Tartalomjegyzék

[Bevezetés 1](#_Toc72478250)

[Entity-Component-System (ECS) 1](#_Toc72478251)

[Bevezetés az ECS-be 1](#_Toc72478252)

[Interfész 2](#_Toc72478253)

[Adatstruktúrák: 3](#_Toc72478254)

[Függvények 3](#_Toc72478255)

[Implementálás 4](#_Toc72478256)

[ECS\_EntityStore 5](#_Toc72478257)

[Tesztelés 6](#_Toc72478258)

# Bevezetés

# Entity-Component-System (ECS)

## Bevezetés az ECS-be

Az ECS egy programozási minta, amit többnyire játékprogramoknál alkalmaznak, viszont egyéb szoftverekben is használható, ahol az ECS nyújtotta előnyöket ki lehet használni.

Az ECS egy Data-Oriented dizájn, ami kompozíciót alkalmaz az objektum-orientált öröklődés helyett. Ez nagy rugalmasságot enged meg.

De hogy mit értünk kompozíció alatt? A kódban definiálhatunk több, különböző komponenst, és ezekből egyet, vagy többet hozzá rendelhetünk az entitásokhoz. Fontos megjegyezni, hogy a komponensek csupán adatok, viselkedést majd az ECS System része fogja szolgáltatni.  
Egy példa: Tegyük fel, hogy a játékunkban van a játékos, virágok és fák. Mind a hármat ki akarjuk rajzolni, azonban csak a fával akarunk tudni a játékossal összeütközni. Ekkor a fának, illetve a virágoknak is lesz egy grafikus-komponense, ami a kirajzolásukhoz szükséges adatot tárolja, viszont hitbox-komponenssel csak a fa, illetve a játékos fog rendelkezni.

Azt látjuk, hogy ez a megközelítés rugalmas, hiszen egy entitást akármilyen komponensekből össze tudunk rakni, viszont, hogy ez hogyan garantálja a gyorsaságot. És ez a jó cache kihasználtságából ered. Az entitásokhoz tartozó komponensek adataid az ECS memóriában egymás után eltárolja.

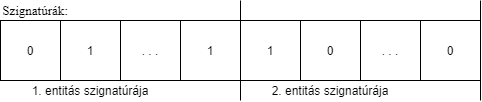
Szemléltető ábra:

A képen asztal látható

Automatikusan generált leírás  
Az ábrán látszik, hogy egy entitás n darab komponensből állhat össze. Ez implementációtól függ, de jelen esetben minden entitásnak lefoglalunk az összes komponensnek elegendő helyet, legfeljebb nem fog mindegyikből összeállni az adott entitásunk.

Egy entitást teljes mértékben csak a komponensei határozzák meg, és így egy entitásra tekinthetünk úgy, mint egy egyszerű index, vagy azonosító, amin keresztül el tudjuk érni az adott entitás komponenseit.

Ezen túl érdemes eltárolni egy szignatúra-tömböt, ebbe speciális számokat tárolunk, amikkel meg lehet határozni, hogy egy adott entitás mely komponenseket használja. Ha ezeket a speciális szignatúrákat binárisan reprezentáljuk, akkor a 0 bit azt jelenti, hogy az adott komponenst nem használja, ha 1-es a bit, akkor pedig használja.

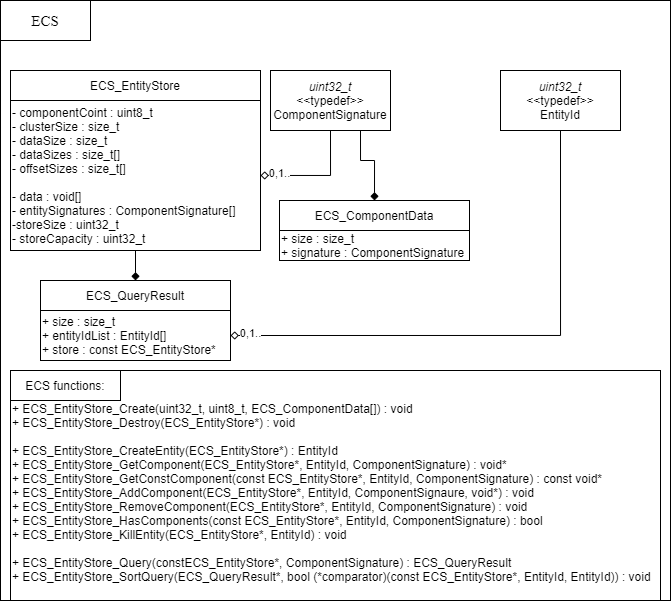


Az ECS-be olyan elemek fognak kerülni, amiken nagyon sűrűn végig akarunk iterálni. Így, mivel az entitások komponensei a cache-be kerülnek, sokkal gyorsabb lesz az elérésük, mint ha memóriában szétszórva lennének.

## Interfész

Ebben a részben az általam megvalósított Entity-Component-System interfészét mutatom be.

Modulról ábra:



### A képen asztal látható Automatikusan generált leírásAdatstruktúrák:

#### ComponentSignature

Minden komponenshez kell, hogy tartozzon egy egyedi pozitív egész szám. Ezzel tudjuk az adott komponenst azonosítani. Ezen túl ennek a számnak kettő hatványnak kell lennie, így a bináris alakjában csak egy darab egyes lesz, a többi bináris helyiérték nulla lesz. Ezáltal az adott komponenst valójában csak egy bit azonosít. Kettő különböző komponens szignatúráját, ha binárisan össze vagyoljuk, akkor az az érték mind a kettő komponensre hivatkozik.

#### EntityId

Ez egy pozitív egész szám, amin keresztül el tudunk érni egy entitást az Entity-Component-Systemből. Fontos megjegyezni, hogy ha törlünk az ECS-ből egy entitást, akkor ezek az azonosítók már más elemre mutathatnak, így nem érdemes ezeket elmenteni hosszú távra.

#### ECS\_EntityStore

Rajta keresztül érhetjük el az Entity-Component-Systemet. Többet róla az implementáció részben.

#### ECS\_QueryResult

Amikor az ECS-en szűrni szeretnék valamilyen szempont szerint, akkor egy ilyen ECS\_QueryResult típusú változót kapunk vissza.

size: Megmondja, hogy hány elemre teljesült a szűrés.

entityIdList: Ebből tudjuk, hogy mely EntityId-jű entitásokra teljesül a szűrt feltétel.

store: Arra az ECS\_EntityStore-ra egy pointer, amire a szűrést végrehajtottuk.

#### ECS\_ComponentData

Erre az ECS\_EntityStore\_Create (…) függvényben fogjuk használni. Ezen keresztül meg kell mondani, hogy az adott komponens, amit regisztrálni akarunk az ECS-be, annak mekkora a mérete (ezt a sizeof operátorral célszerű), illetve meg kell adni neki egy ComponentSignature-t, amin keresztül hivatkozhatunk rá.

#### ECS\_EntityStore\_Comparator

Egy függvény-mutató, amit az ECS\_QueryResult rendezésénél fogunk használni. Ezzel határozhatjuk meg, hogy két, azonos ECS\_EntityStore-beli EntityId-vel rendelkező entitást hogyan hasonlítsunk össze.

### Függvények

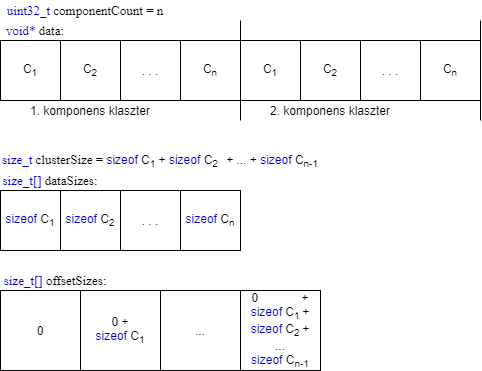
|  |  |
| --- | --- |
| Definíció | Leírás |
| ECS\_EntityStore\* ECS\_EntityStore\_Create  (uint32\_t, uint8\_t, ECS\_ComponentData…) | Visszaad egy megadott kapacitású ECS-t. A paraméterül megadott komponenseket lehet az Entitásokhoz kötni, illetve a megadott ComponentSignature-rel lehet rá majd hivatkozni. |
| void  ECS\_EntityStore\_Destroy  (ECS\_EntityStore\*) | Felszabadítja az adott ECS-t. |
| EntityId  ECS\_EntityStore\_CreateEntity  (ECS\_EntityStore\*) | Létrehoz egy új entitást az ECS-ben, és visszaad egy Id.-t, amivel elérhetjük azt. Ha a maximum kapacitáson túl akarunk új entitást létrehozni, akkor az ECS mérete nő. |
| void\*  ECS\_EntityStore\_GetComponent  (ECS\_EntityStore\*, EntityId, ComponentSignature) | Visszaadja a megadott entitáshoz tartozó kért komponenst. |
| const void\*  ECS\_EntityStore\_GetConstComponent  (const ECS\_EntityStore\*, EntityId, ComponentSignature) | Visszaadja a megadott entitáshoz tartozó kért komponenst konstansként. |
| void  ECS\_EntityStore\_AddComponent  (ECS\_EntityStore\*, EntityId, ComponentSignature, void\*) | A kért entitáshoz hozzárendel egy komponenst megadott adattal, amit lemásolunk. |
| void  ECS\_EntityStore\_RemoveComponent  (ECS\_EntityStore, EntityId, ComponentSignature) | A kért entitásnak törli egy komponensét. |
| bool ECS\_EntityStore\_HasComponents  (const ECS\_EntityStore\*, EntityId, ComponentSignature) | Megmondja, hogy egy entitásnak van-e megadott szignatúrájú komponense/komponensei, utóbbi esetben bináris vagy-gyal lehet őket összekötni. |
| void  ECS\_EntityStore\_KillEntity  (ECS\_EntityStore\*, EntityId) | Törli az ECS-ből a kért entitást.  Ennek a műveletnek a hatásául módosulhatnak korábban elmentett EntityId-k, így azt nem tanácsos csinálni. |
| ECS\_QueryResult\*  ECS\_EntityStore\_Query  (const ECS\_EntityStore\*, ComponentSignature) | A kért szignatúrájú entitások Id.-it tartalmazó ECS\_QueryResult-ot adja vissza. A ECS\_EntityStore\_HasComponents-el hasonlóan lehet a paraméterben bináris vagy-ot használni. |
| void  ECS\_EntityStore\_SortQuery  (ECS\_QueryResult\*, ECS\_EntityStore\_Comparator) | Az kért ECS\_EntityStore\_Query-t rendezi a megadott logika szerint. |
| void  ECS\_QueryResult\_Destroy  (ECS\_QueryResult\*) | Felszabadítja a megadott ECS\_EntityStore\_Query-t |

## Implementálás

Ebben a részben ismertetem, hogy én hogyan valósítottam meg az Entity-Component-Systemet, azaz, hogy az interfészben ismertetett függvények hogyan működnek. Szó lesz az ECS\_EntityStore belső működéséről, hogy hogyan működik, illetve a használt statikus függvényeket is ismertetni fogom.

### ECS\_EntityStore

Mindenekelőtt a void\*-t ismertetem, mert ez nem feltétlenül ismert mindenki számára. A többi típusos mutatóval ellentétben, …



# Tesztelés

## Unit tesztek

A unit teszteket c++-ban mert kényelmes framework, a doctest keretrendszer segítségével készítettem el, a következő modulokhoz: Camera, Collision, DataStructures, ECS, Pathfinding, World.

Camera tesztjeiben vannak előre definiálva különböző kamerák, amik segítségével történnek különböző konverziók, illetve egy fix képernyő szélességgel, illetve magassággal dolgozik.

Collision tesztjeiben a különböző geometriai alakokkal történő ütközések, detektálását, vagy ha nincs ütközés, akkor annak hiányát ellenőrizzük. Itt a CollisionDetection függvényeinek definiáljuk a helyes működést.

A DataStructures tesztjében a jelenleg egyetlen adatstruktúránkra írtam teszteket, a rendezetlen tömbre. Szükséges tesztelni a helyes inicializálást, elem hozzáadást, elem levételt.

Az ECS-re is készültek tesztek. Tesztelésre került az inicializálás, egyszerű komponensekkel, új entitás hozzáadása. Entitásokhoz tartozó komponens lekérése, hozzáadása, törlése, létezésének lekérése. Ezentúl a lekérdezések (Query), illetve annak rendezése.

A Pathfinding tesztjeiben szükséges egy mock ECS olyan entitásokkal, amikre a CollisionDetection alkalmazható. Ezek után lehet tesztelni az A\* algoritmus helyességét.

A World tesztjében is szükséges egy mock ECS. Itt alapvetően az EntityActions függvényeit teszteljük.