# Lovetoys

Lovetoys ist eine Library für Löve2D, welche sich aus drei Paketen zusammensetzt. Der grundlegende Part ist das Entity Component System, welches sich an dem Beispiel von Richard Lord orientiert. Für eine ausführliche Erklärung des Prinzips eines Entity Systems, lest euch diese externe Seite durch Richardlord.net. Ebenfalls enthalten sind ein Collision manager, welcher das Löve2D Collision Management erleichtert und eine Event Manager.

## Installation

Integriert die benötigten Dateien in euer Projekt und ändert die requires der jeweiligen Files. Wir empfehlen natürlich unser Bundle zu nehmen, da dieses bereits alles miteinander verknüpft und über den engine gesteuert wird.

## Entity Component System:

### Entity:

Die Entity ist das Objekt, welches tatsächlich vom Engine und den Systems verwaltet und bearbeitet wird. Dieses setzt sich aus verschiedenen Components zusammen welche in Entitiy.components abgesichert werden.

`Entity:addComponent(Component)`

Fügt eine Component hinzu. Nach dem hinzufügen einer Component, muss der Engine darüber informiert werden. Dies geschieht mit Hilfe von Engine:componentAdded() (Siehe engine)

`Entity:removeComponent(name)`

Entfernt die Component, welche dem Namen entspricht. Falls die Component nicht vorhanden wird, wird kein Fehler geworfen. Nach dem entfernen einer Component, muss der Engine darüber informiert werden. Dies geschieht mit Hilfe von Engine:componentAdded() (Siehe engine)

`Entity:getComponent(name)`

Gibt die jeweilige Component zurück. Ist equivalent zu Entity.components[name]

### Component:

Components selbst sind nichts weiteres als Objekte, die einige Daten beinhalten, wie z.B. Größe, Scaling und Adresse eines Bildes. Daher gibt es keinen bestimmten Aufbau einer Component, sondern lediglich eine Superclass, von der abgeleitet wird.

### System:

Nun kommen wir zu den Dienstleistern unseres Systems. Der Engine weist dem System die für das System kompatiblen Entities zu, welche das System anschließend bearbeitet. Die Entities werden in

System.targets mit dem key entity.id (Siehe Engine) gesichert.

`System:getRequiredComponents() return {"Componentname1", "Componentname2"} end`

:getRequiredComponents() bestimmt, welche Components eine Entity besitzen muß, damit sie vom Engine dem System als Target zugewiesen wird. Die zurückgegebene Tabelle mit Componentnames muss beim Schreiben des Systems angelegt werden. Die Funktion wird also beim Erstellen der Subclass überschrieben.

`System:update(dt)`

Diese Funktion ist standartmäßig vorhanden, jedoch können auch andere Funktionen wie :draw() oder :fireEvent() in die Systems eingebaut werden.

`System:addEntity(entity)`

Fügt Entity zu den Targets hinzu. Diese Funktion wird normal nie benutzt, da der Engine die Entitys verwaltet

`System:removeEntity(entity)`

Entfernt Entity von den Targets. Diese Funktion wird normal nie benutzt, da der Engine die Entitys verwaltet

## Engine:

Das Herzstück des Entity Component Systems. Der Engine selbst verwaltet die Entities und Systems, legt Entitylists an und ruft die jeweiligen Funktionen der Systeme aus.

`Engine.entities`

Beinhaltet alle Entities.

`Engine.allSystems`

Beinhaltet alle Systeme

`Engine.logicSystems/.drawSystems`

Manuell angelegte Systemaufteilung, sodass zwischen :update und :draw unterschieden werden kann.

`Engine.stack`

Ein wichtiger Bestandteil. Hier werden die Keys der gelöschten Entities gesichert und anschließend neuen Entities zugewiesen. Dies verhindert eine ewig anwachsenden Keygröße. (siehe Engine:addEntity)

`Engine.requirements`

Alle Components, welche von einem System required werden, werden hier als subtable, z.B. Engine.requirements.Componentname, angelegt. Jedes System, welches diese Component benötigt, trägt sich in diese Liste ein, sodass falls eine Component hinzugefügt wird, nur die eingetragenen Systems kontrolliert werden.

`Engine:addSystem(system, type)`

Fügt ein System hinzu. Je nach Type kann eine Unterteilung in verschiedene Systemtypen vorgenommen werden. Dies muss jedoch hardcoded im Engine vonstattengehen.

`Engine:addEntity(entity)`

Sucht die Systems, welche die Componentkonstellation der Entitiy als ausreichend betrachten und fügt die Entity diesen hinzu.

`Engine:removeEntity(entity)`

Entfernt die jeweilige Entity und trägt sie aus Systems und Engine aus.

`Engine:componentAdded(entity, added)`

Ueberprüft, ob die Entity durch die hinzugefügte Component die Requirements eines bestehendes Systems erfüllt. Es wird eine entity und eine Liste mit Components übergeben, welche hinzugefügt wurden. z.B. Engine:componentAdded(entitiy, {"Componentname1", "Componentname2"})

`Engine:componentRemoved(entity, removed)`

Entfernt die Entity aus allen Systems, die diese Component benötigten. Es wird eine entity und eine Liste mit Components übergeben, welche entfernt wurden. z.B. Engine:componentAdded(entitiy, {"Componentname1", "Componentname2"})

`Engine:getEntitylist(component)`

Gibt eine Liste mit allen Entities zurück, welche diese Component beinhalten. Als Parameter wird immer der Name der Klasse übergeben.