HTML5 and JavaScript Games Programming

Week 1

Mario.Soflano@uws.ac.uk

Module Overview

- This module focuses on web games development
- The skill to be learnt in this module:
  - HTML5
  - JavaScript
  - CSS
  - JSON
  - Phaser
- At the end of this module, the students are expected to be able to develop their own web-games

Module Contents

- Week 1: Induction Week (4-8th Sept 2017)
- Week 2: Module and Game Frameworks Introduction(12th - 13th Sept 2017)
- Week 3: HTML5 Elements (19th – 20th Sept 2017)
- Week 4: JavaScript – Revisit (26th – 27th Sept 2017)
- Week 5: JavaScript - Object-Oriented (3th – 4th Oct 2017)
- Week 6: Phaser - Initialisation and Basic Elements (10th – 11th Oct 2017)
- Week 7: Phaser – Game environment (17th – 18th Oct 2017)
- Week 8: Phaser – Character Animation, Input and Camera (24th – 25th Oct 2017)
- Week 9: Phaser – Physics, Filters and Particles (31th Oct – 1st Nov 2017)
- Week 10: Web game and Database (7th – 8th Nov 2017)
- Week 11: Lab only (14th – 15th Nov 2017)
- Week 12: Lab only (21st – 22nd Nov 2017)

Module Overview

- 1 hour lecture and 3 hours lab per week
- Assessments
  - Demo as progress report on Week 5 and week 11 (10%)
  - Develop a website to showcase the game including game documentation (20%)
  - Develop a web game using JavaScript Games Framework (70%)

Assessment 1 - Website

- To promote your game
- Can use any template or software to build the website
- The website has to be able to host the game
- Structure of the website:
  - A main page (a banner, screenshots, short introduction of the game)
  - A game page (to host the game)
  - A game design page (the source of inspirations, details on the game design, interface and game mechanics, features implemented)
  - A team / developer page (individual roles, development minutes, development reflection)

Assessment 1 – Marking Scheme

- Website Design (5%)
  - Aesthetic: how does the website look like? Are the contents easy to read and understand?
  - Relevancy: is the website design, such as colour scheme, background, banners and icons, relevant to the game?
  - Structure: does it has a clear structure? are all sections connected and accessible?

Assessment 1 – Marking Scheme

- Game Documentation (15%)
  - Inspiration and comparison: how do you come up with the idea? Is there any existing games similar to your idea? How is your game similar and different?
  - Game Specification: what is the genre of the game? If there is, what is the story of the game? How is the gameplay of the game? How is the mechanic of the game? What is the objective of the game? What kind of experience do you expect from the player to feel when playing the game? What are the features you adopt from the other games? What are new features you add?

Assessment 2 - Game

- Design and develop a game of any genre by using Phaser
- Assessment criteria:
  - Features(40%): What are the features implemented? How complex and creative the features implemented compared to the existing games? If there is a story to be told in the game, is the story interesting and has it successfully conveyed to the player? Are there relevant music and sound effects implemented? how complex / creative does the implementation?
  - Usability and Interactivity (20%): Can the game be played immediately? Is there any instruction on how to play the game? Are the sections of the game are distinguish of each other (menu, pause, play, game over)?
  - Interface (10%): How does the game look like (background, colour pattern etc)?

FAQs

- Can I use Wordpress or Weebly?

  I would discourage the use of Wordpress. You are expected to build your own website and the submission has to include the HTML, CSS and JavaScript you created. Also you would gain more benefit in learning PHP and MySQL to attach to your own website.

  However, you can use any tool as long as:

  - You can submit the HTML, CSS and JavaScript including other supporting files you created
  - The game needs to be embedded / included in the website
  - If you are going to use server-side scripting, you need to make sure the tool allows you to do so

FAQs

- Can I use website generator like dreamweaver or webstorm?

  You can use any tools to help you building your website. Please remember that you need to submit HTML, CSS, JavaScript and other supporting files of the website (and the game) you created

- Can I use different framework?

  The main framework you should use is Phaser but you can add other framework as supplementary if you need

- Do I have to stick to the number of page suggested?

  No. You are welcomed to add as many pages as you see fit

- Can I add more description or information of the game I create?

  Yes. In fact I encourage you to include more information and description of the project. After all, the reason why I suggested to create a website is because you can use it as part of your portfolio to promote your project as well as yourself

FAQs

- Can I work in group?

  You could work in pair but I would encourage you to work individually.

- What is "Expected experience" in the specification?

  "Expected experience" is what you would want the user to feel when playing a game. Is it for a laugh? Maybe suspense? Maybe thrilling?

- What is “the objective” in the specification?

  "The objective" is what you would want the user to gain from playing the game. Maybe the game is educational therefore the player can learn something? Or perhaps you already conduct research on current similar games and you found some aspects you can improve, you can address that the intended goals of the game is to improve so on and so forth of the current similar games.

- What are files are to be submitted?

  All HTML, CSS, JavaScript and all supporting files (image, audio). You DO NOT have to submit all web server files; only submit the files that you created and assets that you are using for the project.

Module Materials

- Online Resources:
  - http://phaser.io/
  - http://www.w3.org/TR/html5/
  - https://jquery.com/
  - http://www.w3schools.com
  - https://jsfiddle.net/
- Recommended Texts:
  - HTML5 Game Development for Dummies, Andy Harris
  - Learning HTML5 Game Programming, James L. Williams
  - The HTML5 Canvas (for Games and Entertainment), Rob Hawkes

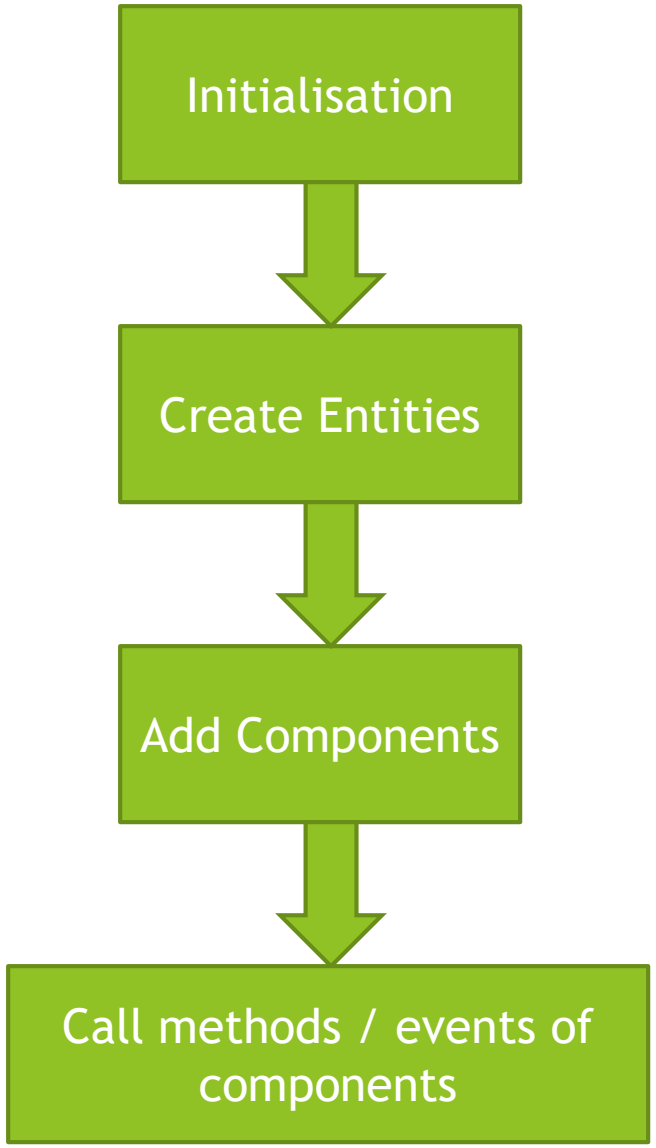
HTML 5 Games

- HTML5

Crafty

- Crafty is an open-source JavaScript game library developed by Louis Stowasser.
- It renders to either Canvas or DOM
- Aim for light-weight and simple use
- Cross-browser compatible
- Sprite map support
- Simple collision detection

Crafty – How To Use



Crafty – How To Use

1. Include CraftyJS to HTML

```
//Option 1: CDN
<script type="text/javascript"
 src="https://rawgithub.com/craftyjs/Crafty/release/dist/cra
 fty-min.js"></script>
//Option 2: Download the library from CraftyJS.com and run
 it locally
<script type="text/javascript" src="crafty-min.js"></script>
```

2. Include CraftyJS to HTML

```
//Crafty.init([Number width, Number height, HTMLElement
 stage_element])
Crafty.init(500,350, document.getElementById('game'));
```

Crafty – How To Use

3. Create Crafty entity and add components to the entity

```
//Crafty.e(String componentList)
var myEntity = Crafty.e("2D, DOM, Color");
```

4. Call methods of the components

```
myEntity.text("Hello World !");
```

5. Add more components

```
myEntity.addComponent("Keyboard");
```

Crafty – Features

- Animations
  - Tweens with Easing functions ("linear", "smoothStep", "smootherStep", "easeInQuad", "easeOutQuad", and "easeInOutQuad")
  - Sprite Animation
- Collision Detection
  - Circular
  - Polygon

Crafty – Features

- Allows keyboard, mouse and touch input
- Allows physics:
  - Gravity
  - Velocity
  - Acceleration
- Allows particle creation through Canvas element

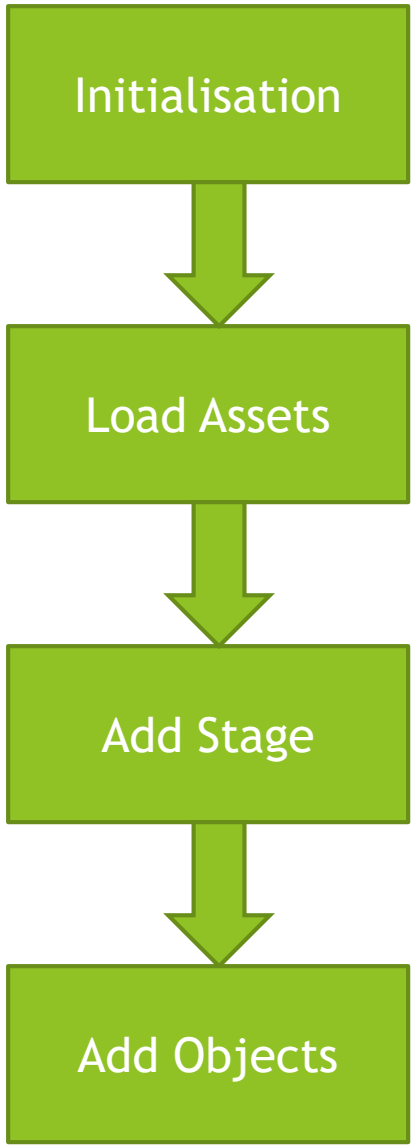
Crafty – Limitations

- Do not have native shape makers
- Collision detections are limited to circular and polygon
- Does not have camera settings
- Does not allow gamepad input
- Physics are limited
- Lack of samples
- Using Scenes rather than States

CreateJS

- CreateJS suite is sponsored mostly by Adobe and led by Grant Skinner
- CreateJS has structure inspired by Flash and it is intended to build rich HTML5 applications
- CreateJS suite includes
  - EaselJS: To manipulate HTML5 Canvas element
  - TweenJS: For tweenings and animations
  - SoundJS: To handle audio
  - PreloadJS: Manage assets loading
  - Zoë: tool for generating spritesheet images and frame data from SWF files.

CreateJS



CreateJS

- EaselJS CDN

  <script src="https://code.createjs.com/easeljs-0.8.1.min.js"></script>

- TweenJS CDN

  <script src="https://code.createjs.com/tweenjs-0.6.1.min.js"></script>

- AudioJS CDN

  <script src="https://code.createjs.com/soundjs-0.6.1.min.js"></script></head>

- PreloadJS CDN

  <script src="https://code.createjs.com/preloadjs-0.6.1.min.js"></script>

CreateJS - Features

- Has a preloader library
  - Create manifest of the resources (id, filename)
  - Create Loader to implement the manifest
  - Use Loader predefined functions as event handlers
    - onFileLoad
    - onComplete
    - handleComplete

CreateJS - Features

- Allows keyboard, mouse and touch input
- Allows drawing native shapes such as rectangle, circle, polygon
- Has a sprite sheet system that allows parameters such as:
  - Speed
  - Frequency
  - Chaining next animations, etc

CreateJS - Limitations

- Basic collision system
- No specific game physics system
- Does not have camera settings
- Does not allow gamepad input
- Do not support Filters and Particles
- Limited number of samples
- Using Stages rather than States

Impact

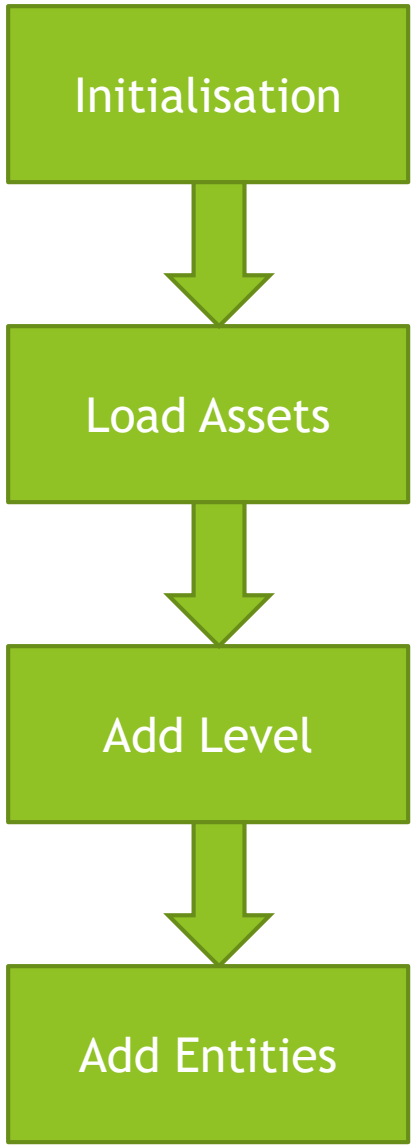
- Impact is a proprietary HTML5 JavaScript game engine developed by Dominic Szablewski
- Cost: $99 (engine, Weltmeister level editor, source code, demo)
- Runs on a web server such as XAMPP or WAMP
- Project created have the structure:
  - *Lib*: Contains the core logic and the code with go into the “game” subfolder inside this folder
  - *Media*: Graphics, tile sheets, audio and other media
  - *Tools*: Contains files to minify (bake) the project for publishing

Impact

- *Dev.html*: to run and test the unbaked game
- *Weltmeister.html*: Run this in the browser to launch the Weltmeister level editor
- *Index.html* and *game.min.js*: *game.min.js* gets generated when the project is baked. *Index.html* is identical to *dev.html* except inside the file it points to *game.min.js* instead of the unbaked proect

- Inside the lib/game folder, there are directories:
  - Entities: where the game codes are written
  - Levels: holds the level files saved by Weltmeister
  - Main.js: The boilerplate start-up code that ultimately launches the first level

Impact



Impact - Features

- Allows keyboard, mouse and touch input
- Entities should be coded in certain way to work with Weltmeister so the level design can be done easier
- Support animations with animationsheet
- Support background creation by using backgroundmap

Impact - Features

- Collision detection system:
  - ACTIVE: Will collide with ACTIVE and PASSIVE and both will be separated after collision
  - PASSIVE: No separation when PASSIVE collides with PASSIVE (the units may pass through each other)
  - FIXED: Nothing will move this entity
  - LITE: Collides with ACTIVE AND FIXED
  - NEVER: No collision

Impact - Features

- Weltmeister will scan the game/lib/entities/ folder for *.js files and automatically provide access to the the entities on the JS files for level creation
- A level consists of layers. Each layer has parameters:
  - Name: the name of the layer
  - Tileset: the image representing the tiles
  - Tilesize: Must be square
  - Dimensions: Dimension of the layer

Impact - Features

- Distance: Determines scroll speed. “1” will scroll the same speed as the entities, “2” will scroll slower, “3” will be eve slower
- Pre-render: can improve performance although this will not allow the tiles to be animated
- Repeat: will repeat the tiles as it scrolls by
- Is collision layer: states the layer is a collision layer and will start using collision tiles allowing to draw a collision map
- Link with collision: create tiles that automatically have a solid collision tiles behind it located on the collision layer

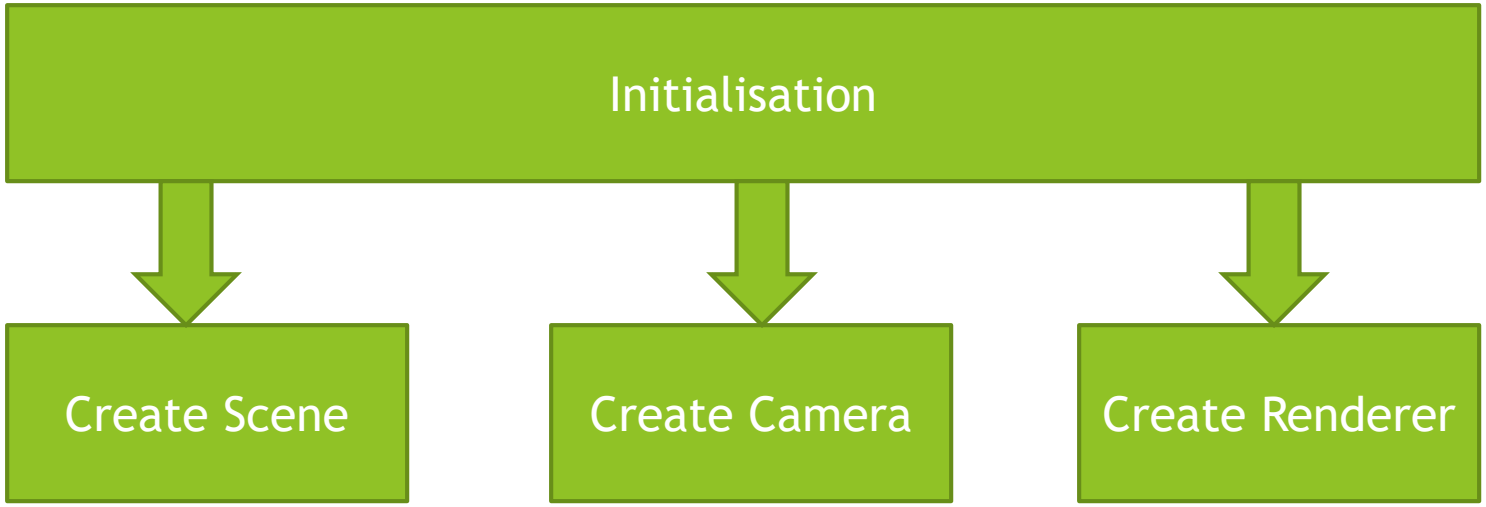
Impact - Limitations

- Basic collision system
- Game physics system is limited to Box2D
- Does not have camera settings
- Does not allow gamepad input
- Do not support Filters and Particles
- Limited number of samples
- Using Stages rather than States

ThreeJS

- ThreeJS is an open-source JavaScript WebGL libraries that contains intuitive set of objects that are commonly found in 3D graphics
- ThreeJS hides the low-level details of WebGL rendering
- ThreeJS contains prebuilt objects useful for developing games, animations, presentations, data visualization, modeling applications and post processing special effects
- ThreeJS includes extensive error checks and 3D maths such as matrices, projections and vectors
- ThreeJS is object-oriented and support text format files from 3D modelling packages (JSON)
- ThreeJS can also renders to 3D Canvas, SVG and CSS

ThreeJS


Initialisation
Create Scene
Create Camera
Create Renderer

ThreeJS

- There are 3 main elements in ThreeJS
  - The Scene: to set up what (objects, lights and cameras) and where is to be rendered by three.js
  - The Camera: a point of view in the scene
    - CubeCamera: Creates 6 cameras that render to a WebGLRenderTargetCube
    - OrthographicCamera: Camera with orthographic projection
    - PerspectiveCamera: Camera with perspective projection

ThreeJS

- The Renderer
  - CanvasRenderer: The Canvas renderer displays the scenes *not* using WebGL, but draws it using the (slower) Canvas 2D Context API
  - The WebGL renderer displays the scenes using WebGL, if your device supports it.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75,
    window.innerWidth / window.innerHeight, 0.1, 1000 );
var renderer = new THREE.WebGLRenderer();
```

ThreeJS

- ThreeJS has loader functions for various different type of files (JSONLoader, SVGLoader, TextureLoader, MaterialLoader, etc)
- ThreeJS has functions to control lights: AmbientLight, DirectionalLight, HemisphereLight, PointLight, SpotLight
- ThreeJS support shaders, materials and geometries

Typescript

- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
- TypeScript offers classes, modules, and interfaces to help you build robust components

- http://www.typescriptlang.org/Playground
- http://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript Detailed explanation about TypeScript
- https://www.typescriptlang.org/Playground Shows how object-oriented type in TypeScript are transcribed into native JavaScript (try to have a look at the Inheritance)

Phaser

- Phaser is one of the most popular JavaScript game framework
- Phaser renders using both WebGL and Canvas internally for performance efficiency
- Phaser has a loader function to make assets loading easier and the loader can also track the assets loading
- Phaser has 3 different physics systems for lightweight to complex game
- Phaser has animation and particle systems
- Phaser has camera system which can be manipulated and allows various inputs

Phaser – Download Library

phaser.io/download/stable

NEWS LEARN DOWNLOAD PHASER EXAMPLES COMMUNITY PLUGINS

Phaser CE 2.8.3

Phaser CE is the Community Edition. Read what this means and how it differs from v2.
The current version is 2.8.3 released on 24th July 2017.

clone | js | min.js | zip | tar.gz

You can also download this version from npm. Need more details? Follow this tutorial.

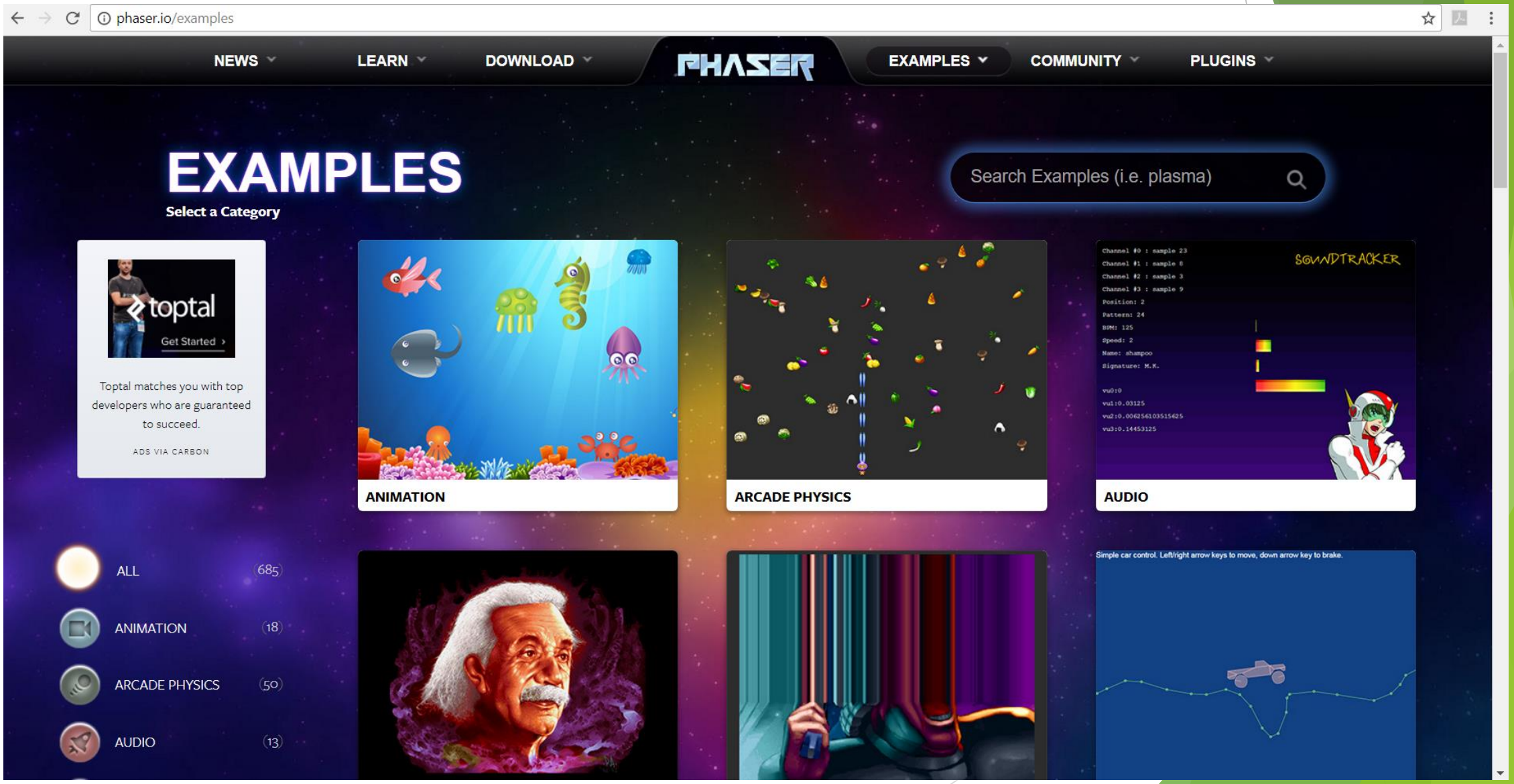
```
npm install phaser-ce@2.8.3
```

RELEASES

Stable

The previous version of Phaser was 2.8.2.
The last officially supported release of Phaser was 2.6.2.

Phaser - Samples



References

- http://craftyjs.com/
- http://www.createjs.com/
- http://impactjs.com/
- http://threejs.org/
- www.**typescript**lang.org/
- http://phaser.io/

- Nagle, D. (2014). HTML5 Game Engines: App Development and Distribution. *CRC Press, An A K Peters Book*
- Williams, J. L. (2012). Learning HTML5 Game Programming: A Hands-on Guide to Building Online Games Using Canvas, SVG and WebGL. *Addison-Wesley*
- Parisi, T. (2014). Programming 3D Applications with HTML5 and WebGL. *O'Reilly*