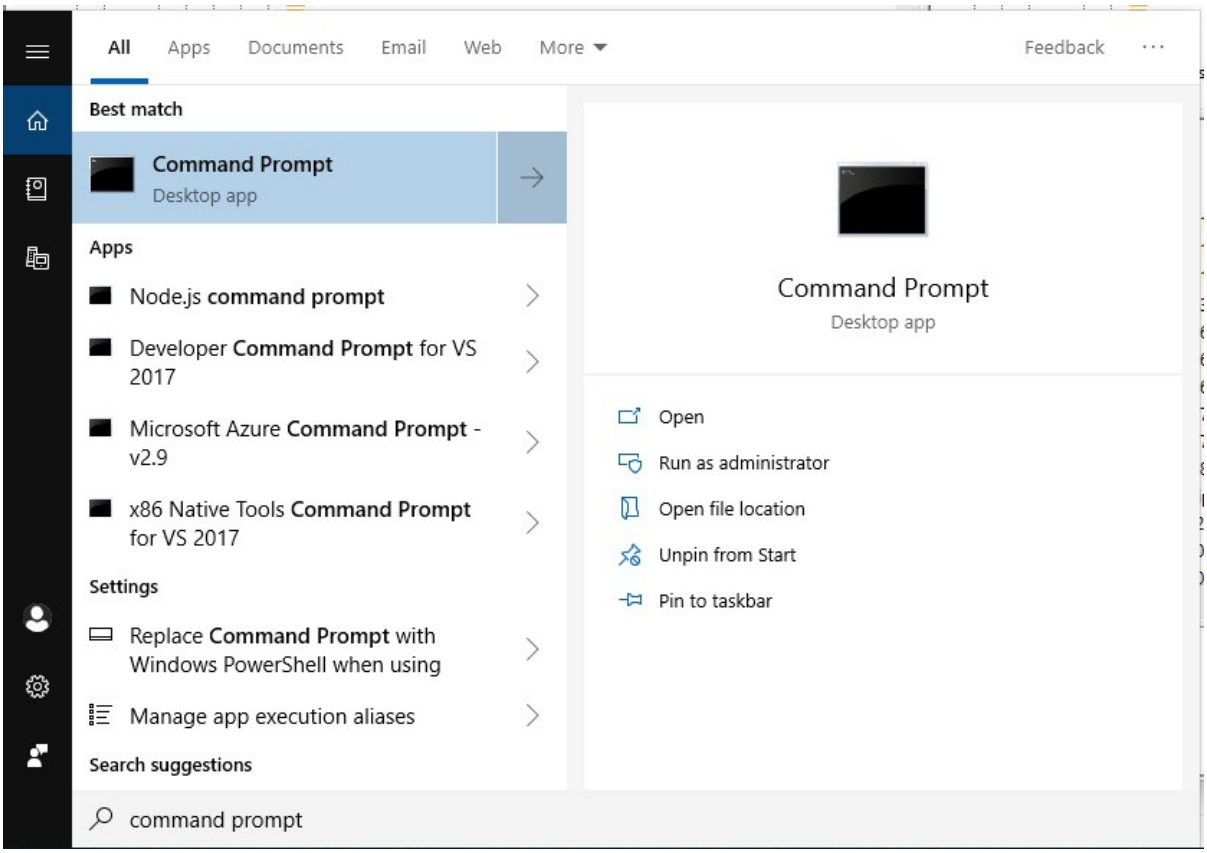
Basically git copies files from one place to another and keeps track of what was copied and when, so it is easy to roll back to a previous time.

We need git here to copy files from github.com

Install here

Step 3. Open A command prompt

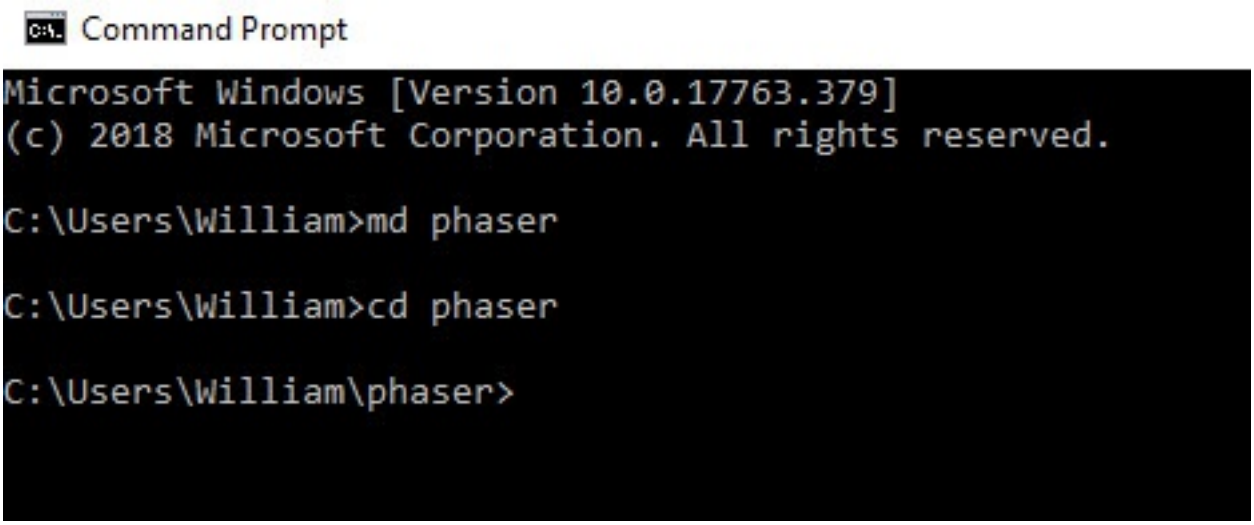
Open a command prompt on Windows, or a terminal on Macs. You may need to search for it. Once you find it, make a shortcut where you can get to it quickly in future.



Step 4. Make a directory

At the command prompt, make and navigate to an empty directory. I like to make directories for each of my project types live. Phaser, React, Vue, ect.

Use md for make directory and cd for change directory.



Step 5. Clone The Project

Now we clone the files from the phaser 3 project template git hub. Copy and paste the text below into your command prompt, replacing game1 with the name of your game or project.

Type or copy:

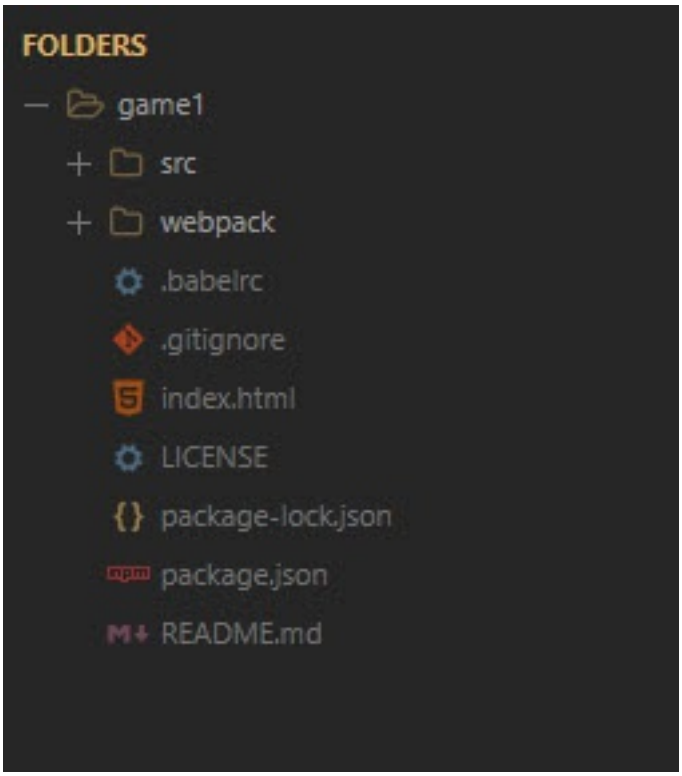
```
_____
```

Then change the directory to your game directory

```
cd game1
```

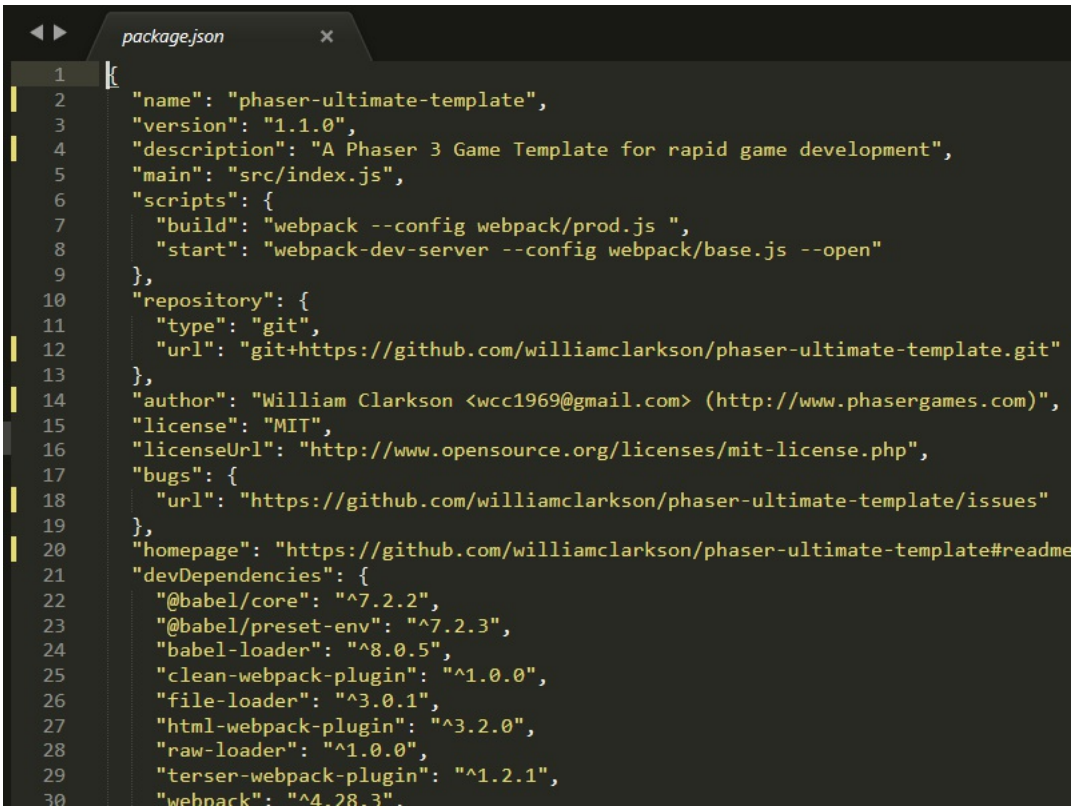
Step 6. Open your directory in your editor

Open the folder with the code editor of your choice. You'll see something like this.



Step 7. Customize the package file - optional

Open the package.json that was included with the template and replace infomation such as author name, homepage etc with your own data.



About WebPack

It was difficult for me to find anything on WebPack on the web that made me understand what it was.

webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging

https://webpack.js.org/

In plain language basically webpack combines all your scripts into 1 script, and working with Babel, translates that down a low level JavaScript that all browsers understand. It also does the same for all css fles and images.

Normally we would use npm to install webpack, but The project template that we cloned in step 5, already contains webpack as dependency so we will install it in the next step.

Step 8. Install the dependencies

There are bits of code that the template needs to run aka the node packages. The template has a list of what is needed.

To install them type:

```
npm install
```

Step 9. Run the code

To start the Phaser Project type out

npm start

If all is working then you will see a bouncing Phaser Logo

Step 10. Install the Ultimate Games Part Code

Take the code you downloaded and paste it into the main code folder

Don’t worry if files are overwritten

The template is set up to make loading assets easier. In sceneLoad you can define folders to use for images and audio. These can be relative to the project or remote or remote servers;

Default - in project access

```
this.common = "./assets/";
this.imagePath = this.common + "images/";
this.audioPath = this.common + "audio/";
```

For a remote server, simply change the common path, and folder names if needed

```
this.common="";
```

Different arrays will point to diffent paths or file types

```
let iconArray = ['gear', 'musicOff', 'musicOn', 'sfxOn', 'sfxOff'];

let pngArray = ['panelBack','title'];

let mp3Array = [];
```

Utilities

The following utilities are integrated into many functions inside this template. However if you'd like to know how to use them by themselves, Here is the information to do so.

Utilities

- Align Grid
- Aligner
- Font Loader
- Image Loader

Align Grid

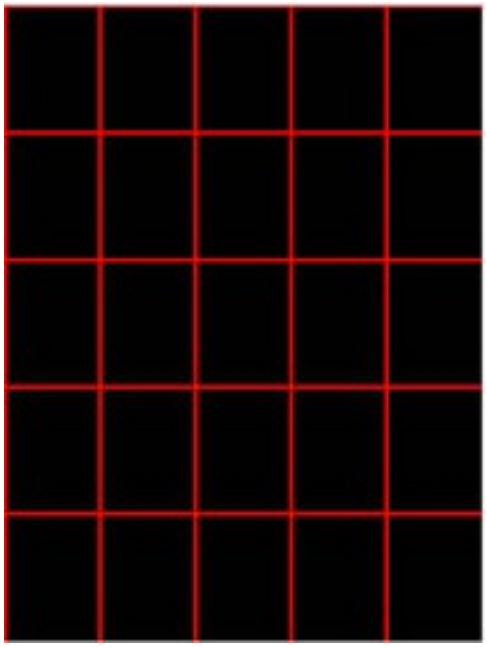
The Align Grid Utility class was designed to help with the layout of images across devices with different screen sizes. The class will create a grid of cells of equal widths and height. The spacing will remain constant regardless of the screen size. You can set the number of rows and columns and choose which object to place the grid on. If you omit an object the component will use the canvas.

This was originally designed to be used to lay out UI elements, but I've found it to be useful as a display grid in a number of games.

It is at the core of this template and most things rely on it

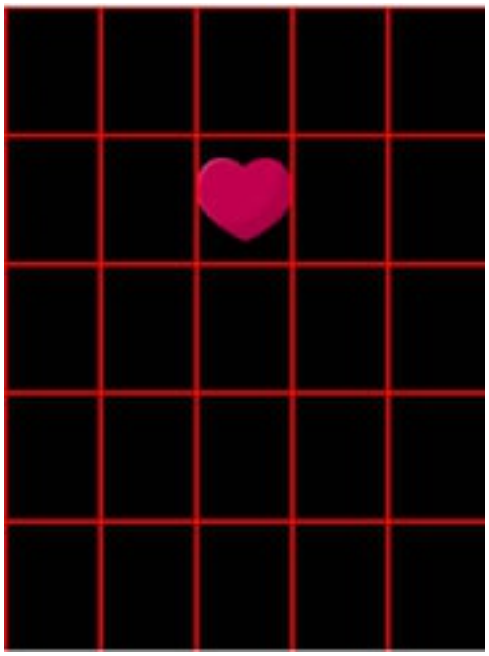
Basic Usage:

```
import {
    AlignGrid
} from "../common/util/alignGrid";

//make alignment grid of 5 columns by 5 rows
 let agrid=new AlignGrid(5,5);
 //show for debug
 agrid.show();
```



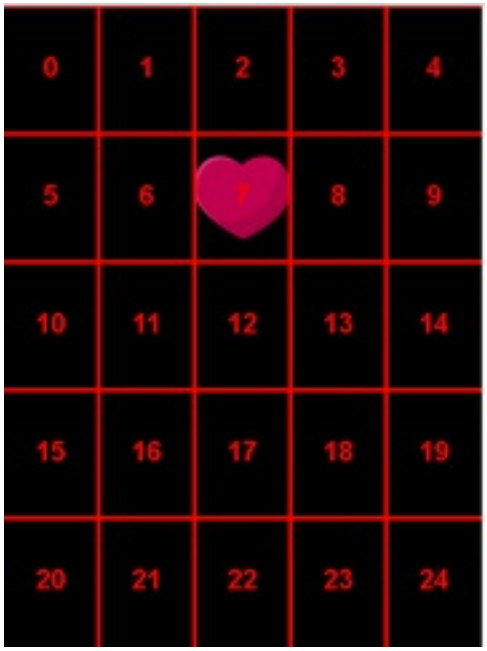
To place a sprite on the grid

```
//make a sprite
var heart=game.add.image(0,0,"heart");
heart.anchor.set(0.5,0.5);

//make alignment grid
var agrid=new AlignGrid(5,5);
//show for debug
agrid.show();
//place at column 2, row 1
agrid.placeAt(2,1,heart);
```

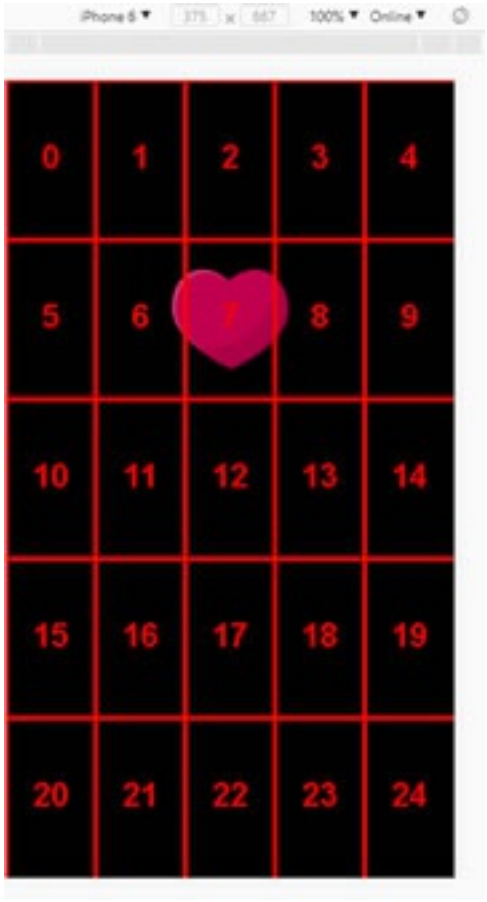


Alternatively you can also use an index number instead of a row and column

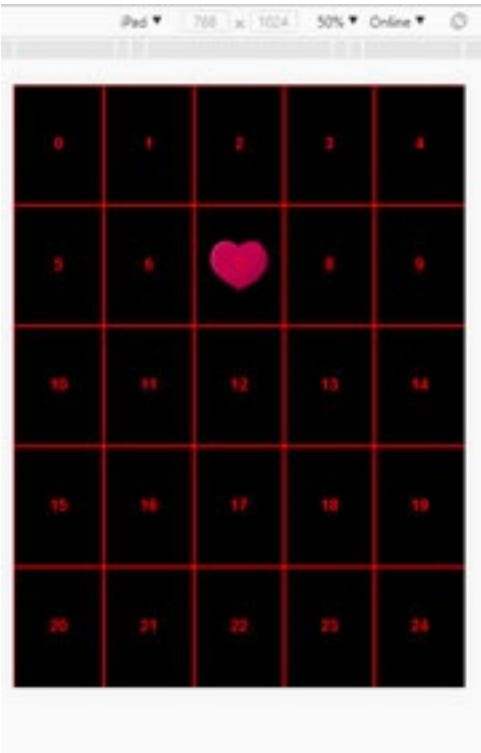
```
var heart=game.add.image(0,0,"heart");
heart.anchor.set(0.5,0.5);

//make alignment grid
var agrid=new AlignGrid(5,5);
//show numbers for debug
agrid.showNumbers();
//place at index
agrid.placeAtIndex(7,heart);
```



View on Iphone

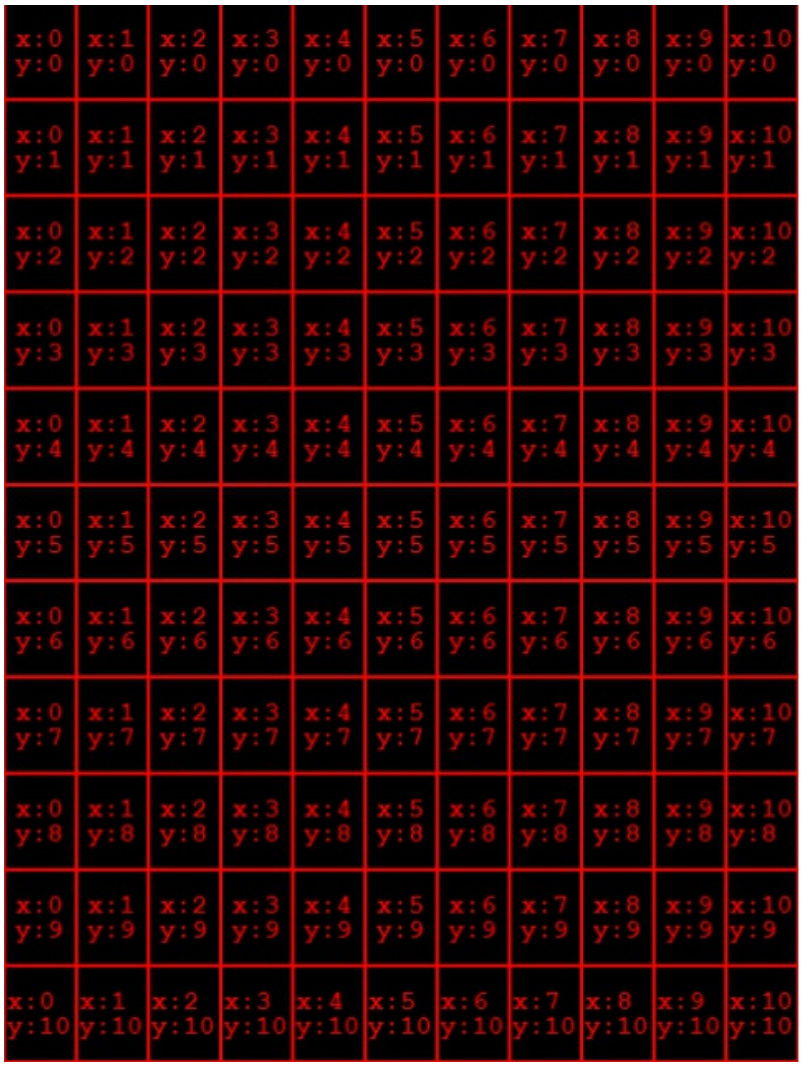

iPhone 6
375 x 667
100%
Online
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24

View On Ipad



To make good use of this, it helps to also scale the elements

You can also view the x and y positions by using this.aGrid.showPos();



Aligner

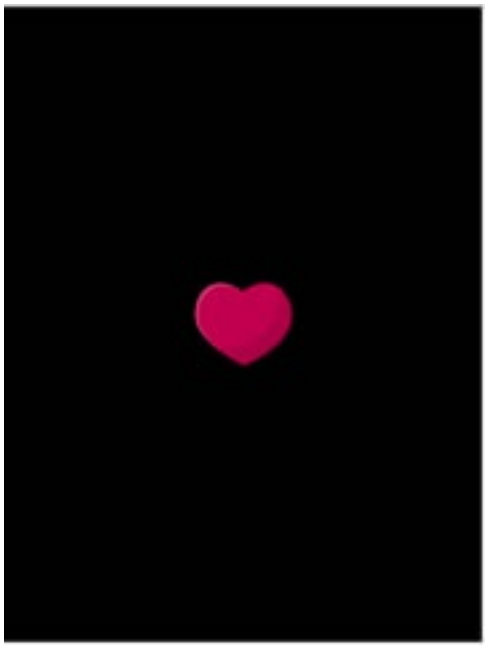
The Aligner is a collection of common functions needed to

place objects on stage in certain positions.

Center in the middle of the stage

```
import {
    Align
} from "../common/util/align";
```

```
var heart = game.add.image(0, 0, "heart");
heart.anchor.set(0.5, 0.5);
Align.center(heart);
```



For Groups, Template Components, or sprites with a 0 anchor settings use

```
var heart = game.add.image(0, 0, "heart");
heart.anchor.set(0, 0);
```

```
Align.centerGroup(heart);
```

You can also center veridically or horizontally

```
Align.centerV(heart);
Align.centerH(heart);

Or for groups

Align.centerHGroup(heart);
Align.centerVGroup(heart);
```

Scaling

```
  //get the width ratio of object to game size
  //this will log to the console
  Align.getScaleToGameW(heart);

  //logs 0.20416666666666666

  //set the width based on the object to game ratio
  //this will keep the size constant in relation
  //to the stage
  Align.scaleToGameW(heart,0.2041666);


  Alternatively you may use height:
  Align.getScaleToGameH(sprite);
  And
  Align.scaleToGameH(spirte,ratio);
```

99% of the time I personally use the width of the game and object.

Font Loader

The font loader loads Goggle Fonts to use in your text.

This is a file that is loaded by the index.html file and there is no need to import it.

Simply add the names of the Google Fonts to the array in the file.

```
google: { families: ["Fresca","Flamenco","Indie Flower"] }
};
```

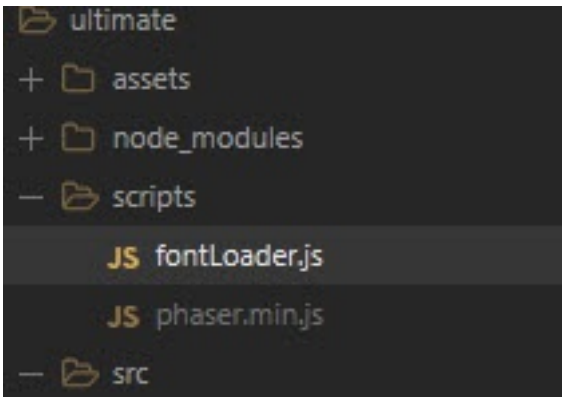
Then use the fonts in the text styles file

```
mainFont = "Flamenco";
```

Or on your text fields

```
var text1=game.add.text(100,50,"My Text");
text1.font="Flamenco";
```

File location



ImageLoader

Most of the time we use the preload function in a state to load the images but sometimes we need to be able to load images at runtime and don't have the luxury of knowing what that image will be at the start of the game. Sometimes we need to get information from a server or to load a image url based on user input. This is where the ImageLoader comes in.

The first step is to create an ImageLoader instance and

pass in a scene and callback function via an object.

1. Create an Instance

```
const imageLoader=new
ImageLoader({scene:this,callback:this.imageLoaded.bind(this)});
```

2. Add the callback into the scene. The scene will be passed the instance of the loader.

```
imageLoaded(loader)
{
   //load the image based on the key from the loader
   this.add.image(100,50,loader.key);

}
```

3. Call the load function passing in a key and a path

```
imageLoader.load(”ninja”,”images/ninja.png”);
```

BaseScene Class

Each scene extends a BaseScene class;

The template has several scenes already but if you need a new one here is the code you need to set up a new scene

```
export class SceneExtra extends BaseScene {
    constructor() {
        super('SceneExtra');
    }
    preload() {}
    create() {
       super.create();
    }
    update(){ }

    }
```

Set Up A Scene

To use the build in features of the BaseScene there are a couple

of steps that have to be done first

1. Call the create in the BaseScene Class

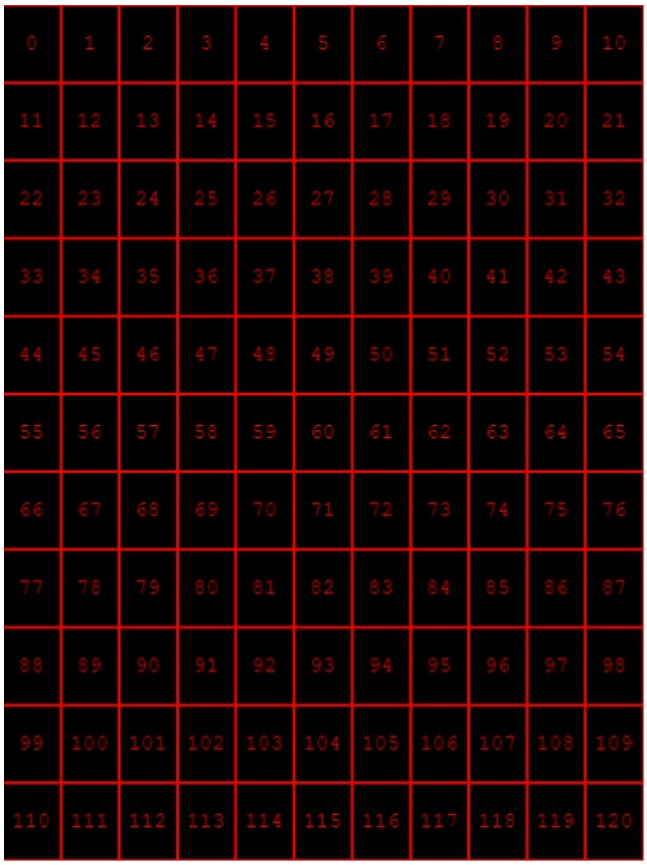
```
super.create();
```

2. Set the grid size

```
this.makeAlignGrid(width,height);
```

Here is an example below

```
export class SceneMain extends BaseScene {
    constructor() {
        super('SceneMain');
    }
    preload() {}
    create() {

        //set up the base scene
        super.create();

        //set the grid for the scene
        this.makeAlignGrid(11, 11);

        //show numbers for layout and debugging
        this.aGrid.showNumbers();


      }
      update(){ }

      }
```



Set the screen's background

Inside any scene that extends the base scene including all the pre-prepared scenes that come in this template, you can use the setBackground function to quickly set an image for the background of that particular screen

Usage

```
this.setBackground(key);
```

Example

```
this.setBackground('sky');
```

Adding Images

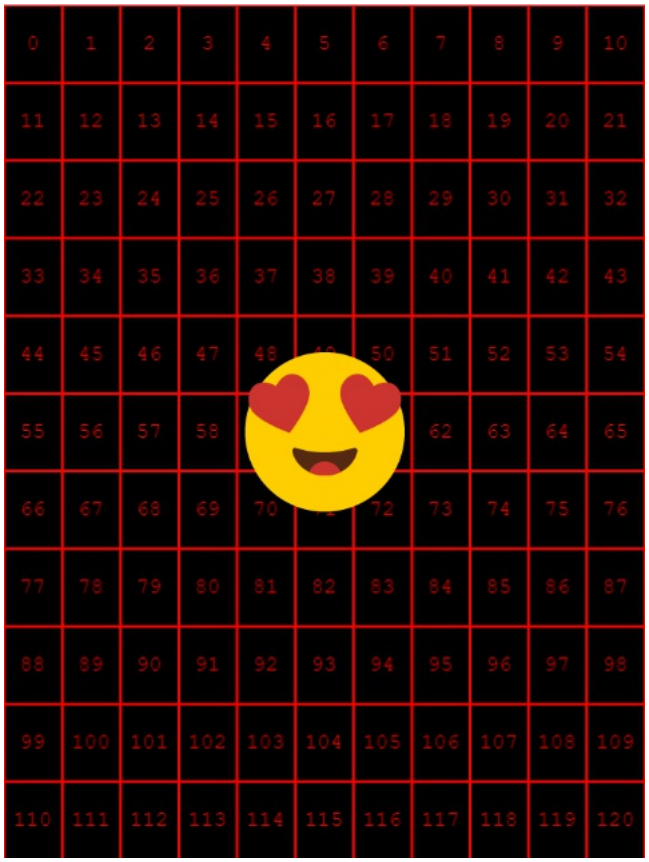
Use **placeImage** to add an image on the grid and scale it.

Usage

```
this.placeImage(key,grid_number,percentage_of_screen_width)
```

Example

```
this.placeImage("face",60,.25);
```



Advanced:

You can also place an image by the column and row number

```
this.placeImage("face",{x:4,y:7},.25);
```

You can also fine tune a position by using decimals

```
this.placeImage("face",{x:4.5,y:7.3},.25);
```

Adding Text

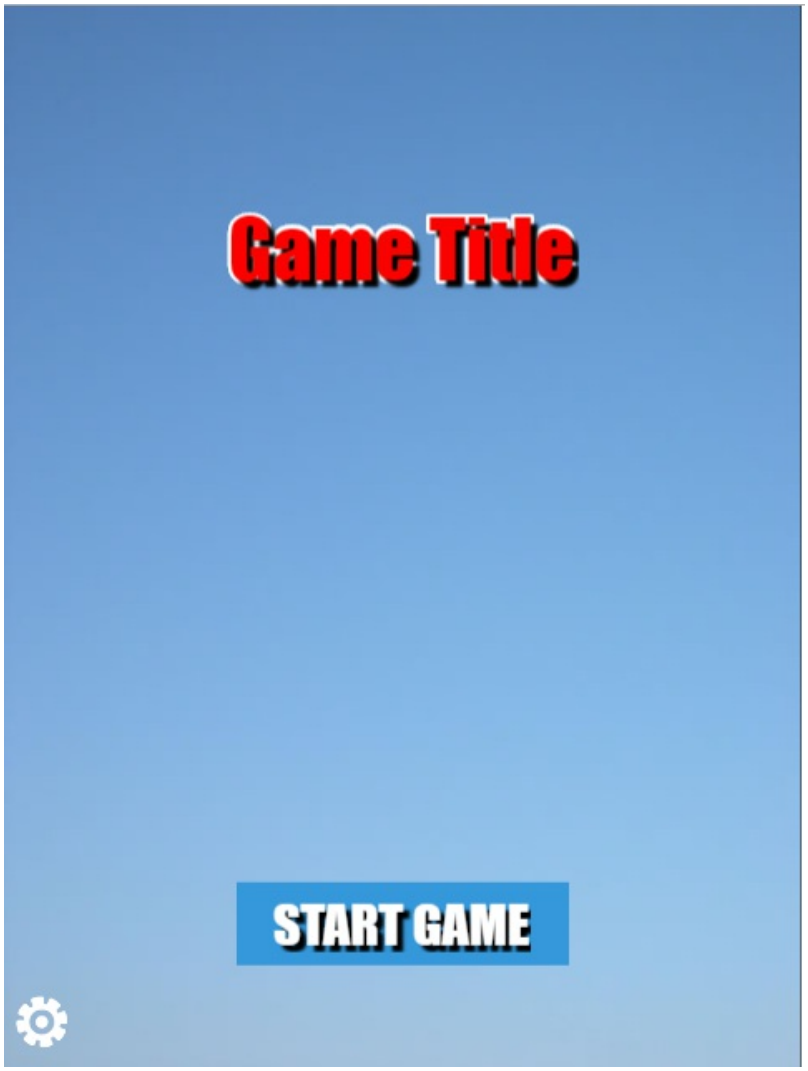
You can place text on the grid by using placeText.

Usage

```
this.placeText(text,grid_number,text_style);
```

Example

```
this.placeText("Game Title",27,"TITLE_TEXT");
```



Using and Adding Text Styles

Text Styles allows you to maintain a library of styles for text fields across the game without having to copy and paste the style code every time you want to use the style in a new location. You can also globally change styles on text fields by updating that style.

Using a text style

To use a text style, just pass it as the 3rd parameter when using placeText

this.placeText("Game Title",27,"TITLE_TEXT");

Defining a text style

To define a simple text style

this.textStyles.regSimple(key, color, fontSize, font)

Key is the name of the text style such as TITLE_TEXT or LIGHT_GREEN

Color is passed as a string with a hex "#ff0000" for red text for example

FontSize is how large you want the text to be.

You can also pass in these constants which are percentage of the width of the game and will scale on different devices

TextStyles.SIZE_VERY_LARGE
TextStyles.SIZE_LARGE
TextStyles.SIZE_MED
TextStyles.SIZE_MED2
TextStyles.SIZE_MED3
TextStyles.SIZE_SMALL
TextStyles.SIZE_SMALL2

If you omit fontSize it will default to TextStyles.SIZE_MED

Font is a string of a font name that you should load via the FontLoader

If you omit a font it will default to TextStyles.MAIN_FONT which you can edit in the textStyles.js file

Advanced

this.textStyles.regAdvanced(key, color, fontSize, font, stroke, strokeThick, shadow)

You can also register an advanced style by using the same values as above and also passing in a stroke color, stroke thickness and a shadow color

Stroke color defaults to #ff0000
Stroke thickness defaults to 4
Shadow color defaults to #000000

Examples

```
  this.textStyles.regSimple("SCORE", "#ffff00",
TextStyles.SIZE_LARGE, "Impact");

  this.textStyles.regAdvanced("SCORE2", "#ffff00",
TextStyles.SIZE_LARGE, "Impact", "#00ff00", 4, "#ffffff");


  this.placeText("score:0",16,"SCORE");
```

Sound Control Panel

The template includes a sound panel for turning on and off background music and sound effects these two lines of code will add a button to open the panel and the panel itself.

Place this in the create function of your scene

```
  super.makeSoundPanel();
  super.makeGear();
```

The Event Dispatcher

The template works on events. The eventDispatcher variable which you will see referenced throughout this documentation, is a global instance of Phaser.signal which broadcasts events to different components.

The format for dispatching an event is

eventDispatcher.dispatch(event name,params)

Parameters:
Event name:a string.
Params:any object, usually a string or number, but sometimes an complex object such as {title:"My Title",message:"something to say"}

Example:

```
eventDispatcher.dispatch("UP_SCORE",10);
```

Will increase the user's score by 10 points.

The Controller

The controller is the main listener for events, and controls much of the built in template functions.

You shouldn't need to access this directly, but looking at the code may point you to some of the features of the template.

Built In Events

There are several built in events that you can call in your game

You can also call the functions directly inside the components, but you might find this useful

SCORE EVENTS

The score events affect the scoreBox component and the model score property

UP_SCORE - updates the score

```
eventDispatcher.dispatch(G.UP_SCORE,10);
```

SCORE_UPDATED - triggered when the score changes, used in the scoreBox

TIME EVENTS

The time events affect the clock component

SET_TIME -sets the time on the clock in seconds
eventDispatcher.dispatch(G.SET_TIME,10);

STOP_TIME-stops the clock
eventDispatcher.dispatch(G.STOP_TIME);

START_TIME-starts the clock
eventDispatcher.dispatch(G.START_TIME);

TIMES_UP-dispatched by the clock when the time runs out

ADD_TIME-adds time to the clock

RESET_TIME-resets clock to the value passed with SET_TIME

MEDIA EVENTS

TOGGLE_MUSIC - toggles the music on or off
eventDispatcher.dispatch(G.TOGGLE_MUSIC);

TOGGLE_SOUND - toggles on or off the ability to play sound effects
eventDispatcher.dispatch(G.TOGGLE_SOUND);

MUSIC_STAT_CHANGED - dispatched when the music is toggled

PLAY_SOUND - plays a sound
eventDispatcher.dispatch(G.PLAY_SOUND, sound_key);

Custom Events

You can also send custom events by setting a unique string

eventDispatcher.emit("my_custom_event");

The second parameter is extra information that can be of any type

```
    eventDispatcher.emit("chooseColor","red");
    eventDispatcher.emit("shootFire",
{'type':'blue','power':20};
```

Listen For Events

The components such as the clock or the scoreBox already have event listeners added, and all you need to is dispatch an event.

However, if you are using custom events, or would like to know when any event has been fired, such as the score increasing you'll need to add add an event listener to the eventDispatcher

```
this.emitter = EventDispatcher.getInstance();

this.emitter.on(this.gotEvent, this);
```

And the callback function should be like this

```
 gotEvent(call, params) {
        if (call == "test") {
            //your code here
        }
    }
```

Flat Button

I created the button classes, because I was losing a lot of time on making graphic assets, especially when just needed to change the text on a button. They are extremely helpful for prototyping but can also be used for the final game

A flat button consists of a background image and a text overlay. You can use any image for the background but some are included with the template.

To add a button to the scene

```
//
//flat button
//
let btnNext = new FlatButton({
    scene: this,
    textStyle: 'BUTTON_STYLE',
    key: "button",
```

```
    text: "START GAME",
    callback: this.startGame.bind(this)
});
```

Parameters:

Scene-the current scene, usually 'this'
TextStyle-the text style key
Key-the image key
Text-text to display
Callback-a function

You can load a preset button in scene load with this code:

```
this.loadButton(key,set,index);

this.loadButton("button", 1, 2);
```

Set 1

Set 2

Toggle Buttons

The toggle button is used for the sound panel but if you need one you can add your own

```
let btnMusic = new ToggleButton({
    scene: this.scene,
    backKey: 'toggle2',
    onIcon: 'onIconKey',
    offIcon: 'offIconKey',
```

```
        event: 'TOGGLE_MUSIC',
        scale: .2,
        value: true,
        x: 0,
        y: 0
    });
```

Parameters:

Scene-the current scene, usually ‘this’
backKey-the image key
OffIcon-icon image key for the off state
OnIcon-icon image key for the on state
Text-text to display
Callback-a function which will be passed a true or false value

Included icons

You can choose which icons to pre-load by adding the file names to the iconArray in sceneLoad

```
let iconArray = ['gear', 'musicOff', 'musicOn', 'sfxOn', 'sfxOff', 'iconLock', 'iconHome', 'iconNext', 'iconPrev'];
```

Components

The template includes a couple of game components to help you build your game faster.

These include:

- Score box
- Clock
- Background
- Bar
- Point Box

Score Box

The score box is handy when you need to increase the score outside the main scene. It works on using the eventDispatcher to send messages to the controller and updating the score in the model. You can also call the methods directly

Add the score box

```
//
//scorebox
//
this.sb = new ScoreBox({
    scene: this
});
this.placeAtIndex(27, this.sb);
```

Events

```
//
//set the score
//
this.eventDispatcher.emit("SET_SCORE", 100);
//
//up the score
//
this.eventDispatcher.emit("UP_POINTS", 1);
```

Methods

```
upScore(points)
scoreUpdated()
```

Clock

The clock is a nice component for any time based game. You can dispatch events to control the clock or call the methods directly.

Add the clock

```
//
//clock
//
```

```
this.clock = new Clock({
    scene: this
})
this.placeAtIndex(38, this.clock);
//
//
//clock events
//
//
this.eventDispatcher.emit("SET_TIME", 60);
this.eventDispatcher.emit("ADD_TIME", 1);
this.eventDispatcher.emit("STOP_TIME");
this.eventDispatcher.emit("START_TIME");
this.eventDispatcher.emit("RESET_TIME");
```

Methods

```
startClock()
stopClock()
setClock(seconds)
addTime(seconds)
ResetClock()
```

Points Box

The points box is a bit of text that will display the number of points received and then fly upwards off the screen.

Usage

```
let pointBox=new PointBox({scene:this,points:number_of_points,x:x_position,y:y_position});
```

Example

```
let pointBox=new PointBox({scene:this,points:10,x:240,y:500});
```

Progress Bar

The bar component was made for the loading screen

progress display but it can also be used for health meters or power displays

Example

```
        this.bar = new Bar({
                    scene: this,
                    height: 100,
                    width: 300,
                    color: 0xffffff
                });

        this.bar.setPercent(25);
```

Effects

The template contains a few effects for your game.

Most of the effects need their images loaded before being used.

This is normally set up in stateLoad or in the preload of a state

Call the static preload function of the effect to preload the images

```
Flare.preload();
StarBurst.preload();
ColorBurst.preload();
```

Sparks and Ripples use graphics instead of images so no preload is needed

Spark Effect

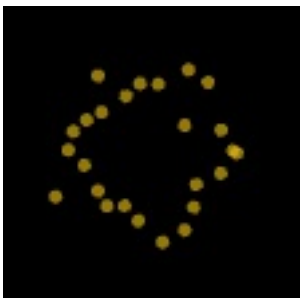
Example

```
let sparks=new Sparks({scene:this,x:pointer.x,y:pointer.y,size:5});
```

Parameters

x,y position
color-color of the dot, default white
size-radius of the circles -optional

count-the number of circles - optional



Ripple Effect

```
let ripple=new
Ripple({scene:this,x:pointer.x,y:pointer.y});
```

Parameters

x,y position
count-number of dots that make up the circle optional
dist-how far the dot will travel before disappearing optional
color-color of the dot

Star Burst

A star burst is a great effect when the player has done something right

Example

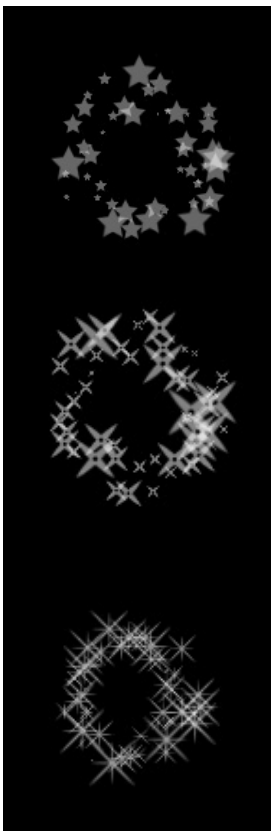
```
let stars = new StarBurst({
    scene: this,
    x: pointer.x,
    y: pointer.y,
```

```
        f: 1,
        tint: 0xffcc00
    });
```

Params:

```
    x,y the position of the effect
    f is frame number of the sprite sheet change to get
different shapes - optional
    count- the number of stars - optional
    tint-the color of the stars - optional
```

Example outputs



Color Burst

The ColorBurst adds a shower of color stars to your game that move out from the center and fade

Example:

```
let colorStars = new ColorBurst({
            scene: this,
            x: pointer.x,
            y: pointer.y,
            count:50,
            size:5
        });
```

x,y position

color-color of the dot, default white
size-radius of the stars -optional
count-the number of stars - optional

Will produce this effect:

Flare Effect

A flare effect scales up rotates and then scales down and fades

Example:

```
let flare=new
Flare({scene:this,x:pointer.x,y:pointer.y,direction:1,duration:5
00,tint:0xffffff});
```

x,y the position of the effect
direction 1 for clockwise, -1 for counter-clockwise
duration time it takes the flare to spin and fade in milliseconds
Tint-color of the flare

Example