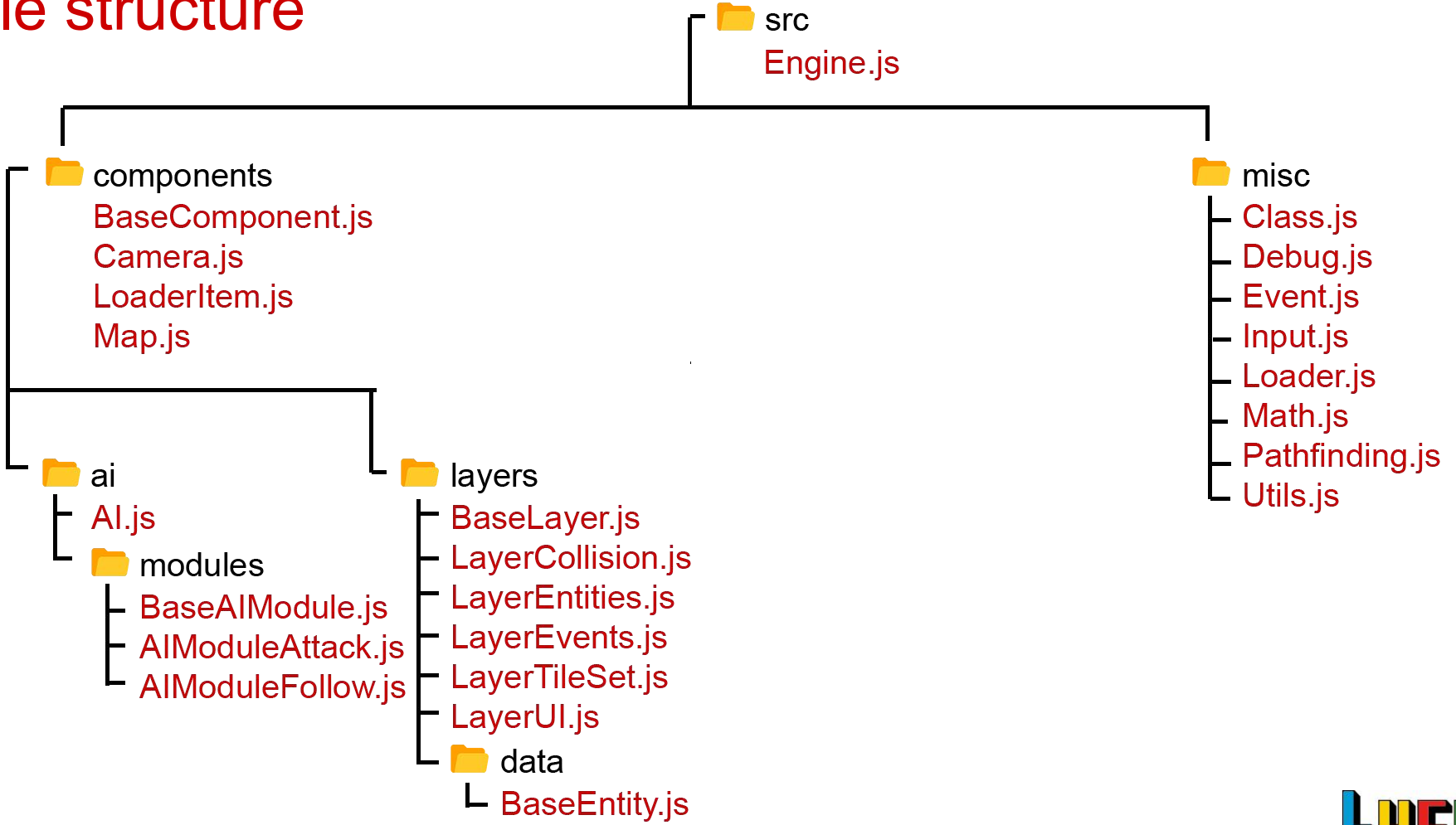
Game Engine

What is Lucid Engine?

- A 2D game engine, written in JavaScript.
- The aim of this project is, to delivers you with all the (core) tools you need, to create your own 2D game.
- The engine supports top-down (RPG style) games, as well as side-scroll (platformer) game types.
- Of course this engine supports both: Mobile devices and desktop computers.

Lucid

File structure



Presentation content

- Publish / Subscribe pattern
- Loading a Map
- AI
- AIModuleFollow
- Line-Of-Sight
- Pathfinding
- Collision
- Collision: Handle high speed scenarios
- Live demo

Publish / Subscribe pattern

Lucid.Event is a static Class and handles all the Event flows.

Lucid.Event.bind(“myEvent”, function(eventName, params…){ … }); // subscribe

Lucid.Event.unbind(“myEvent”); // un-subscribe

Lucid.Event.trigger(“myEvent”); // publish to all subscribers

Loading a Map

Maps contain Layers.
Layers can have different types (e.g. LayerEntities - for Entities).

Engine.loadMap(fileName) loads the Map and fires Event if ready.
You can then “build” the loaded Map:

- Add the Map Layers to Engine Layers-list.
- Engine renders all Layers from Layers-list.
- Layers delegate rendering. E.g. LayerEntities would delegate the rendering to all the Entities.

AI

Instance of AI gets attached to an Entity.
AI contains Modules.

Currently we feature:
- AIModuleFollow
- AIModuleAttack

AI collects mandatory (AI-related) data and passes it forward to each Module.
The Modules then process this data and use Events to notify possible subscribers.

Of course you can write your own AI Module(s) and process the data yourself, with your own logic.

AIModuleFollow

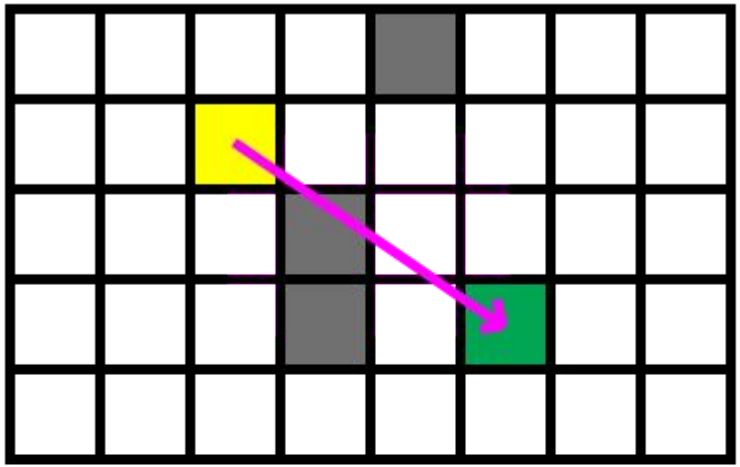
Entities have properties like “sightRadius”, “team”, “type” (e.g. unit, item).

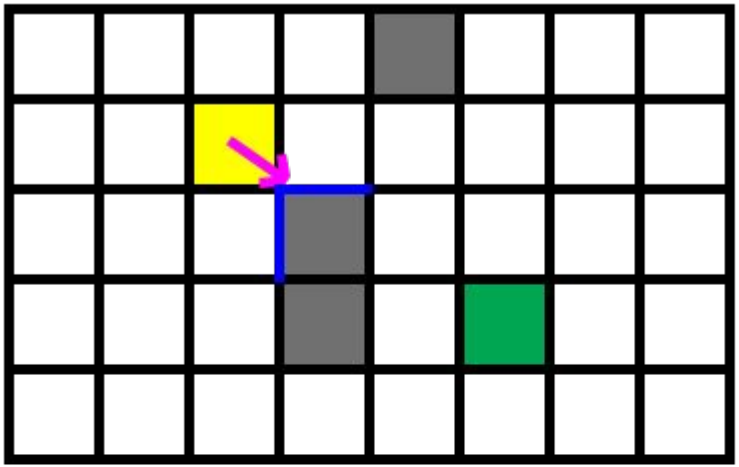
Example for following an enemy Entity:

```
1. for (check all other Entities inside “sightRadius”) {
2.     if (other Entity “team” differs AND other Entity “type” is unit) {
3.         if (other Entity is in Line-Of-Sight) {
4.             walk Path towards the other Entity
5.         }
6.     }
7. }
```

Line-Of-Sight

```
1. for (all Tiles in “sightRadius”) {
3.     for (all relevant sides of Tile) {
4.         if (line between Entity1 and Entity2 COLLIDES WITH side of Tile)
5.             lineOfSight = false;
6.     }
7. }
```

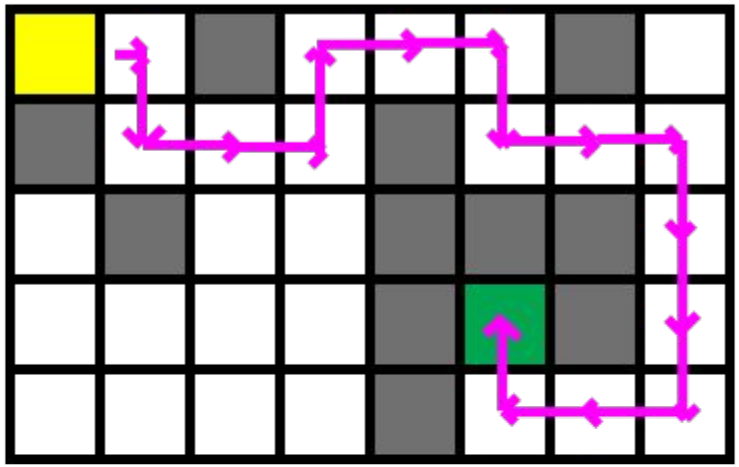




Lucid

Pathfinding

Grid based Pathfinding.
Using A* (A-Star) as algorithm to determine the fastest, possible Path.
- Internally using Heaps
- Supports diagonals
- Supports costs of tiles



Collision

Entity vs Entity
Entity vs World (Grid)

Basically, we only have Box VS Box scenarios.

Pitfalls:
- Never change x/y/z coordinates directly. Just give an “impuls” in a direction.
- **Handle high speed scenarios**

Collision: Handle high speed scenarios

```
1.  var updateSteps = 1 + max( abs(newX - lastX), abs(newY - lastY) );
2.  var updateStepX = (newX - lastX) / updateSteps;
3.  var updateStepY = (newY - lastY) / updateSteps;
4.
5.  newX = lastX;
6.  newY = lastY;
7.
8.  for (updateSteps) {
9.      if (HAS collided on X-axis AND Y-axis)
10.         break;
11.
12.     if (has NOT collided on X-axis yet)
13.         newX += updateStepX;
14.
15.     if (has NOT collided on Y-axis yet)
16.         newY += updateStepY;
17.
18.     use newX / newY to check collision vs other Entities
19.     use newX / newY to check collision vs World (Grid)
20. }
```

Live Demo