

# Progetto di Applicazioni e Giochi Multiplayer su Architettura Unity DOTS

---

CANDIDATO:

MICHELE RIGHI

RELATORE:

DOTT. PAOLO BELLAVISTA

CORRELATORE:

DOTT. ANDREA GARBUGLI



# Introduzione

---

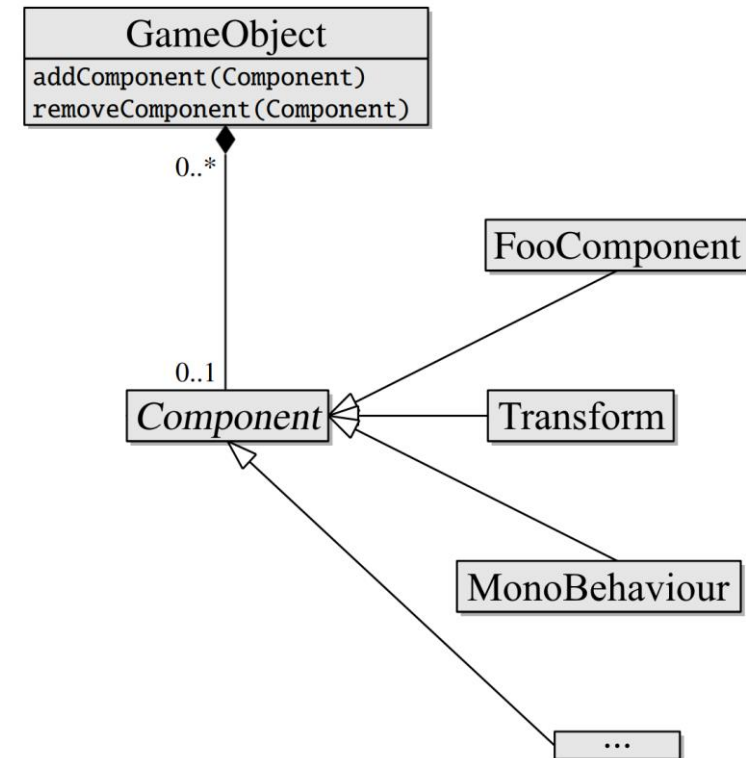
# Introduzione

---

- Nel 2018 Unity ha iniziato una profonda **ristrutturazione** del proprio motore di gioco.

# Introduzione

- Nel 2018 Unity ha iniziato una profonda **ristrutturazione** del proprio motore di gioco.
- **Architettura classica** limitata dal modello a **componenti** (GameObject e MonoBehaviour).



Unity: modello a componenti.

# Introduzione

- Nel 2018 Unity ha iniziato una profonda **ristrutturazione** del proprio motore di gioco.
- **Architettura classica** limitata dal modello a **componenti** (GameObject e MonoBehaviour).
  - **Overhead** delle classi (sia GameObject che MonoBehaviour sono classi).

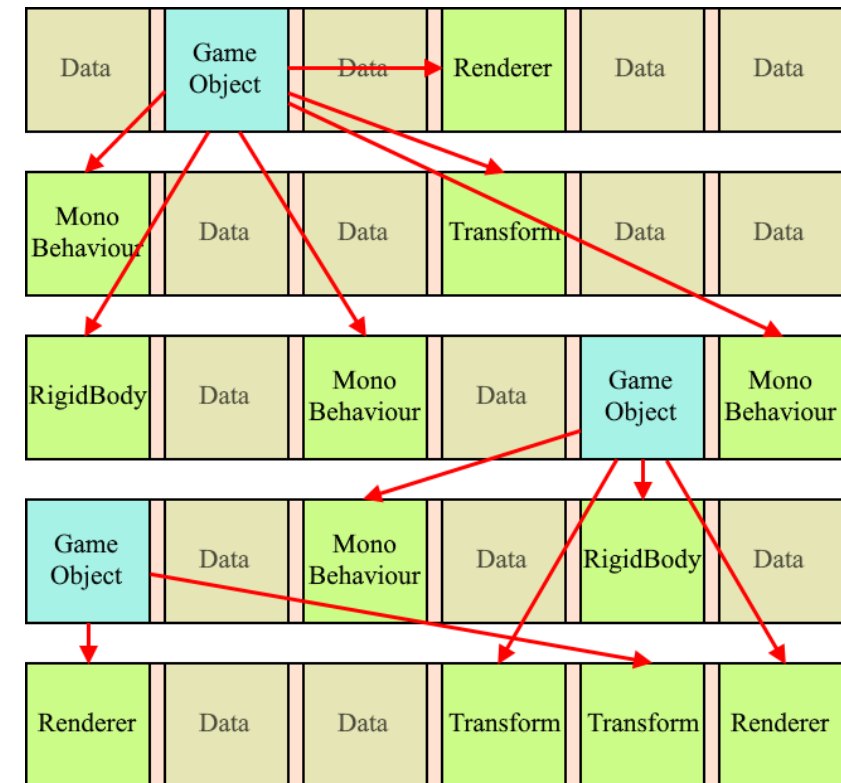
```
namespace UnityEngine
{
    ... public class Transform : Component, IEnumerable
    {
        protected Transform();

        ... public Vector3 localPosition { get; set; }
        ... public Vector3 eulerAngles { get; set; }
        ... public Vector3 localEulerAngles { get; set; }
        ... public Vector3 right { get; set; }
        ... public Vector3 up { get; set; }
        ... public Vector3 forward { get; set; }
        ... public Quaternion rotation { get; set; }
        ... public Vector3 position { get; set; }
        ... public Quaternion localRotation { get; set; }
        ... public Transform parent { get; set; }
        ... public Matrix4x4 worldToLocalMatrix { get; }
        ... public Matrix4x4 localToWorldMatrix { get; }
        ... public Transform root { get; }
        ... public int childCount { get; }
        ... public Vector3 lossyScale { get; }
        ... public bool hasChanged { get; set; }
        ... public Vector3 localScale { get; set; }
        ... public int hierarchyCapacity { get; set; }
        ... public int hierarchyCount { get; }
    }
}
```

Unity: componente Transform.

# Introduzione

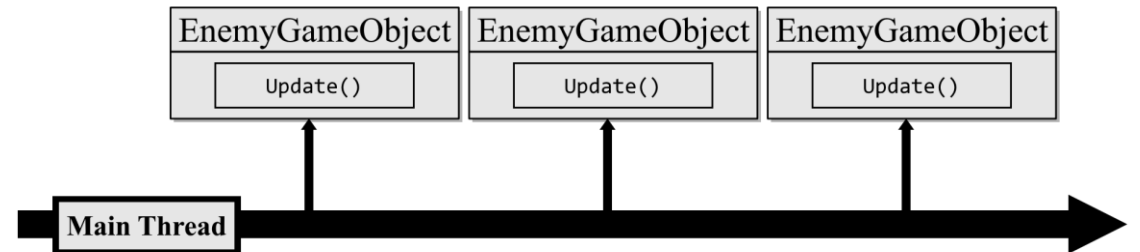
- Nel 2018 Unity ha iniziato una profonda **ristrutturazione** del proprio motore di gioco.
- **Architettura classica** limitata dal modello a **componenti** (GameObject e MonoBehaviour).
  - **Overhead** delle classi (sia GameObject che MonoBehaviour sono classi).
  - **Dati sparpagliati** in memoria, a causa dei riferimenti.



Dati sparpagliati in memoria.

# Introduzione

- Nel 2018 Unity ha iniziato una profonda **ristrutturazione** del proprio motore di gioco.
- **Architettura classica** limitata dal modello a **componenti** (GameObject e MonoBehaviour).
  - **Overhead** delle classi (sia GameObject che MonoBehaviour sono classi).
  - **Dati sparpagliati** in memoria, a causa dei riferimenti.
  - **Core multipli** della CPU **non utilizzati**.

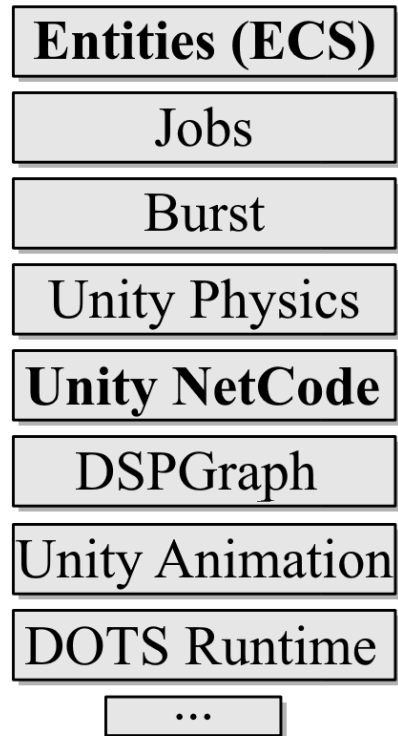


Unity: esecuzione dei MonoBehaviour.

# Introduzione

---

- Nel 2018 Unity ha iniziato una profonda **ristrutturazione** del proprio motore di gioco.
- **Architettura classica** limitata dal modello a **componenti** (GameObject e MonoBehaviour).
  - **Overhead** delle classi (sia GameObject che MonoBehaviour sono classi).
  - **Dati sparpagliati** in memoria, a causa dei riferimenti.
  - **Core multipli** della CPU **non utilizzati**.
- Soluzione: Unity **Data-Oriented Technology Stack** (DOTS).



Unity: librerie DOTS.



# Argomenti principali

---



Entity Component  
System (ECS)



NetCode



Prototipi



Risultati  
Sperimentali



Conclusioni

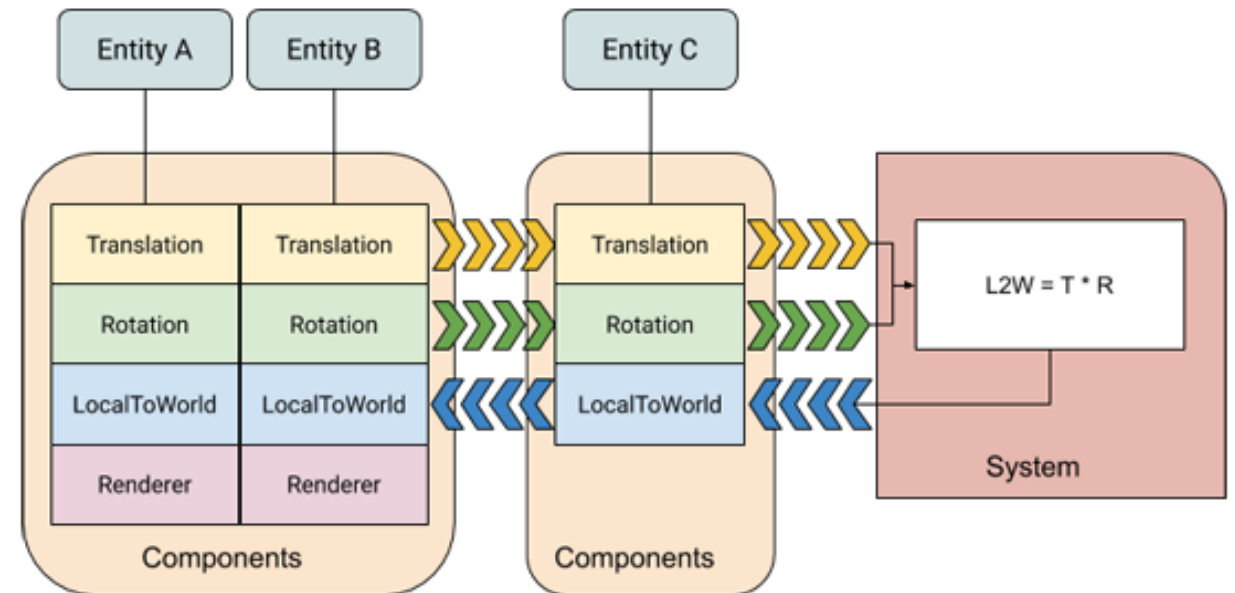
# Entity Component System (ECS)



- Entities.

- Components.

- Systems.

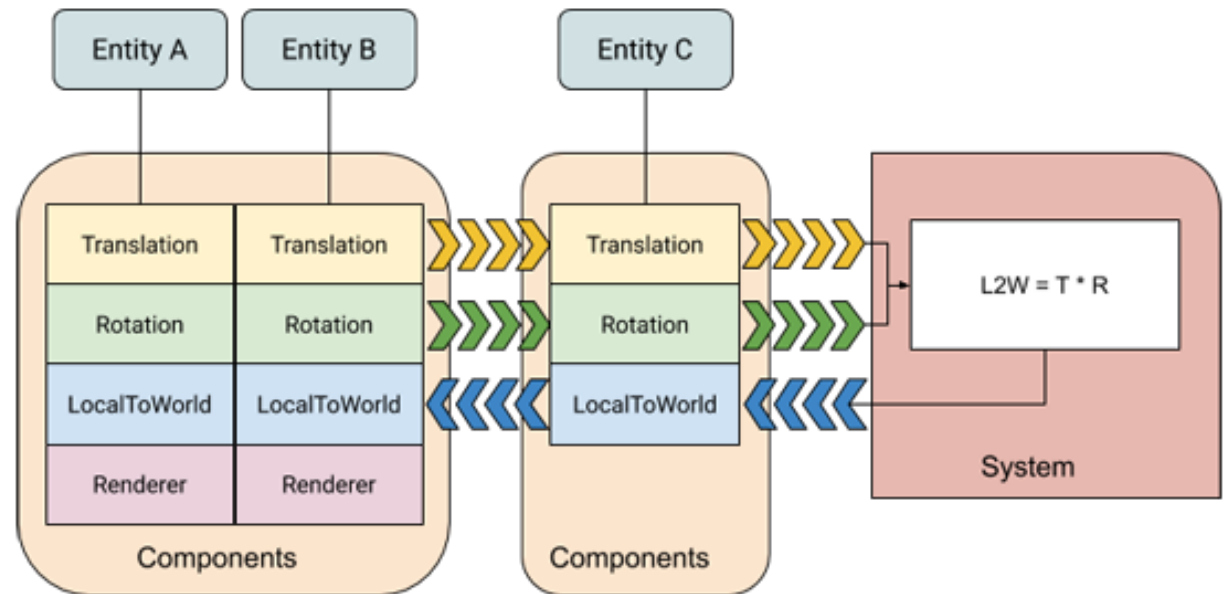


Esempio funzionamento ECS.

# Entity Component System (ECS)



- **Entities.** Le «**cose**» concrete che popolano il gioco a tempo di esecuzione. Sono paragonabili alle **chiavi** (ID numerici) di un database.
- **Components.**
- **Systems.**

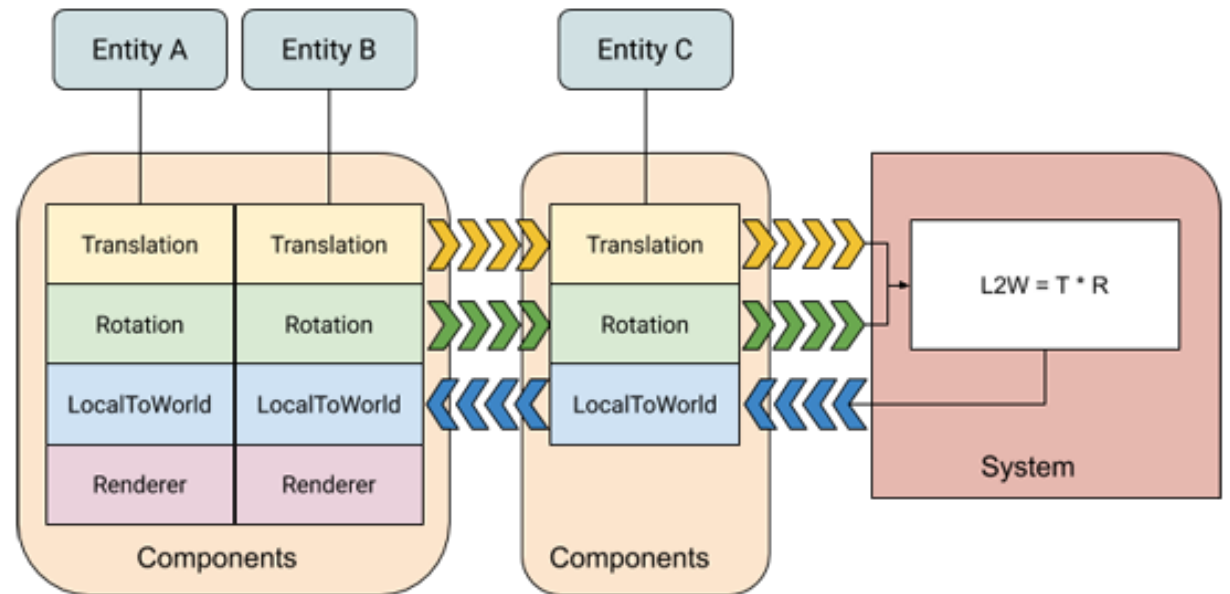


Esempio funzionamento ECS.

# Entity Component System (ECS)



- **Entities.** Le «**cose**» concrete che popolano il gioco a tempo di esecuzione. Sono paragonabili alle **chiavi** (ID numerici) di un database.
- **Components.** I **dati** associati alle entità. Immagazzinano lo stato ma non contengono alcun tipo di logica. Sono paragonabili alle **tuple** di un database.
- **Systems.**

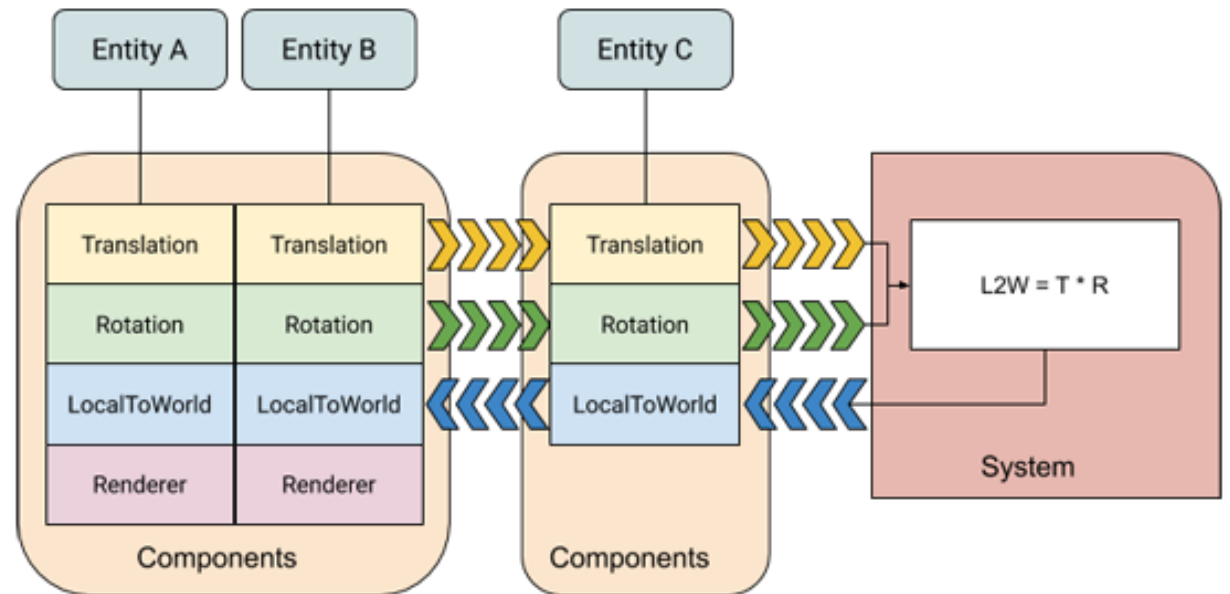


Esempio funzionamento ECS.

# Entity Component System (ECS)



- **Entities.** Le «**cose**» concrete che popolano il gioco a tempo di esecuzione. Sono paragonabili alle **chiavi** (ID numerici) di un database.
- **Components.** I **dati** associati alle entità. Immagazzinano lo stato ma non contengono alcun tipo di logica. Sono paragonabili alle **tuple** di un database.
- **Systems.** Permettono di realizzare il **comportamento**, modificando lo stato dei componenti. Sono paragonabili alle **query** di una database.



Esempio funzionamento ECS.

# NetCode

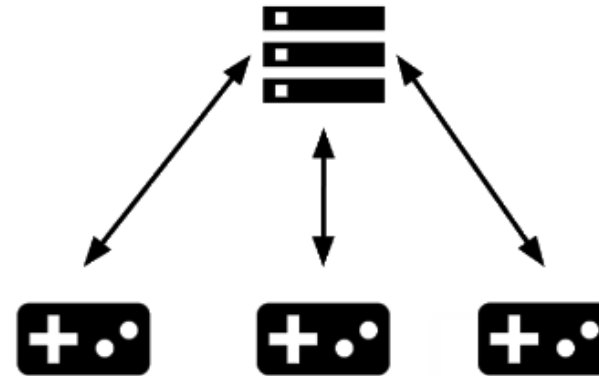
---



# NetCode



- Topologia di rete basata su un modello a client/server con **server autoritativo**.

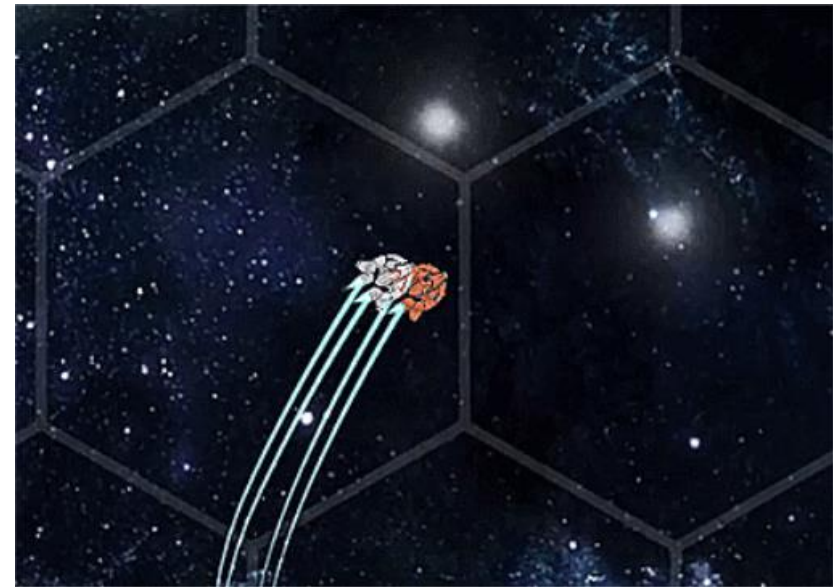


Modello a server autoritativo.

# NetCode



- Topologia di rete basata su un modello a client/server con **server autoritativo**.
- Riduzione della latenza tramite l'utilizzo della **predizione lato client**.



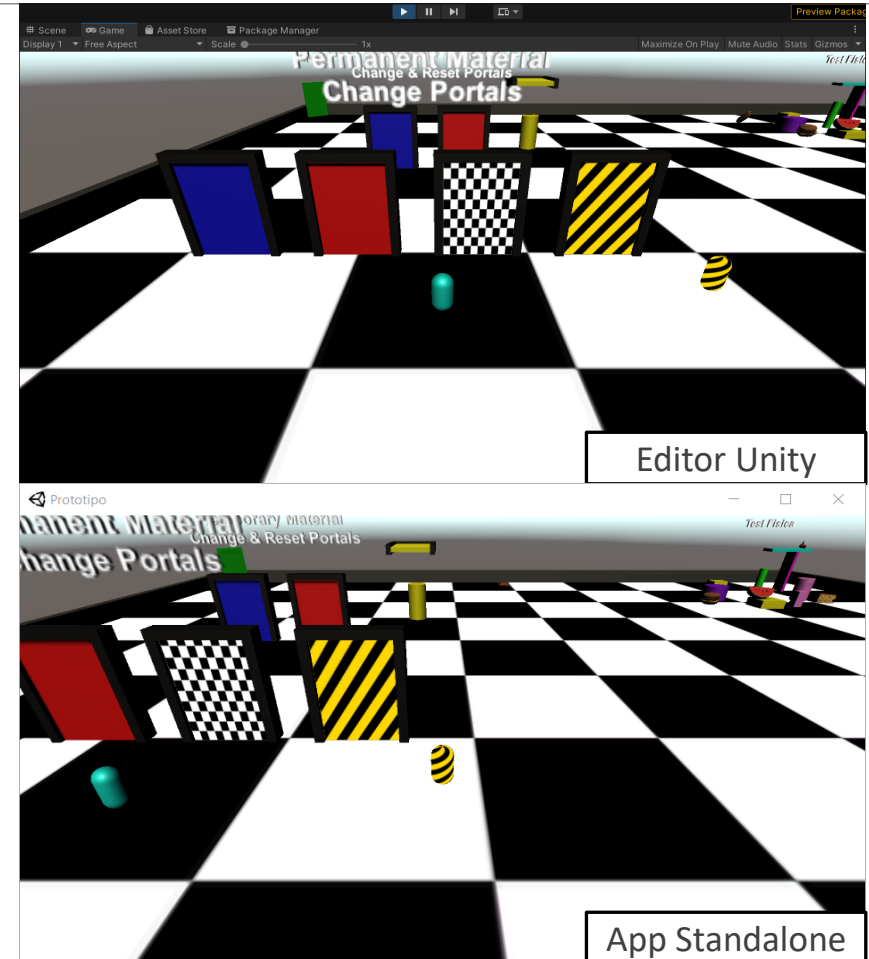
Esempio predizione lato client  
(nave bianca).



# Prototipi



## PROTOTIPO 1: VIDEOGIOCO COMPLETO

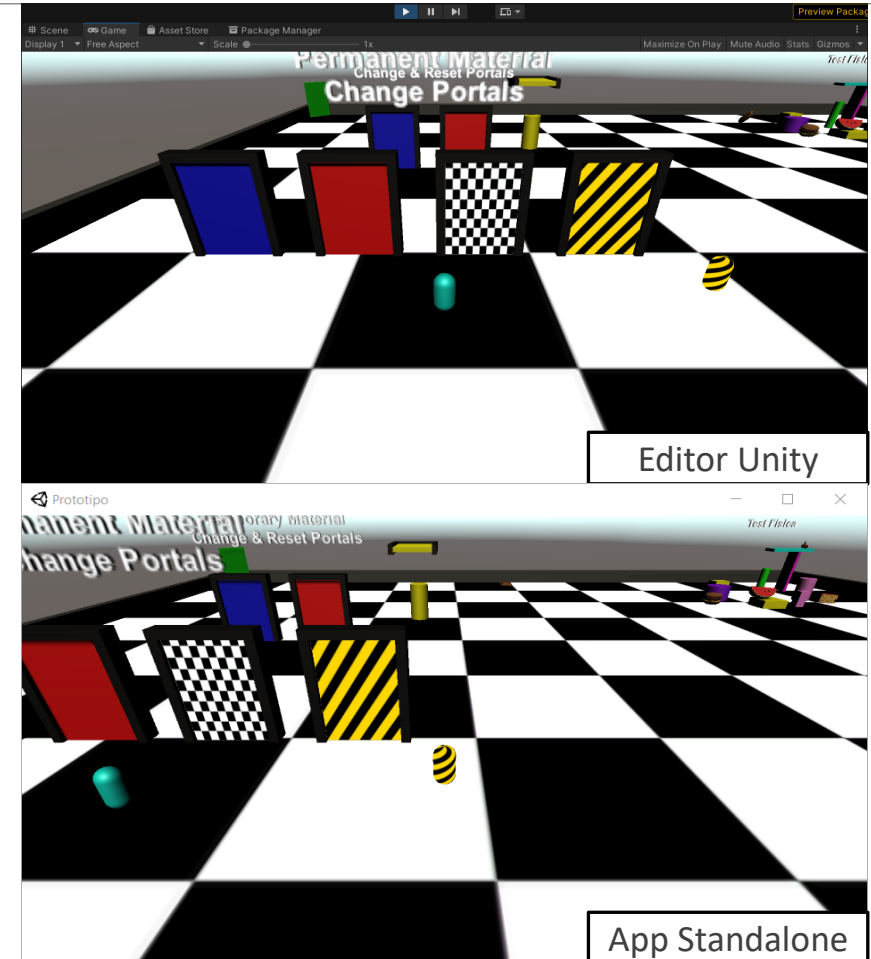


# Prototipi



## PROTOTIPO 1: VIDEOGIOCO COMPLETO

- Applicazione basata su architettura **ECS**, fornita dal package Entities.

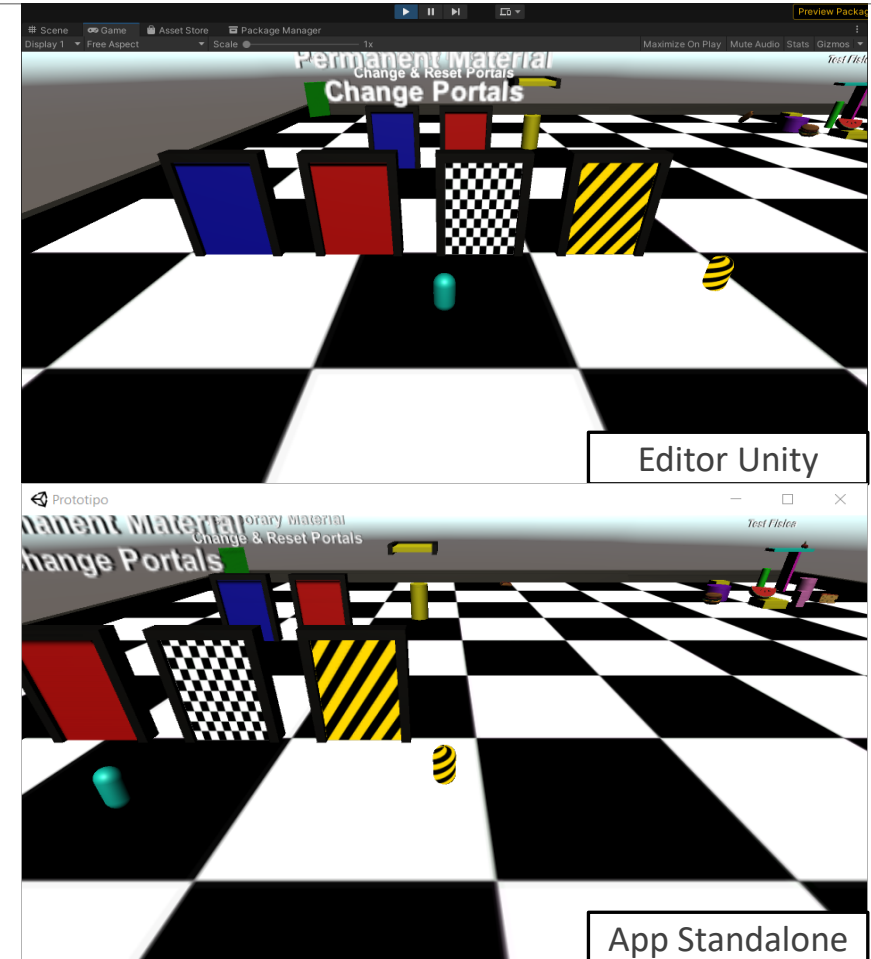


# Prototipi



## PROTOTIPO 1: VIDEOGIOCO COMPLETO

- Applicazione basata su architettura **ECS**, fornita dal package Entities.
- Networking implementato tramite l'utilizzo del package **NetCode**.

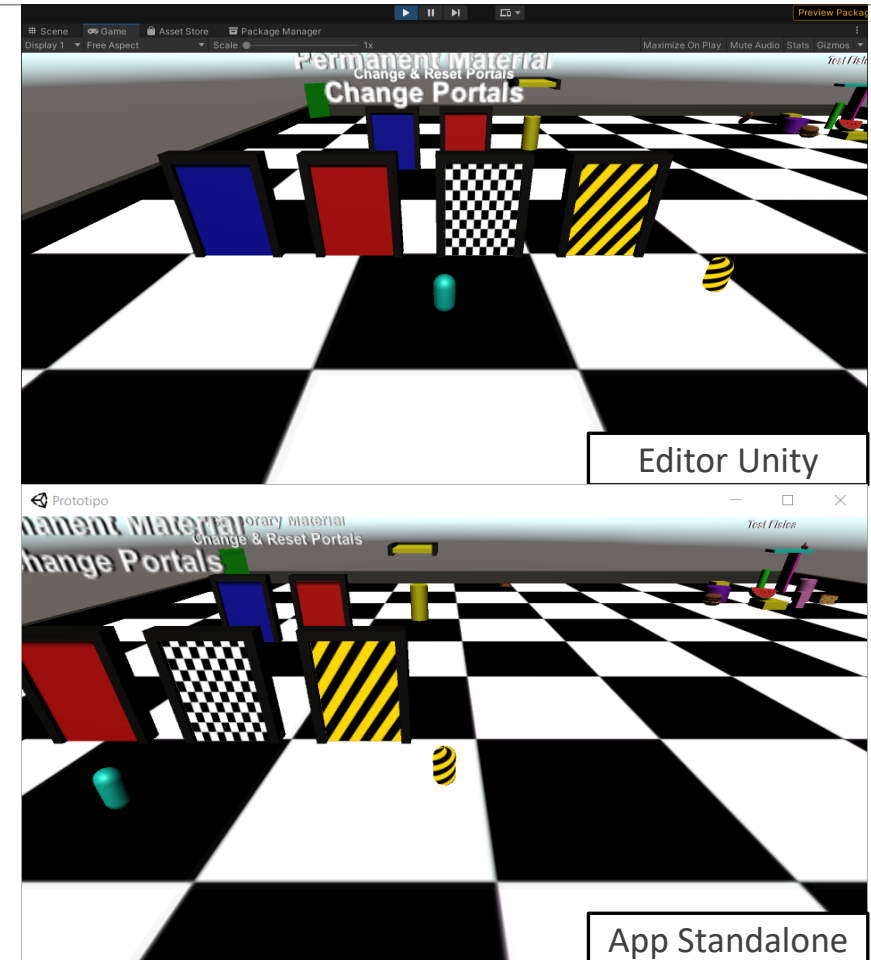


# Prototipi



## PROTOTIPO 1: VIDEOGIOCO COMPLETO

- Applicazione basata su architettura **ECS**, fornita dal package Entities.
- Networking implementato tramite l'utilizzo del package **NetCode**.
- Simulazione fisica realizzata utilizzando il package **Physics**.

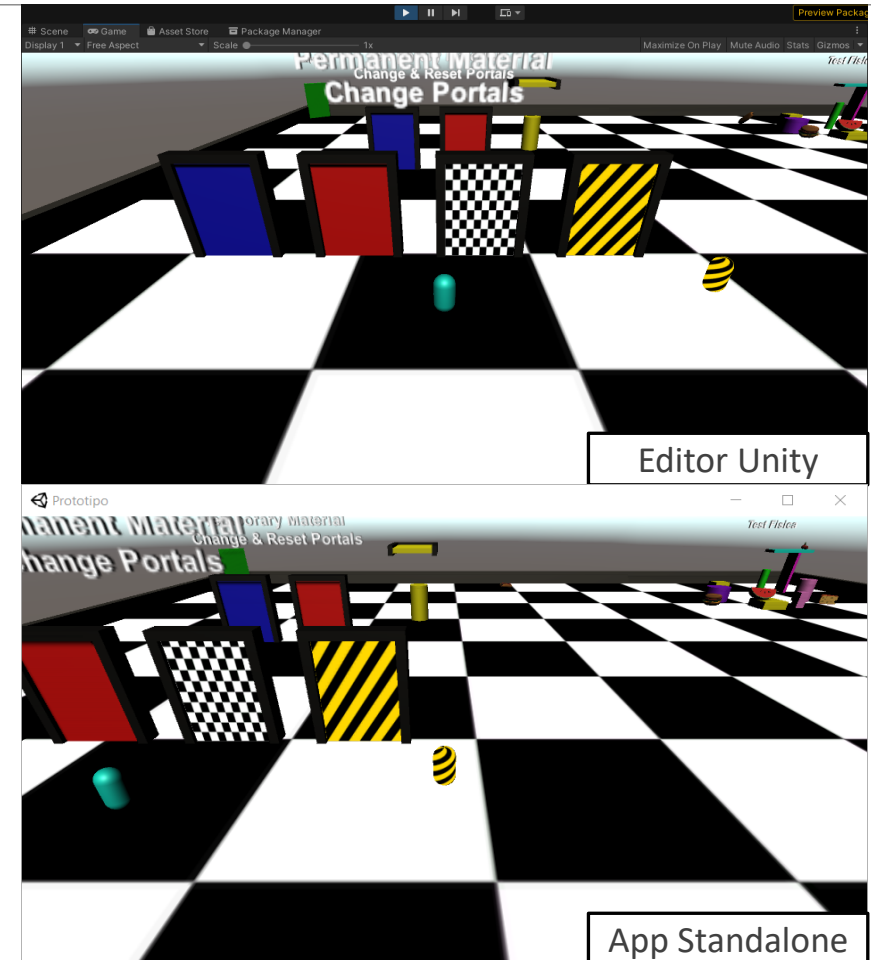


# Prototipi



## PROTOTIPO 1: VIDEOGIOCO COMPLETO

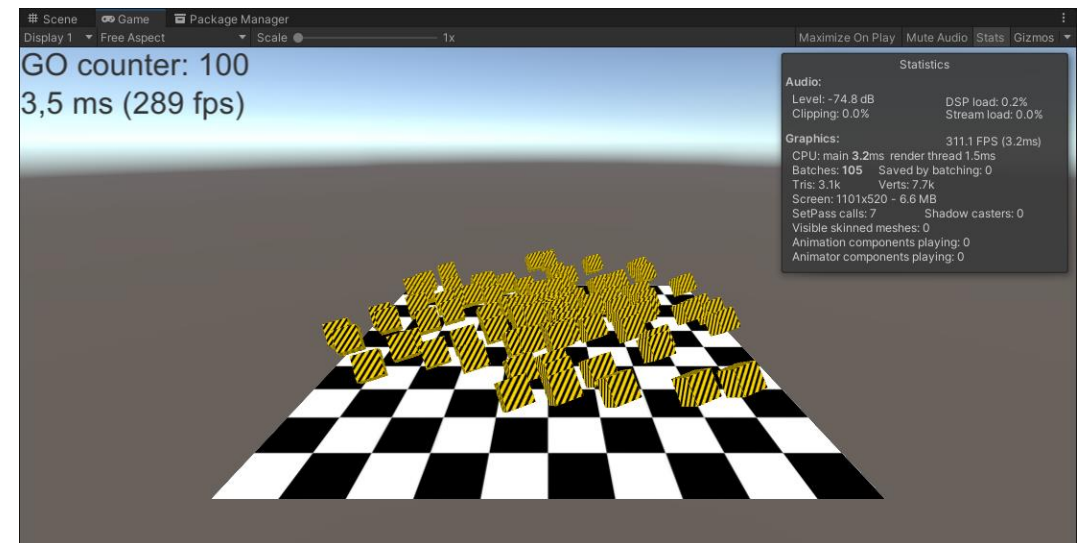
- Applicazione basata su architettura **ECS**, fornita dal package Entities.
- Networking implementato tramite l'utilizzo del package **NetCode**.
- Simulazione fisica realizzata utilizzando il package **Physics**.
- Funzionalità realizzate: **portali cambia colore, teletrasporti, raccolta oggetti**.



# Prototipi



## PROTOTIPO 2: STRESS TEST



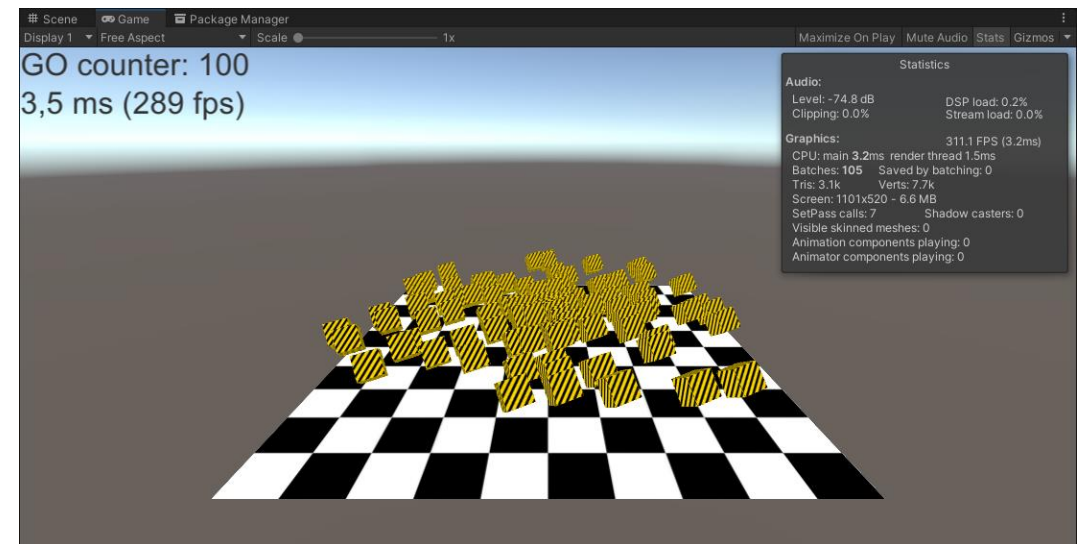
Prototipo per lo Stress Test (100 cubi).

# Prototipi



## PROTOTIPO 2: STRESS TEST

- Piccola applicazione che genera un numero arbitrario di cubi che vengono poi fatti ruotare.



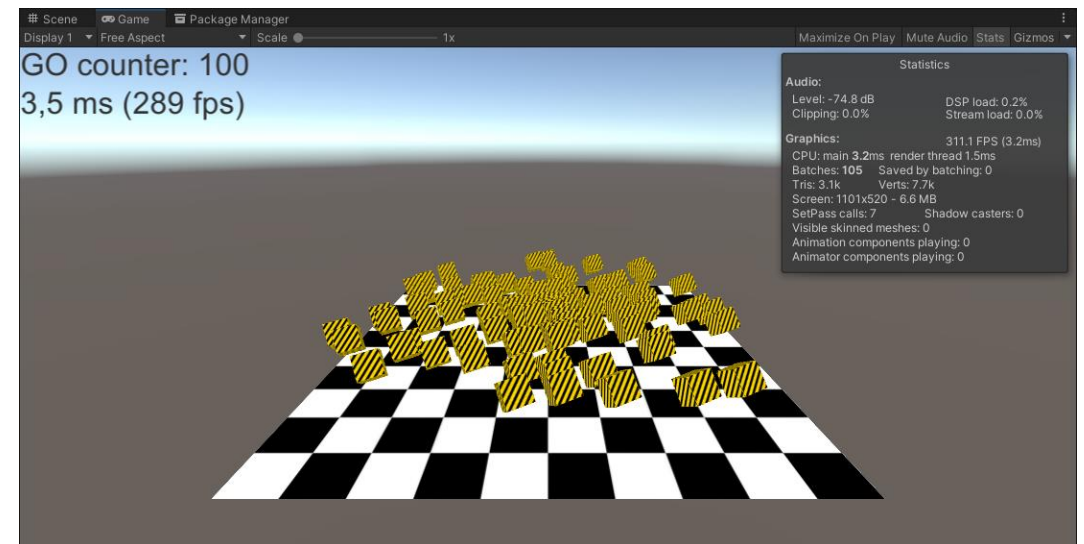
Prototipo per lo Stress Test (100 cubi).

# Prototipi



## PROTOTIPO 2: STRESS TEST

- Piccola applicazione che genera un numero arbitrario di cubi che vengono poi fatti ruotare.
- Rotazione dei cubi realizzata in modi differenti:



Prototipo per lo Stress Test (100 cubi).

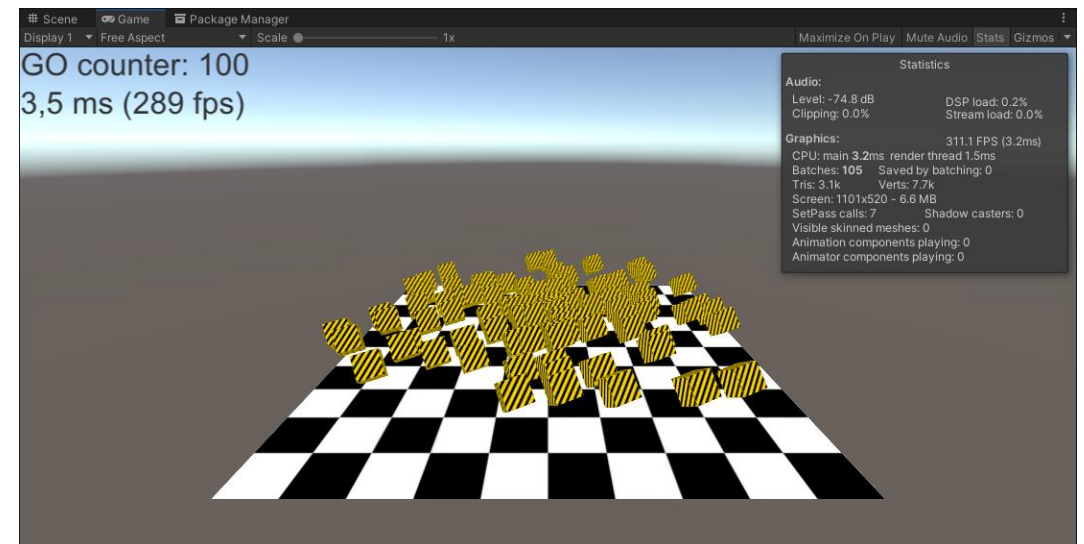


# Prototipi



## PROTOTIPO 2: STRESS TEST

- Piccola applicazione che genera un numero arbitrario di cubi che vengono poi fatti ruotare.
- Rotazione dei cubi realizzata in modi differenti:
  1. GameObject + MonoBehaviour (architettura classica).
  2. ECS «vanilla».
  3. ECS + Jobs.
  4. ECS + Parallel Jobs.
  5. ECS + Parallel Jobs + Burst.



Prototipo per lo Stress Test (100 cubi).

# Risultati Sperimentali

---



# Risultati Sperimentali

---



- Lo sviluppo dei prototipi ed i relativi test sono stati eseguiti sulla seguente

## **architettura hardware:**

- Processore Intel® Core™ i7-7700HQ con 4 core (8 processori logici) e frequenza di clock pari a 2.80GHz.
- Memoria RAM da 16GB.
- Scheda video NVIDIA GeForce GTX 1060.
- Sistema Operativo Microsoft Windows 10 Home, a 64 bit.

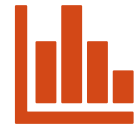
# Risultati Sperimentali

---



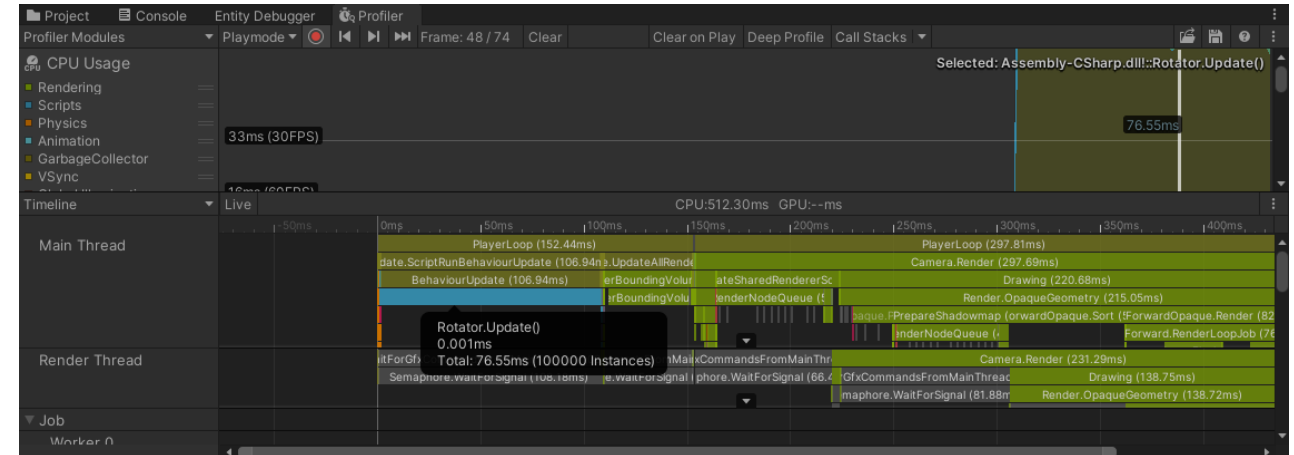
- Lo sviluppo dei prototipi ed i relativi test sono stati eseguiti sulla seguente **architettura hardware**:
  - Processore Intel® Core™ i7-7700HQ con 4 core (8 processori logici) e frequenza di clock pari a 2.80GHz.
  - Memoria RAM da 16GB.
  - Scheda video NVIDIA GeForce GTX 1060.
  - Sistema Operativo Microsoft Windows 10 Home, a 64 bit.
- I test sono stati eseguiti, per ciascuna soluzione, su 10, 100, 1.000, 10.000, 100.000 e 1.000.000 di cubi.

# Risultati Sperimentali



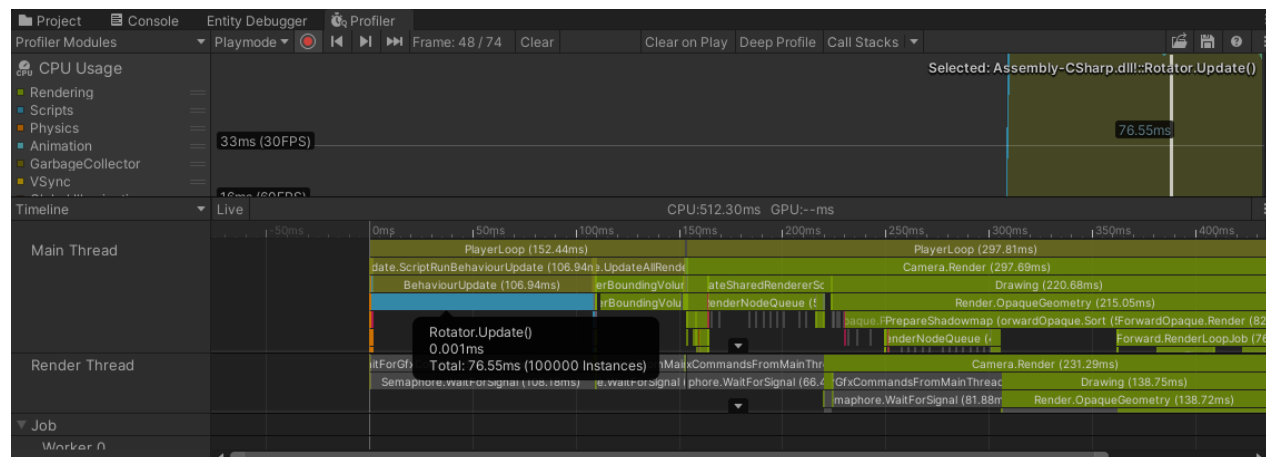
- Rotazione di 100.000 GameObject, tramite MonoBehaviour:

FPS	~2,5
ms Rotator	~77
ms totali CPU	~430

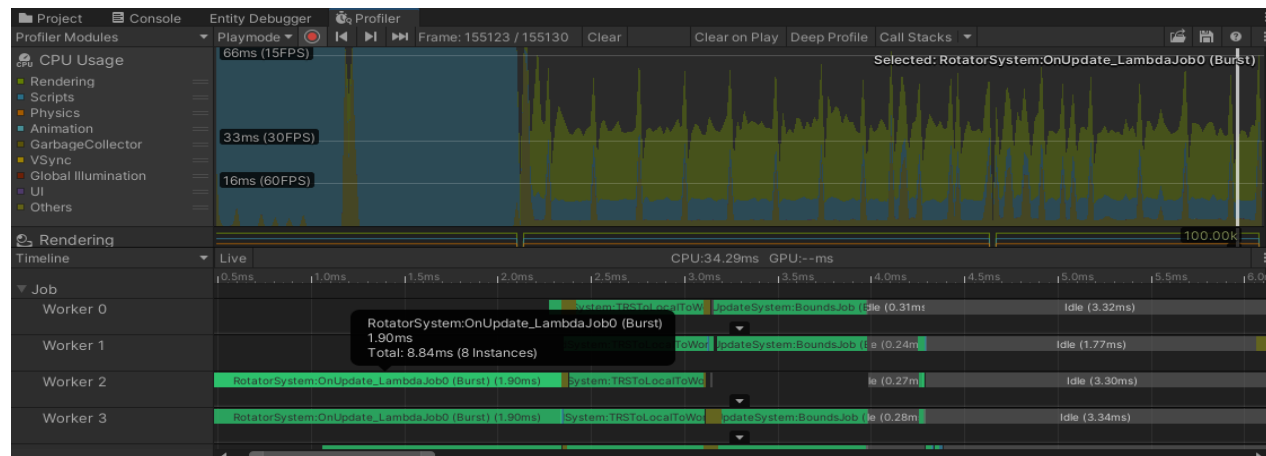


Age Group	Percentage
18-24	~75%
25-34	~85%
35-44	~70%
45-54	~65%
55-64	~55%
65+	~45%

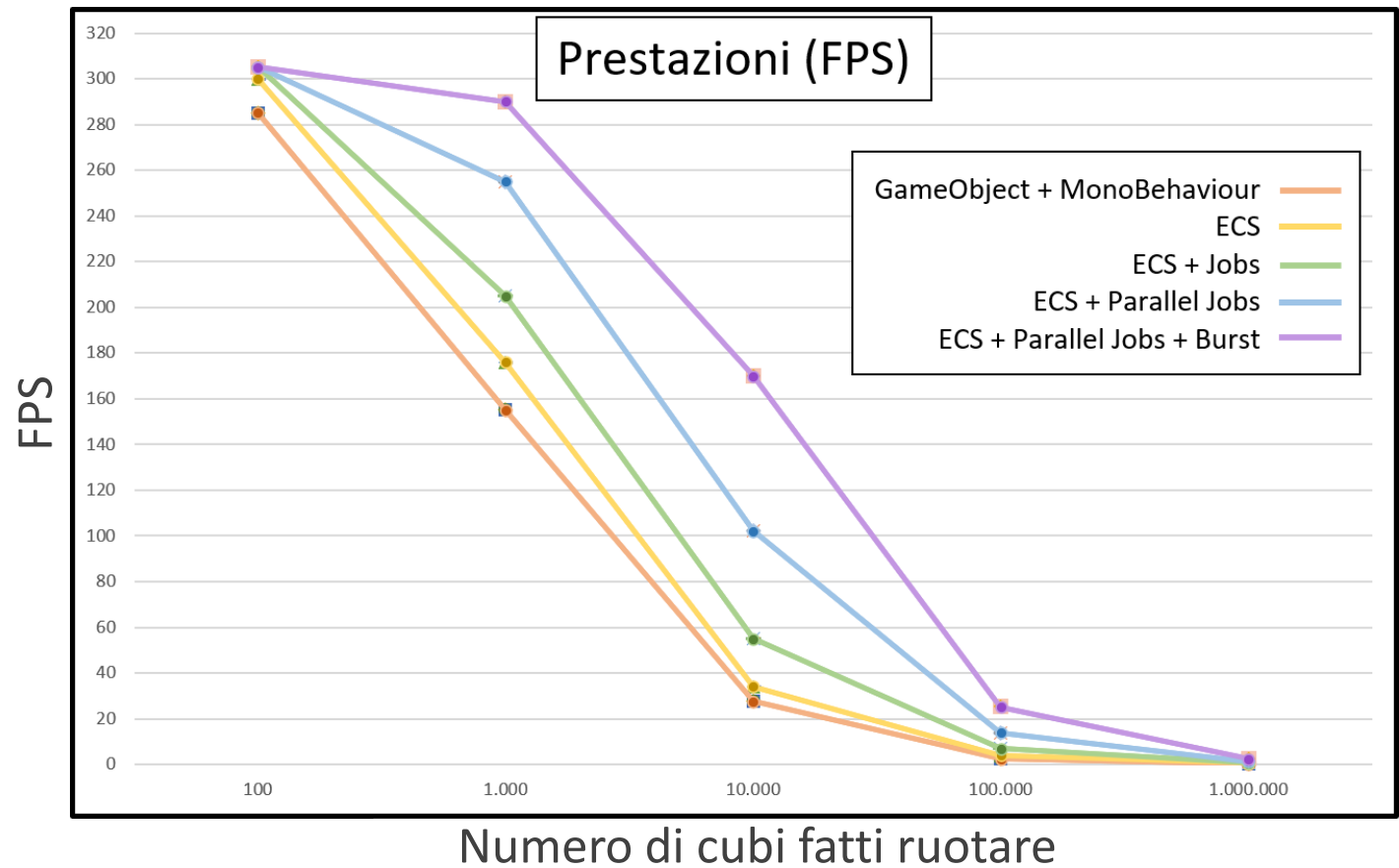
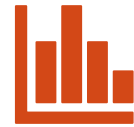
- |               |      |
|---------------|------|
| FPS           | ~2,5 |
| ms Rotator    | ~77  |
| ms totali CPU | ~430 |



- |               |       |
|---------------|-------|
| FPS           | ~25,2 |
| ms Rotator    | ~8,84 |
| ms totali CPU | ~39,8 |



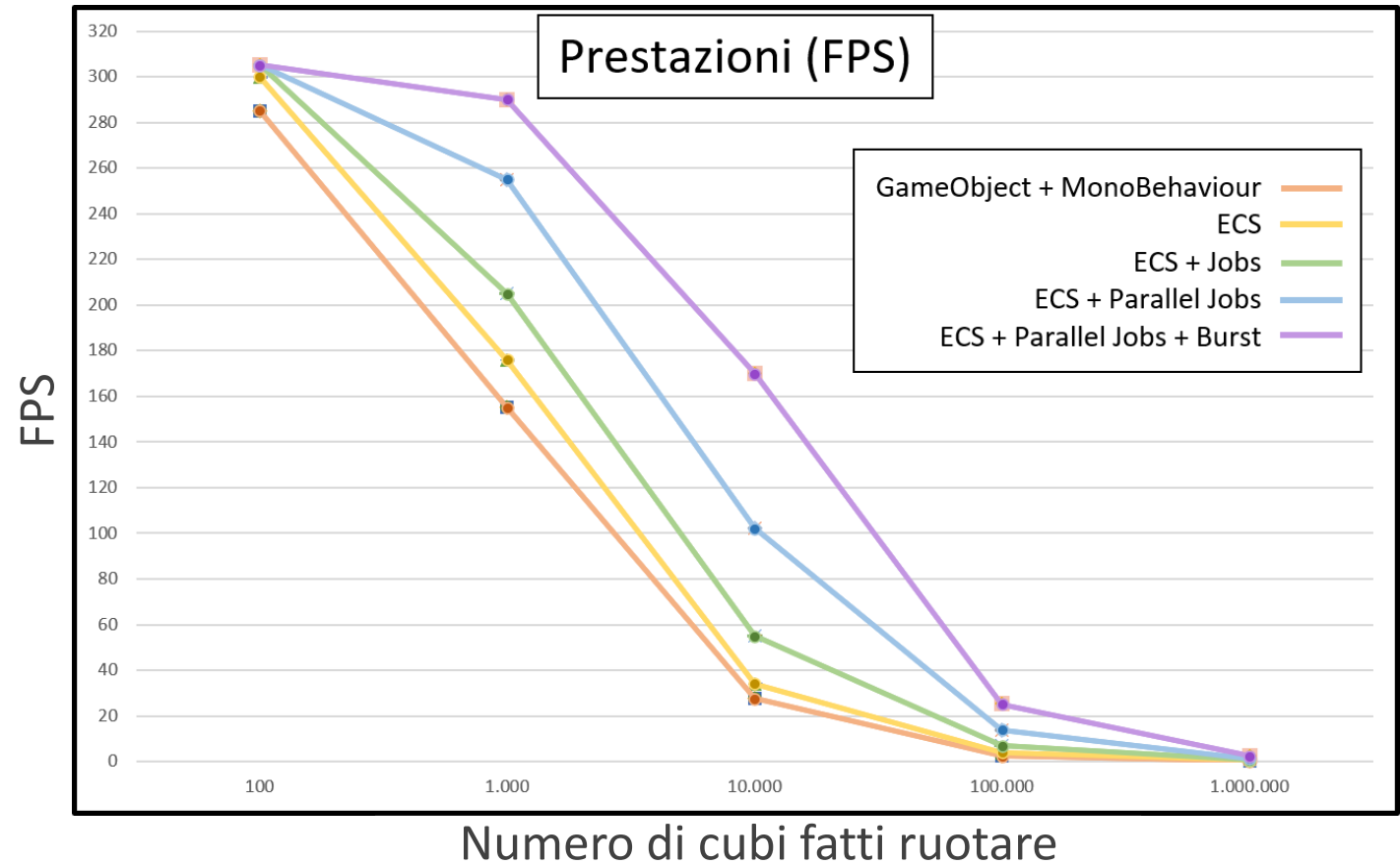
# Risultati Sperimentali



# Risultati Sperimentali



- Il grafico mostra le **differenza di FPS** fra le varie implementazioni.

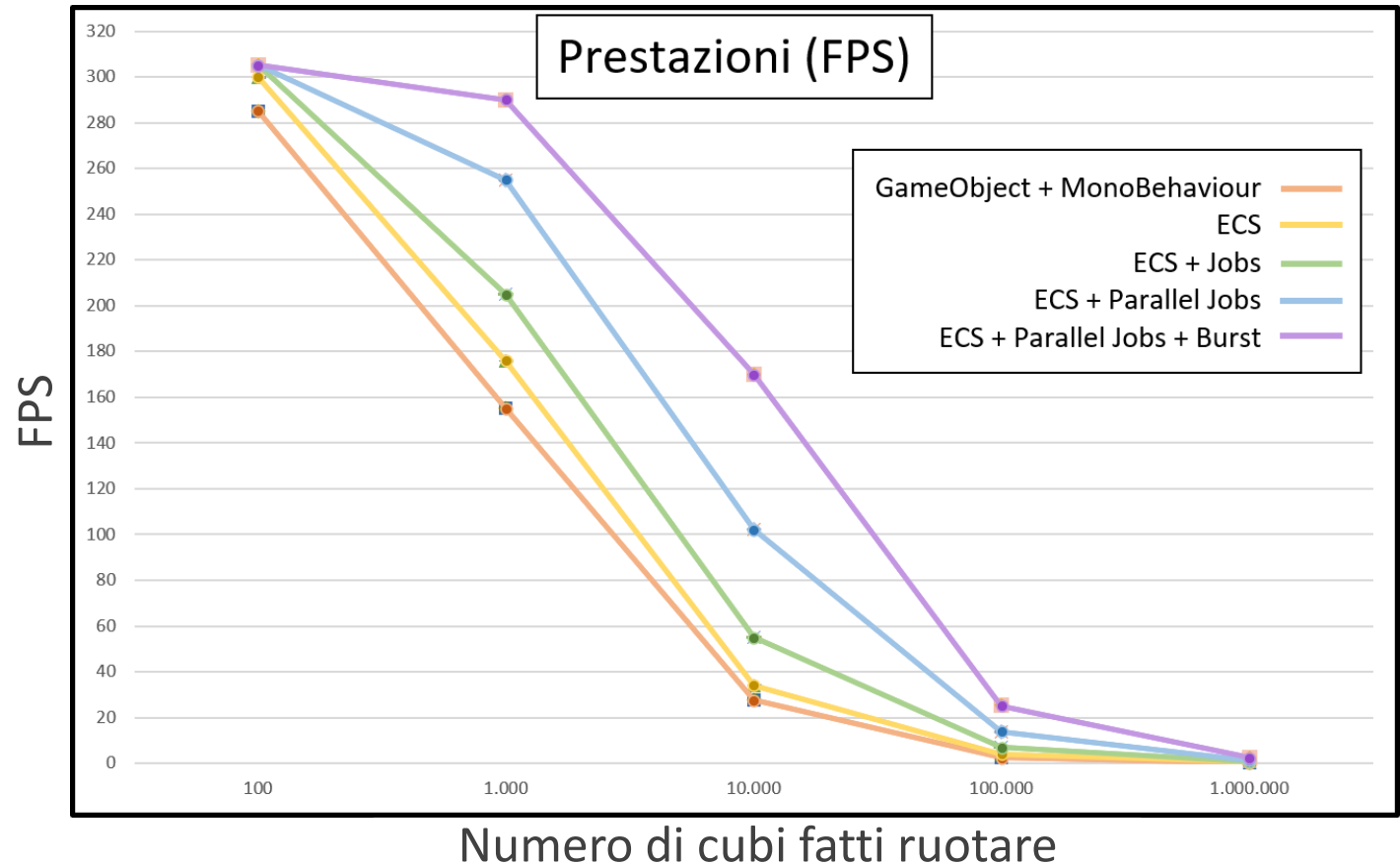




# Risultati Sperimentali



- Il grafico mostra le **differenza di FPS** fra le varie implementazioni.
- ECS e l'utilizzo dei package DOTS portano ad un **miglioramento di prestazioni**, rispetto all'architettura classica basata su GameObject.



# Conclusioni

---



**PRO**

**CONTRO**

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.

## CONTRO

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.

## CONTRO

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.
- **Uso massimizzato** delle risorse, soprattutto **CPU** e **cache**, grazie al layout dei dati.

## CONTRO

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.
- **Uso massimizzato** delle risorse, soprattutto **CPU** e **cache**, grazie al layout dei dati.
- Riduzione dei consumi (maggiore durata della batteria).

## CONTRO

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.
- **Uso massimizzato** delle risorse, soprattutto **CPU** e **cache**, grazie al layout dei dati.
- Riduzione dei consumi (maggiore durata della batteria).
- Modello di rete con **latenza minima**.

## CONTRO

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.
- **Uso massimizzato** delle risorse, soprattutto **CPU** e **cache**, grazie al layout dei dati.
- Riduzione dei consumi (maggiore durata della batteria).
- Modello di rete con **latenza minima**.

## CONTRO

- Ancora in fase di sviluppo.



# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.
- **Uso massimizzato** delle risorse, soprattutto **CPU** e **cache**, grazie al layout dei dati.
- Riduzione dei consumi (maggiore durata della batteria).
- Modello di rete con **latenza minima**.

## CONTRO

- Ancora in fase di sviluppo.
- I package saranno soggetti a possibili cambiamenti dal calibro differente.

# Conclusioni

---



## PRO

- **Separazione logica** dei dati e del comportamento.
- **Codice** altamente **leggibile** e **riutilizzabile**.
- **Uso massimizzato** delle risorse, soprattutto **CPU** e **cache**, grazie al layout dei dati.
- Riduzione dei consumi (maggiore durata della batteria).
- Modello di rete con **latenza minima**.

## CONTRO

- Ancora in fase di sviluppo.
- I package saranno soggetti a possibili cambiamenti dal calibro differente.
- Parti delle funzionalità presenti nell'architettura classica non ancora supportate.

# Conclusioni

---



**SVILUPPI FUTURI DOTS**

**POSSIBILI SVILUPPI FUTURI PROTOTIPO**

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.
- Riduzione del codice necessario per implementare il flusso di esecuzione di NetCode.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.
- Riduzione del codice necessario per implementare il flusso di esecuzione di NetCode.
- Aggiunta di ulteriori interfacce utili per l'analisi.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.
- Riduzione del codice necessario per implementare il flusso di esecuzione di NetCode.
- Aggiunta di ulteriori interfacce utili per l'analisi.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

- Lobby prepartita.



# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.
- Riduzione del codice necessario per implementare il flusso di esecuzione di NetCode.
- Aggiunta di ulteriori interfacce utili per l'analisi.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

- Lobby prepartita.
- Scoreboard.

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.
- Riduzione del codice necessario per implementare il flusso di esecuzione di NetCode.
- Aggiunta di ulteriori interfacce utili per l'analisi.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

- Lobby prepartita.
- Scoreboard.
- Sistema per la gestione dell'inventario.

# Conclusioni

---



## SVILUPPI FUTURI DOTS

- Release ufficiale di DOTS.
- Estensione del supporto alla conversione.
- Riduzione del codice necessario per implementare il flusso di esecuzione di NetCode.
- Aggiunta di ulteriori interfacce utili per l'analisi.

## POSSIBILI SVILUPPI FUTURI PROTOTIPO

- Lobby prepartita.
- Scoreboard.
- Sistema per la gestione dell'inventario.
- Nuova valutazione approfondita della latenza in rete.



# Unity DOTS

---

PERFORMANCE BY DEFAULT