

KOMPONENTSYSTEM

ELIA RÖNNING - SP20



**MYCKET ATT GÅ IGENOM SÅ SPARA
FRÅGORNA TILL SLUTET.**

DISCLAIMER!

**BEGREPPEN ÄR INTE HUGGNA I STEN OCH KAN VARIERA BEROENDE PÅ VEM DU PRATAR
MED. DET FINNS INGET RÄTT ELLER FEL.**

VAD ÄR ETT KOMPONENTSYSTEM?

VAD ÄR ETT KOMPONENTSYSTEM?

- Motorarkitektur

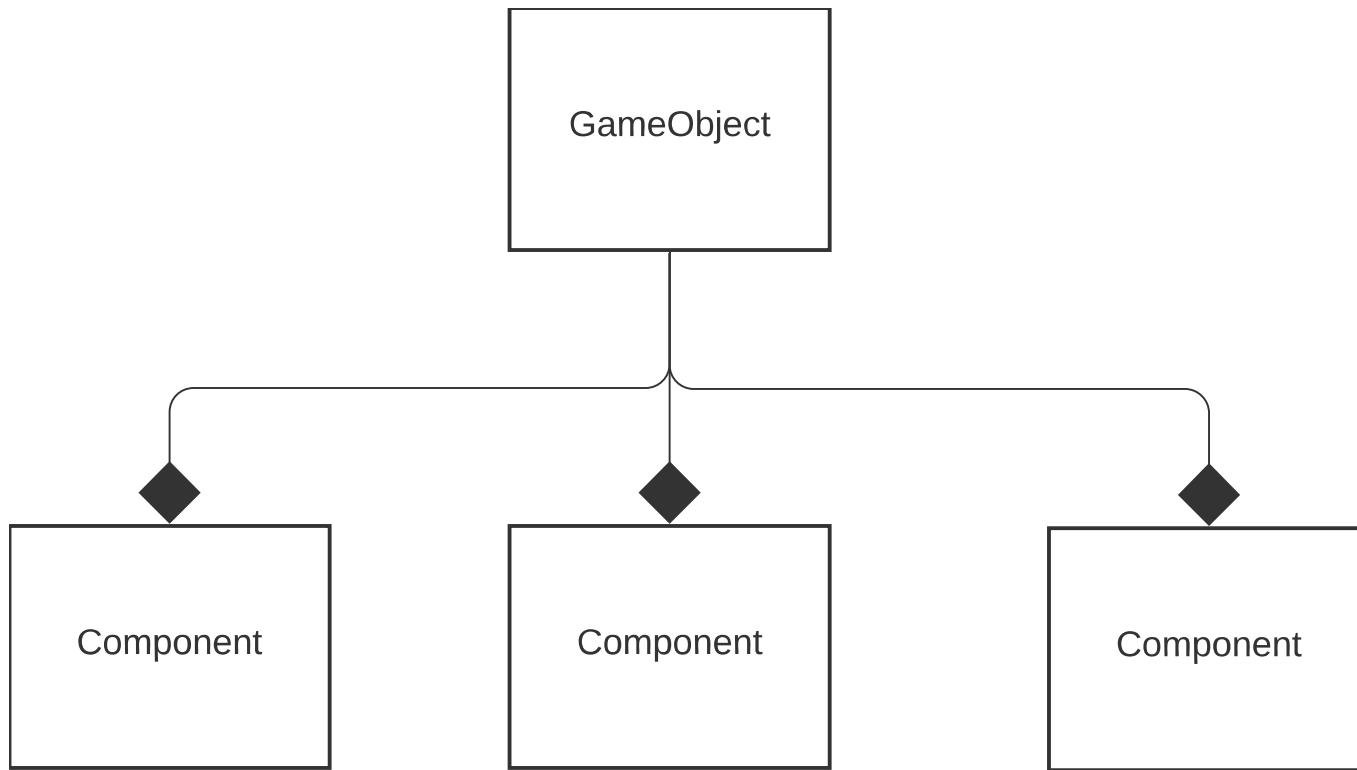
VAD ÄR ETT KOMPONENTSYSTEM?

- Motorarkitektur
- Composition over Inheritance

VAD ÄR ETT KOMPONENTSYSTEM?

- Motorarkitektur
- Composition over Inheritance
- Dynamisk och generell hantering av objekt i spelvärlden

GAMEOBJECTS HAR KOMPONENTER



COMPOSITION OVER INHERITANCE

```
1 class PlayerCharacter :  
2     public Transform, public Sprite,  
3     public PhysicsBody, public BoxCollider  
4 {  
5     public:  
6         PlayerCharacter();  
7         ~PlayerCharacter();  
8  
9         void Init();  
10        void Update(float dt);  
11        void Render();  
12  
13        // ...  
14  
15     private:
```

COMPOSITION OVER INHERITANCE

```
1 class PlayerCharacter :  
2     public Transform, public Sprite,  
3     public PhysicsBody, public BoxCollider  
4 {  
5     public:  
6         PlayerCharacter();  
7         ~PlayerCharacter();  
8  
9         void Init();  
10        void Update(float dt);  
11        void Render();  
12  
13        // ...  
14  
15     private:
```

DYNAMISK KOMPOSITION, ISTÄLLET FÖR STATISK.

DYNAMISK KOMPOSITION, ISTÄLLET FÖR STATISK.

- Kompositionen av ett objekt kan ändras i runtime.

DYNAMISK KOMPOSITION, ISTÄLLET FÖR STATISK.

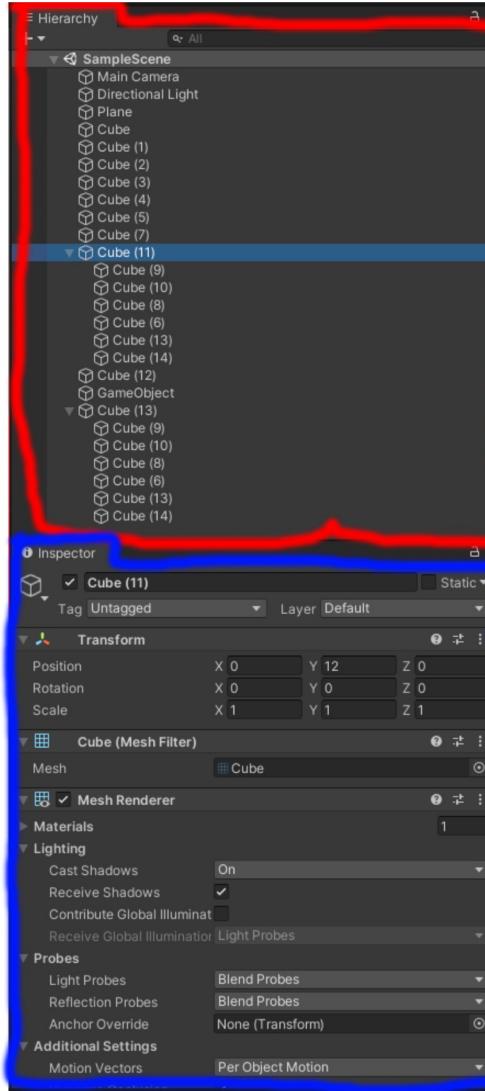
- Kompositionen av ett objekt kan ändras i runtime.
- Gjort för serialisering av objekt.

DYNAMISK KOMPOSITION, ISTÄLLET FÖR STATISK.

- Kompositionen av ett objekt kan ändras i runtime.
- Gjort för serialisering av objekt.
- GameObjects ska kunna laddas in från exempelvis JSON, istället för att bestämmas vid compile time.

EXEMPEL

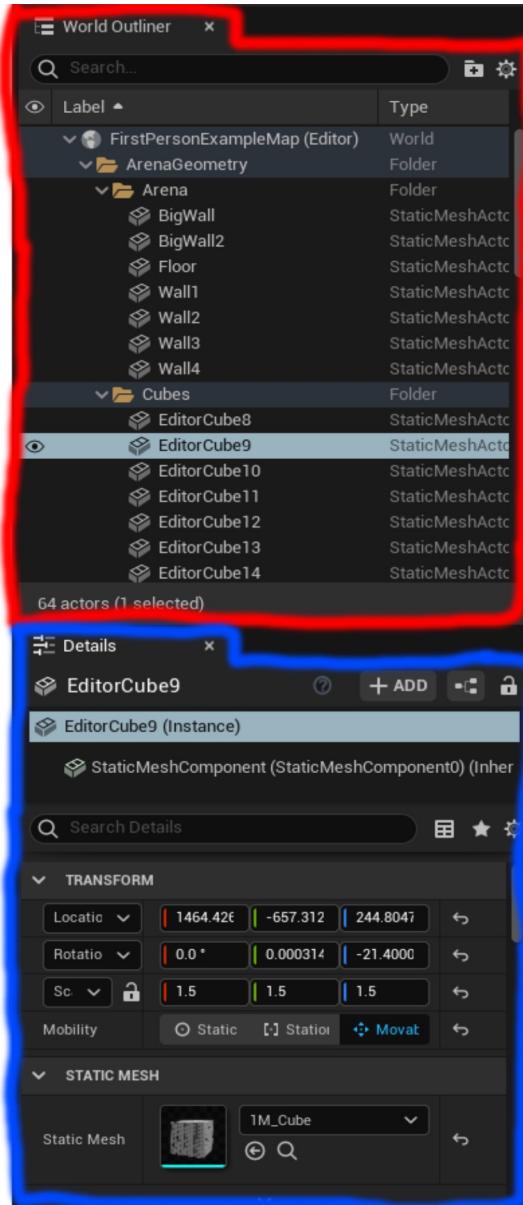
GameObjects



Components

EXEMPEL

Actors

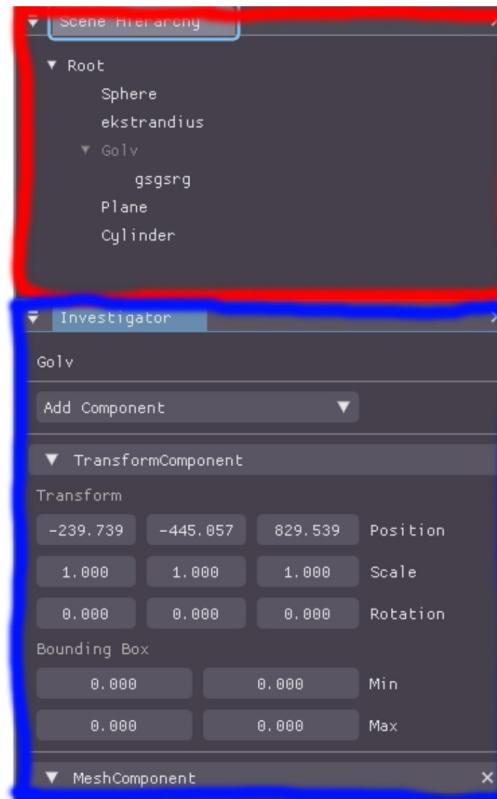


Components

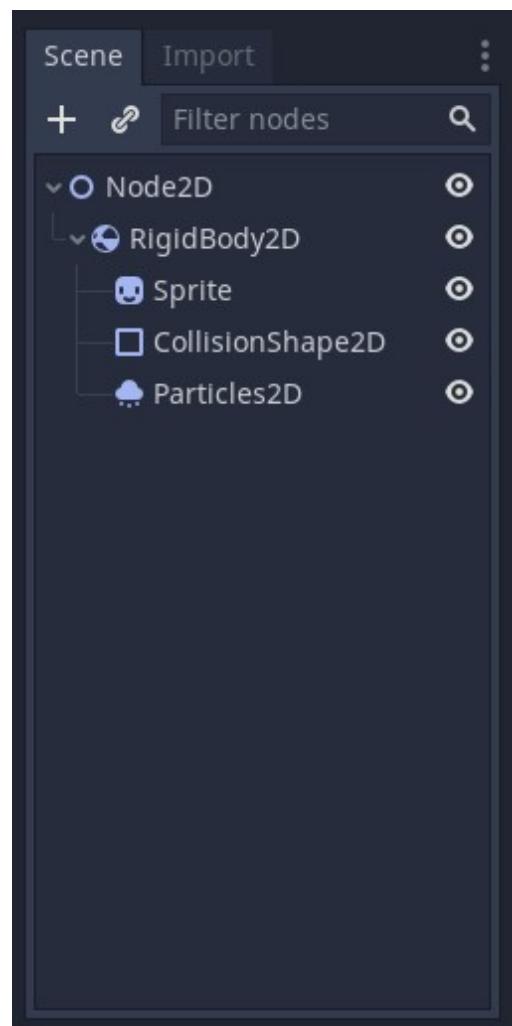
EXEMPEL

Entities

Components



EXEMPEL



VAD ÄR INTE ETT KOMPONENTSYSTEM? NÅGRA EXEMPEL:

ARV

```
1 class PlayerCharacter :  
2     public Transform, public Sprite,  
3     public PhysicsBody, public BoxCollider  
4 {  
5     public:  
6         PlayerCharacter();  
7         ~PlayerCharacter();  
8  
9     void Init();
```

KOMPOSITION I COMPILE TIME

```
1 class PlayerCharacter
2 {
3 public:
4     PlayerCharacter();
5     ~PlayerCharacter();
6
7     void Init();
8     void Update(float dt);
9     void Render();
10
11    // ...
12
13 private:
14     Transform myTransform;
15     Sprite mySprite;
```

TYPE OBJECT PATTERN

```
class MonsterEnemy
{
public:
    MonsterEnemy(MonsterType& aTypeObject);

private:
    MonsterType& myType;
};
```

I ETT KOMPONENTSYSTEM...

I ETT KOMPONENTSYSTEM...

- ...är spelobjekt uppbyggda av komponenter.

I ETT KOMPONENTSYSTEM...

- ...är spelobjekt uppbyggda av komponenter.
- ...komponeras objekt i runtime.

I ETT KOMPONENTSYSTEM...

- ...är spelobjekt uppbyggda av komponenter.
- ...komponeras objekt i runtime.
- ...sköter komponenterna sig själva.

EXEMPELKOD!



NÅGRA SAKER ATT TÄNKA PÅ:

NÅGRA SAKER ATT TÄNKA PÅ:

- All exempelkod ligger uppe på GitHub och är fungerande.

NÅGRA SAKER ATT TÄNKA PÅ:

- All exempelkod ligger uppe på GitHub och är fungerande.
- Koden är ganska shit och verkligen bare minimum.

NÅGRA SAKER ATT TÄNKA PÅ:

- All exempelkod ligger uppe på GitHub och är fungerande.
- Koden är ganska shit och verkligen bare minimum.
- Och igen, det finns inget rätt eller fel, detta är bara ett sätt.

CLASS GAMEOBJECT;

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void LateUpdate();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual ~GameObject();
17}
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual ~GameObject();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.H

```
1 class GameObject
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     GameObject();
6     virtual ~GameObject();
7
8     GameObject(const GameObject& other);
9     GameObject(GameObject&& other) noexcept;
10    GameObject& operator=(const GameObject& other);
11    GameObject& operator=(GameObject&& other) noexcept;
12
13    /* Virtual Functions */
14    virtual void Init();
15    virtual void Update(float dt);
16    virtual void Draw();
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

GAMEOBJECT.CPP

```
1 #include "GameObject.h"
2
3 #include "Component.hpp"
4
5 std::vector<GameObject> GameObject::ourGameObjects;
6
7 GameObject::GameObject()
8     : myComponents{ }
9 {
10     ourGameObjects.push_back(this);
11 }
12
13 GameObject::~GameObject()
14 {
15     for (Component* comp : myComponents)
16     {
```

COMPONENT.HPP

```
1 class Component
2 {
3 public:
4     Component() = default;
5     virtual ~Component() = default;
6
7     Component(const Component&) = default;
8     Component(Component&&) = default;
9     Component& operator=(const Component&) = default;
10    Component& operator=(Component&&) = default;
11
12    virtual void Init(class GameObject* parent) = 0;
13    virtual void Update(GameObject* parent, float dt) = 0;
14    virtual void Render(GameObject* parent) = 0;
15 }
```

COMPONENT.HPP

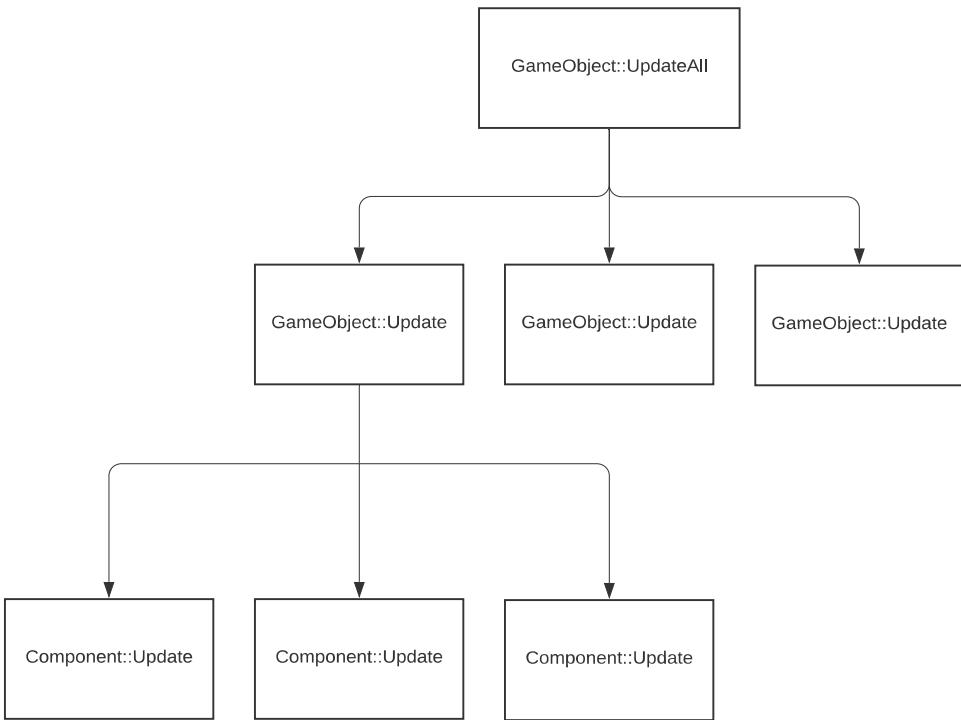
```
1 class Component
2 {
3 public:
4     Component() = default;
5     virtual ~Component() = default;
6
7     Component(const Component&) = default;
8     Component(Component&&) = default;
9     Component& operator=(const Component&) = default;
10    Component& operator=(Component&&) = default;
11
12    virtual void Init(class GameObject* parent) = 0;
13    virtual void Update(GameObject* parent, float dt) = 0;
14    virtual void Render(GameObject* parent) = 0;
15 }
```

KOMPONENTSYSTEMETS INTERFACE

```
void Init()
{
    GameObject::InitAll();
}

void Update(float dt)
{
    GameObject::UpdateAll();
    GameObject::RenderAll();
}
```

KOMPONENTSYSTEMETS INTERFACE



EXEMPELKOMPONENT!

SPRITECOMPONENT 

SPRITECOMPONENT.H

```
1 class SpriteComponent : public Component
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     SpriteComponent();
6     ~SpriteComponent();
7
8     SpriteComponent(const SpriteComponent& sc);
9     SpriteComponent(SpriteComponent&& sc) noexcept;
10    SpriteComponent& operator=(const SpriteComponent& sc);
11    SpriteComponent& operator=(SpriteComponent&& sc) noexcept;
12
13    /* Component Virtual Methods */
14    void Init(class GameObject* parent);
15    void Update(GameObject* parent, float dt);
16    virtual ~SpriteComponent();
```

SPRITECOMPONENT.H

```
1 class SpriteComponent : public Component
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     SpriteComponent();
6     ~SpriteComponent();
7
8     SpriteComponent(const SpriteComponent& sc);
9     SpriteComponent(SpriteComponent&& sc) noexcept;
10    SpriteComponent& operator=(const SpriteComponent& sc);
11    SpriteComponent& operator=(SpriteComponent&& sc) noexcept;
12
13    /* Component Virtual Methods */
14    void Init(class GameObject* parent);
15    void Update(GameObject* parent, float dt);
16    void Draw();
```

SPRITECOMPONENT.H

```
1 class SpriteComponent : public Component
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     SpriteComponent();
6     ~SpriteComponent();
7
8     SpriteComponent(const SpriteComponent& sc);
9     SpriteComponent(SpriteComponent&& sc) noexcept;
10    SpriteComponent& operator=(const SpriteComponent& sc);
11    SpriteComponent& operator=(SpriteComponent&& sc) noexcept;
12
13    /* Component Virtual Methods */
14    void Init(class GameObject* parent);
15    void Update(GameObject* parent, float dt);
16    void Draw(GLContext* context);
```

SPRITECOMPONENT.H

```
1 class SpriteComponent : public Component
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     SpriteComponent();
6     ~SpriteComponent();
7
8     SpriteComponent(const SpriteComponent& sc);
9     SpriteComponent(SpriteComponent&& sc) noexcept;
10    SpriteComponent& operator=(const SpriteComponent& sc);
11    SpriteComponent& operator=(SpriteComponent&& sc) noexcept;
12
13    /* Component Virtual Methods */
14    void Init(class GameObject* parent);
15    void Update(GameObject* parent, float dt);
16    void Draw(GLContext* context);
```

SPRITECOMPONENT.H

```
1 class SpriteComponent : public Component
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     SpriteComponent();
6     ~SpriteComponent();
7
8     SpriteComponent(const SpriteComponent& sc);
9     SpriteComponent(SpriteComponent&& sc) noexcept;
10    SpriteComponent& operator=(const SpriteComponent& sc);
11    SpriteComponent& operator=(SpriteComponent&& sc) noexcept;
12
13    /* Component Virtual Methods */
14    void Init(class GameObject* parent);
15    void Update(GameObject* parent, float dt);
16    void OnCollision(GameObject* other);
```

SPRITECOMPONENT.H

```
1 class SpriteComponent : public Component
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     SpriteComponent();
6     ~SpriteComponent();
7
8     SpriteComponent(const SpriteComponent& sc);
9     SpriteComponent(SpriteComponent&& sc) noexcept;
10    SpriteComponent& operator=(const SpriteComponent& sc);
11    SpriteComponent& operator=(SpriteComponent&& sc) noexcept;
12
13    /* Component Virtual Methods */
14    void Init(class GameObject* parent);
15    void Update(GameObject* parent, float dt);
16    void Draw();
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

SPRITECOMPONENT.CPP

```
1 #include "SpriteComponent.h"
2
3 #include "GameObject.h"
4
5 SpriteComponent::SpriteComponent()
6     : myTexture(0U)
7     , mySize({ 0, 0 })
8     , myColor({ 255, 255, 255 })
9     , myTexturePath("")
10 { }
11
12 SpriteComponent::~SpriteComponent()
13 {
14     if (myTexture)
15     {
16         // ...
17     }
18 }
```

ANVÄNDNINGSEXEMPEL!

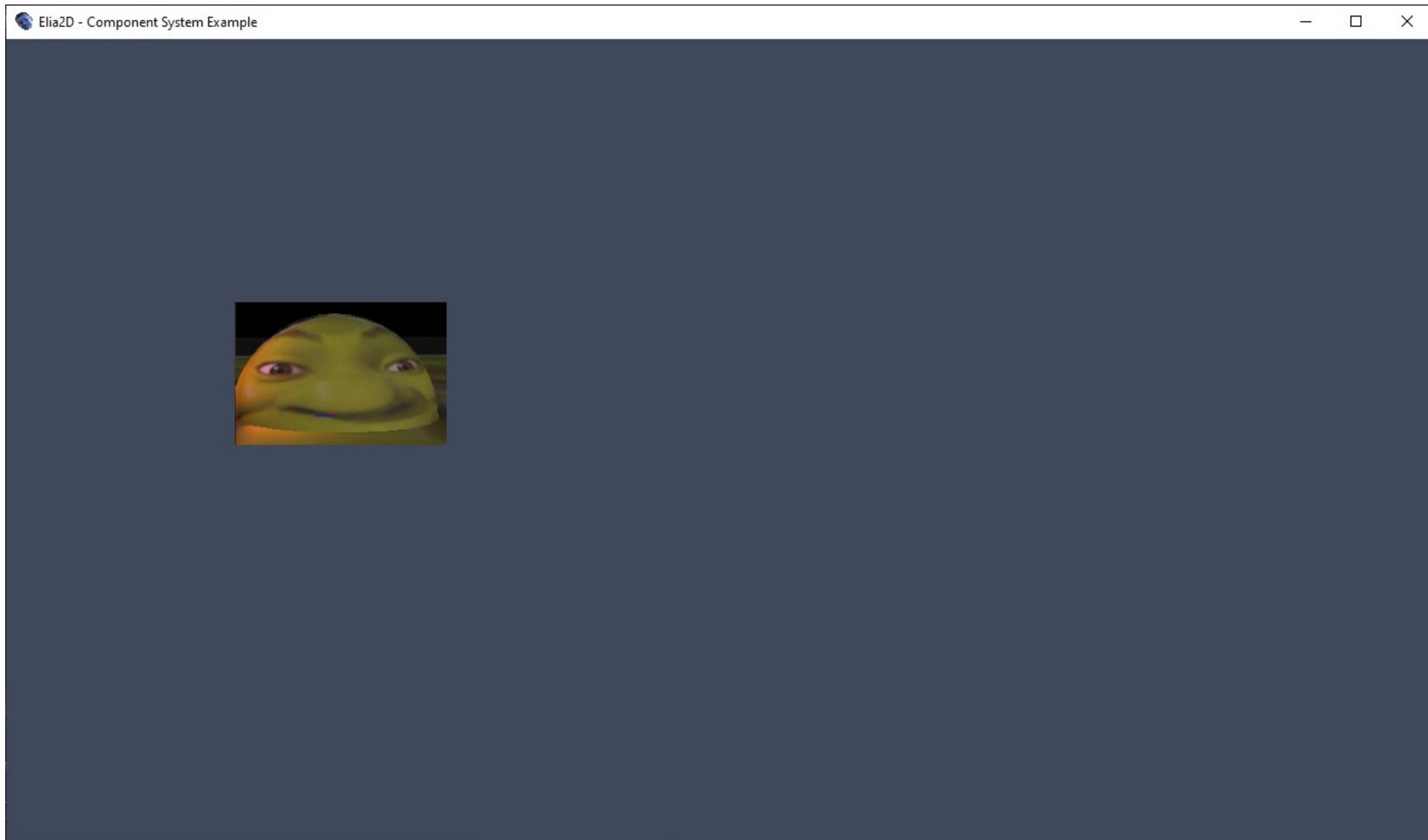


SKAPA GAMEOBJECT MED EN SPRITECOMPONENT.

```
GameObject* testObj = nullptr;

void Init()
{
    testObj = new GameObject;
    testObj->SetPosition({ 300, 300 });
    SpriteComponent* spr = testObj->AddComponent<SpriteComponent>()
    spr->SetSpritePath("../Assets/dawdwdawda.png");
}
```

RESULTAT



UPPDATERA GAMEOBJECT

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5
6     float factor = abs(sinf(totalTime));
7     testObj->SetPosition(
8         { 400.0f + 300.0f * factor, 300.f });
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    testObj->GetComponent()
12        ->SetColor({ color, 0, uint8_t(255 - color) });
13 }
```

UPPDATERA GAMEOBJECT

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5
6     float factor = abs(sinf(totalTime));
7     testObj->SetPosition(
8         { 400.0f + 300.0f * factor, 300.f });
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    testObj->GetComponent()
12        ->SetColor({ color, 0, uint8_t(255 - color) });
13 }
```

UPPDATERA GAMEOBJECT

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5
6     float factor = abs(sinf(totalTime));
7     testObj->SetPosition(
8         { 400.0f + 300.0f * factor, 300.f });
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    testObj->GetComponent()
12        ->SetColor({ color, 0, uint8_t(255 - color) });
13 }
```

UPPDATERA GAMEOBJECT

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5
6     float factor = abs(sinf(totalTime));
7     testObj->SetPosition(
8         { 400.0f + 300.0f * factor, 300.f });
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    testObj->GetComponent()
12        ->SetColor({ color, 0, uint8_t(255 - color) });
13 }
```

UPPDATERA GAMEOBJECT

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5
6     float factor = abs(sinf(totalTime));
7     testObj->SetPosition(
8         { 400.0f + 300.0f * factor, 300.f });
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    testObj->GetComponent()
12        ->SetColor({ color, 0, uint8_t(255 - color) });
13 }
```

RESULTAT

ECS



ENTITY-COMPONENT-SYSTEM

ENTITY-COMPONENT-SYSTEM

- Inte samma sak som ett komponentsystem.

ENTITY-COMPONENT-SYSTEM

- Inte samma sak som ett komponentsystem.
- Men som sagt, begreppen är inte huggna i sten och alla har olika meningar.

ENTITY-COMPONENT-SYSTEM

- Inte samma sak som ett komponentsystem.
- Men som sagt, begreppen är inte huggna i sten och alla har olika meningar.
- Definition från Designmönster för Spelutveckling.

NAMNET

NAMNET

- Inte ett system av entities och komponenter.

NAMNET

- Inte ett system av entities och komponenter.
- De tre "pelarna" som utgör ett ECS.

NAMNET

- Inte ett system av entities och komponenter.
- De tre "pelarna" som utgör ett ECS.
- Tänk Model-View-Controller.

CS VS. ECS

CS VS. ECS

CS VS. ECS

- Komponentsystem - Objektorientering

CS VS. ECS

- Komponentssystem - Objektorientering
- Entity-Component-System - Dataorientering

CS VS. ECS

- Komponentsystem - Objektorientering
- Entity-Component-System - Dataorientering
- (Inte att förväxla med datadriven programmering)

DATAORIENTERING

DATAORIENTERING

- Kan ses som ett paradigm, precis som objektorientering.

DATAORIENTERING

- Kan ses som ett paradigm, precis som objektorientering.
- Objektorientering: Bunta ihop beteende och data i objekt.

DATAORIENTERING

- Kan ses som ett paradigm, precis som objektorientering.
- Objektorientering: Bunta ihop beteende och data i objekt.
- Dataorientering: Separera data och beteende.

DATAORIENTERING

- Kan ses som ett paradigm, precis som objektorientering.
- Objektorientering: Bunta ihop beteende och data i objekt.
- Dataorientering: Separera data och beteende.
- Array of structs vs. Struct of arrays: Undviker hierarkiska datastrukturer

DATAORIENTERING

- Kan ses som ett paradigm, precis som objektorientering.
- Objektorientering: Bunta ihop beteende och data i objekt.
- Dataorientering: Separera data och beteende.
- Array of structs vs. Struct of arrays: Undviker hierarkiska datastrukturer
- Optimerat för CPU-cachen: Hot/Cold splitting.

ARRAY OF STRUCTS VS. STRUCT OF ARRAYS

```
1 class AssociativeArrayOO
2 {
3     struct KeyValuePair
4     {
5         Key k;
6         Value v;
7     };
8     KeyValuePair data[size];
9 }
10
11 class AssociativeArrayDO
12 {
13     Key keys[size];
14     Value values[size];
15 }
```

ARRAY OF STRUCTS VS. STRUCT OF ARRAYS

```
1 class AssociativeArrayOO
2 {
3     struct KeyValuePair
4     {
5         Key k;
6         Value v;
7     };
8     KeyValuePair data[size];
9 }
10
11 class AssociativeArrayDO
12 {
13     Key keys[size];
14     Value values[size];
15 }
```

ARRAY OF STRUCTS VS. STRUCT OF ARRAYS

```
1 class AssociativeArrayOO
2 {
3     struct KeyValuePair
4     {
5         Key k;
6         Value v;
7     };
8     KeyValuePair data[size];
9 }
10
11 class AssociativeArrayDO
12 {
13     Key keys[size];
14     Value values[size];
15 }
```

HOT/COLD SPLITTING

Key	Value	Key	Value	Key	Value	Key	Value
-----	-------	-----	-------	-----	-------	-----	-------

Key							
-----	-----	-----	-----	-----	-----	-----	-----

ENTITY-COMPONENT-SYSTEM

ENTITY-COMPONENT-SYSTEM

- **Entity:** Associera komponenter med varandra; ofta bara ett ID.

ENTITY-COMPONENT-SYSTEM

- **Entity:** Associera komponenter med varandra; ofta bara ett ID.
- **Component:** All data av spelobjekten är atomiserade i komponenter, har inget beteende.

ENTITY-COMPONENT-SYSTEM

- **Entity:** Associera komponenter med varandra; ofta bara ett ID.
- **Component:** All data av spelobjekten är atomiserade i komponenter, har inget beteende.
- **System:** Utför beteende på komponenter, har inget eget tillstånd; ofta bara en funktion.

FÖRDELAR

FÖRDELAR

- Bättre prestanda.

FÖRDELAR

- Bättre prestanda.
- Lättare att utföra datatransformationer över flera trådar.

FÖRDELAR

- Bättre prestanda.
- Lättare att utföra datatransformationer över flera trådar.
- Mer robust med ID istället pekare.

EXEMPEL: UNITY DOTS

EXEMPELKOD



ENTITY



ENTITIES ÄR IDN

```
1 #include  
2 using Entity = uint32_t;
```

ENTITIES ÄR IDN

```
1 #include  
2 using Entity = uint32_t;
```

**VI BEHÖVER ETT SÄTT ATT HÅLLA REDA PÅ
UPPTAGNA/LEDIGA IDN.**

VI BEHÖVER ETT SÄTT ATT HÅLLA REDA PÅ UPPTAGNA/LEDIGA IDN.

- std::set?

VI BEHÖVER ETT SÄTT ATT HÅLLA REDA PÅ UPPTAGNA/LEDIGA IDN.

- std::set?
- bitset?

VI BEHÖVER ETT SÄTT ATT HÅLLA REDA PÅ UPPTAGNA/LEDIGA IDN.

- std::set?
- bitset?
- Helt okej lösningar...

VI BEHÖVER ETT SÄTT ATT HÅLLA REDA PÅ UPPTAGNA/LEDIGA IDN.

- std::set?
- bitset?
- Helt okej lösningar...
- ..men jag körde på en linked list.

ENTITYMANAGER.H

```
1 class EntityManager
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     EntityManager();
6     ~EntityManager() = default;
7
8     EntityManager(const EntityManager& other);
9     EntityManager(EntityManager&& other) noexcept;
10    EntityManager& operator=(const EntityManager& other);
11    EntityManager& operator=(EntityManager&& other);
12
13    /* Public Interface */
14    Entity GetEntity();
15    void ReturnEntity(Entity e);
16
```

ENTITYMANAGER.H

```
1 class EntityManager
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     EntityManager();
6     ~EntityManager() = default;
7
8     EntityManager(const EntityManager& other);
9     EntityManager(EntityManager&& other) noexcept;
10    EntityManager& operator=(const EntityManager& other);
11    EntityManager& operator=(EntityManager&& other);
12
13    /* Public Interface */
14    Entity GetEntity();
15    void ReturnEntity(Entity e);
16
```

ENTITYMANAGER.H

```
1 class EntityManager
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     EntityManager();
6     ~EntityManager() = default;
7
8     EntityManager(const EntityManager& other);
9     EntityManager(EntityManager&& other) noexcept;
10    EntityManager& operator=(const EntityManager& other);
11    EntityManager& operator=(EntityManager&& other);
12
13    /* Public Interface */
14    Entity GetEntity();
15    void ReturnEntity(Entity e);
16
```

ENTITYMANAGER.H

```
1 class EntityManager
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     EntityManager();
6     ~EntityManager() = default;
7
8     EntityManager(const EntityManager& other);
9     EntityManager(EntityManager&& other) noexcept;
10    EntityManager& operator=(const EntityManager& other);
11    EntityManager& operator=(EntityManager&& other);
12
13    /* Public Interface */
14    Entity GetEntity();
15    void ReturnEntity(Entity e);
16
```

ENTITYMANAGER.H

```
1 class EntityManager
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     EntityManager();
6     ~EntityManager() = default;
7
8     EntityManager(const EntityManager& other);
9     EntityManager(EntityManager&& other) noexcept;
10    EntityManager& operator=(const EntityManager& other);
11    EntityManager& operator=(EntityManager&& other);
12
13    /* Public Interface */
14    Entity GetEntity();
15    void ReturnEntity(Entity e);
16
```

ENTITYMANAGER.H

```
1 class EntityManager
2 {
3 public:
4     /* Constructors, Destructor, Assignment Operators */
5     EntityManager();
6     ~EntityManager() = default;
7
8     EntityManager(const EntityManager& other);
9     EntityManager(EntityManager&& other) noexcept;
10    EntityManager& operator=(const EntityManager& other);
11    EntityManager& operator=(EntityManager&& other);
12
13    /* Public Interface */
14    Entity GetEntity();
15    void ReturnEntity(Entity e);
16
```

INITIALISERA LÄNKADE LISTAN

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

INITIALISERA LÄNKADE LISTAN

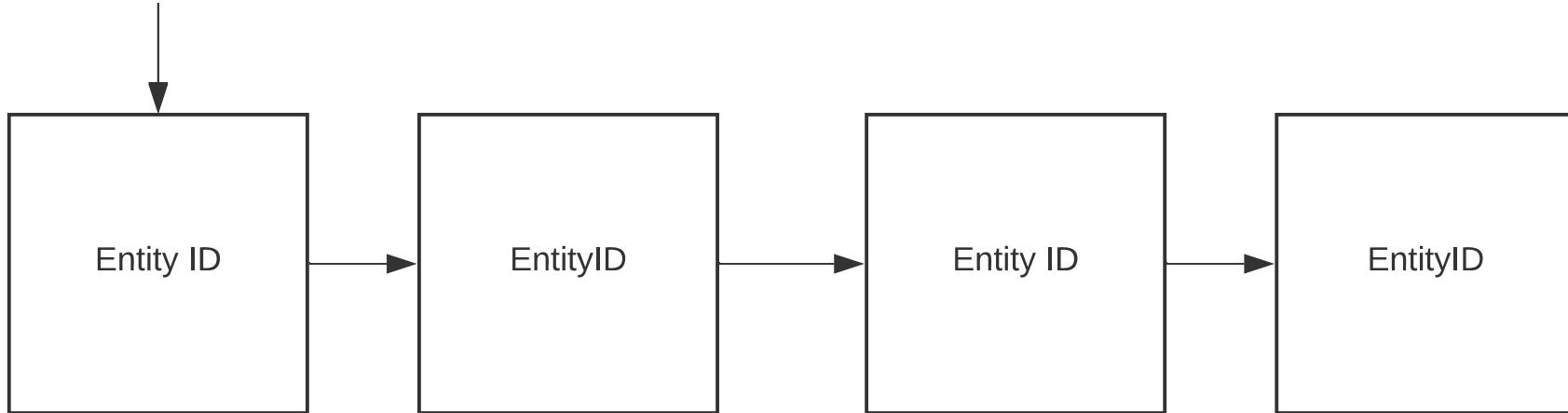
```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

INITIALISERA LÄNKADE LISTAN

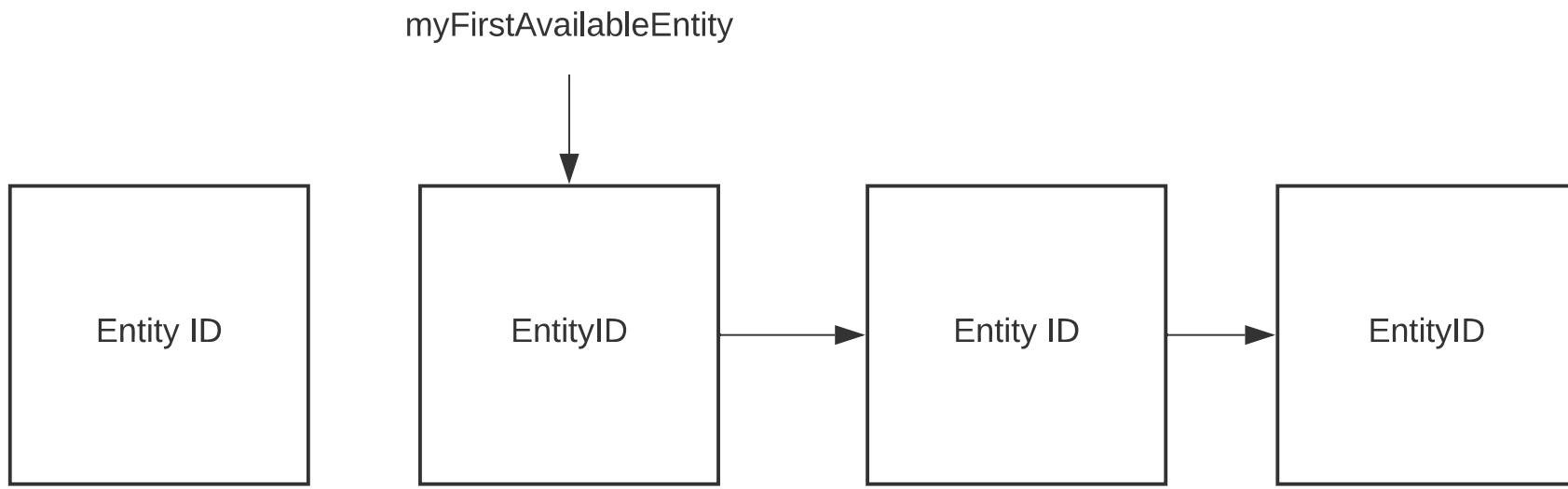
```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

GETENTITY()

myFirstAvailableEntity



GETENTITY()



GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
16 Entity EntityManager::GetEntity(Entity id)
17 {
18     if (id >= MAX_ENTITIES)
19         return Entity();
20
21     Entity entity = myAvailableEntitiesLL[id];
22
23     if (entity == myFirstAvailableEntity)
24         myFirstAvailableEntity = myAvailableEntitiesLL[entity];
25
26     myAvailableEntitiesLL[id] = myAvailableEntitiesLL[entity];
27
28     return entity;
29 }
```

GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

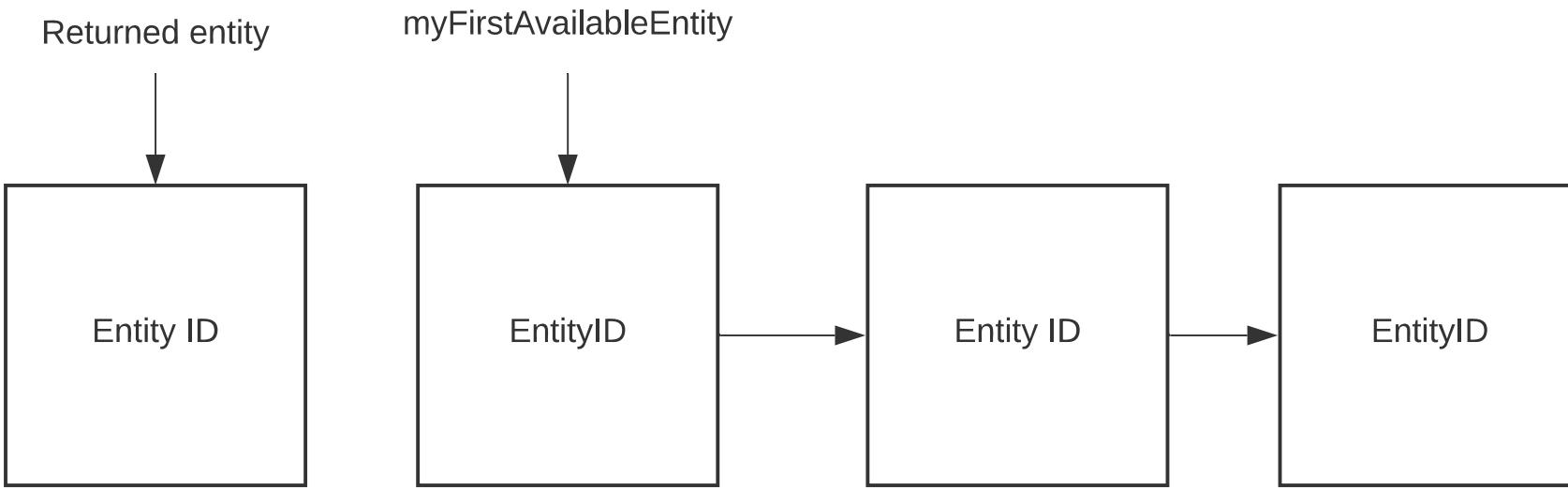
GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

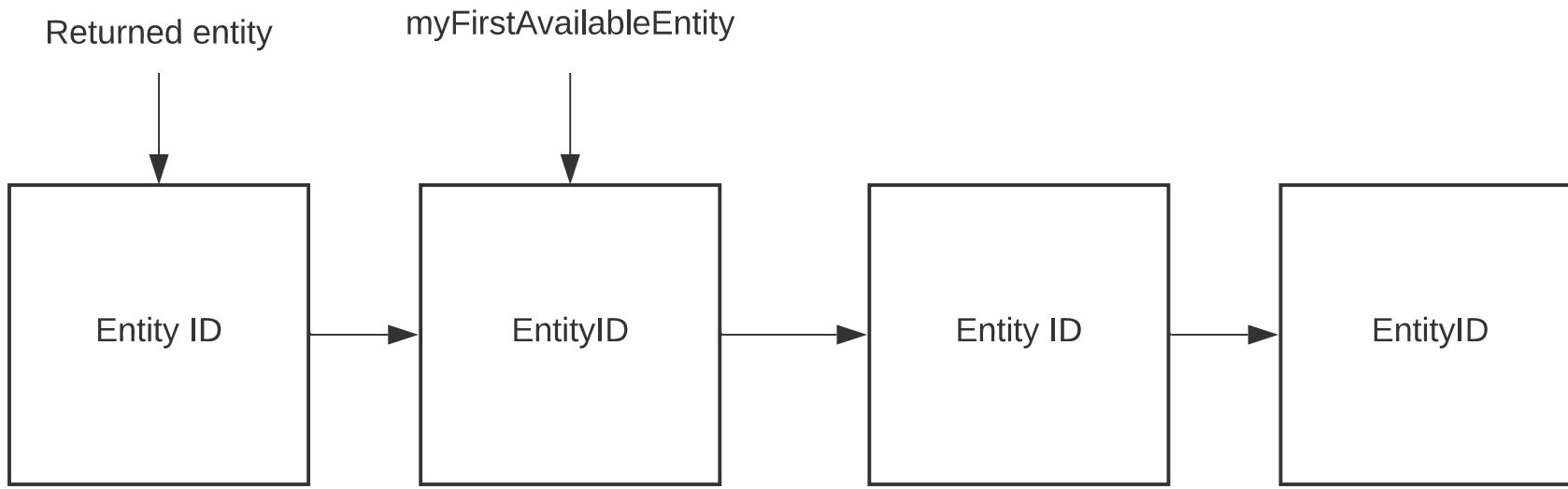
GETENTITY()

```
1 #include "EntityManager.h"
2
3 #include
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RETURNENTITY()

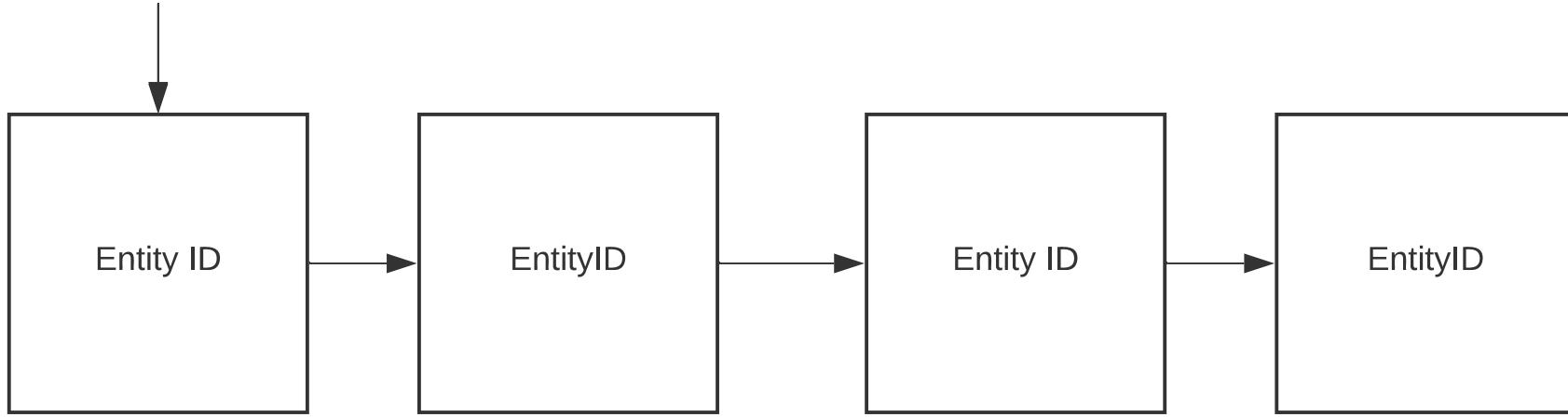


RETURNENTITY()



RETURNENTITY()

myFirstAvailableEntity



RETURNTIVITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
314
315
316
317
318
319
319
320
321
321
322
323
324
325
326
327
327
328
329
329
330
331
331
332
333
334
335
336
337
337
338
339
339
340
341
341
342
343
344
345
345
346
347
347
348
349
349
350
351
351
352
353
353
354
355
355
356
357
357
358
359
359
360
361
361
362
363
363
364
365
365
366
367
367
368
369
369
370
371
371
372
373
373
374
375
375
376
377
377
378
379
379
380
381
381
382
383
383
384
385
385
386
387
387
388
389
389
390
391
391
392
393
393
394
395
395
396
397
397
398
399
399
400
401
401
402
403
403
404
405
405
406
407
407
408
409
409
410
411
411
412
413
413
414
415
415
416
417
417
418
419
419
420
421
421
422
423
423
424
425
425
426
427
427
428
429
429
430
431
431
432
433
433
434
435
435
436
437
437
438
439
439
440
441
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
451
452
453
453
454
455
455
456
457
457
458
459
459
460
461
461
462
463
463
464
465
465
466
467
467
468
469
469
470
471
471
472
473
473
474
475
475
476
477
477
478
479
479
480
481
481
482
483
483
484
485
485
486
487
487
488
489
489
490
491
491
492
493
493
494
495
495
496
497
497
498
499
499
500
501
501
502
503
503
504
505
505
506
507
507
508
509
509
510
511
511
512
513
513
514
515
515
516
517
517
518
519
519
520
521
521
522
523
523
524
525
525
526
527
527
528
529
529
530
531
531
532
533
533
534
535
535
536
537
537
538
539
539
540
541
541
542
543
543
544
545
545
546
547
547
548
549
549
550
551
551
552
553
553
554
555
555
556
557
557
558
559
559
560
561
561
562
563
563
564
565
565
566
567
567
568
569
569
570
571
571
572
573
573
574
575
575
576
577
577
578
579
579
580
581
581
582
583
583
584
585
585
586
587
587
588
589
589
590
591
591
592
593
593
594
595
595
596
597
597
598
599
599
600
601
601
602
603
603
604
605
605
606
607
607
608
609
609
610
611
611
612
613
613
614
615
615
616
617
617
618
619
619
620
621
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
```

RETURNTIVITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RETURNTIVITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RETURNTIVITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RETURNTIVITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RETURNTENTITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RETURNTENTITY()

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
```

RESTERANDE FUNKTIONER

```
1 #include "EntityManager.h"
2
3 #include <assert.h>
4
5 EntityManager::EntityManager()
6     : myFirstAvailableEntity(0)
7     , myAvailableEntitiesLL{ }
8     , myOccupiedEntities{ }
9 {
10    for (Entity e = 0U; e < MAX_ENTITIES; ++e)
11    {
12        myAvailableEntitiesLL[e] = e + 1;
13    }
14 }
15
16
```

COMPONENT



COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5 /* Constructors, Destructor, Assignment Operators */
6 ComponentArray() = delete;
7 ComponentArray(EntityManager* entityManager);
8 ~ComponentArray() = default;
9
10 ComponentArray(const ComponentArray&) = delete;
11 ComponentArray(ComponentArray&&) = delete;
12 ComponentArray& operator=(const ComponentArray&) = delete;
13 ComponentArray& operator=(ComponentArray&&) = delete;
14
15 /* Public Interface */
16 local HasComponents<Entity> component;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponentAt(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTARRAY.HPP

```
1 template <class ComponentType>
2 class ComponentArray
3 {
4 public:
5     /* Constructors, Destructor, Assignment Operators */
6     ComponentArray() = delete;
7     ComponentArray(EntityManager* entityManager);
8     ~ComponentArray() = default;
9
10    ComponentArray(const ComponentArray&) = delete;
11    ComponentArray(ComponentArray&&) = delete;
12    ComponentArray& operator=(const ComponentArray&) = delete;
13    ComponentArray& operator=(ComponentArray&&) = delete;
14
15    /* Public Interface */
16    [[nodiscard]] ComponentType* getComponent(int index) const;
```

COMPONENTS.HPP

```
1 struct TransformComponent
2 {
3     v2f pos;
4 };
5
6 struct SpriteComponent
7 {
8     Texture myTexture;
9     v2 mySize;
10    Color myColor;
11};
```

COMPONENTS.HPP

```
1 struct TransformComponent
2 {
3     v2f pos;
4 };
5
6 struct SpriteComponent
7 {
8     Texture myTexture;
9     v2 mySize;
10    Color myColor;
11};
```

SYSTEM



SPRITESYSTEM.H

```
1 namespace Systems
2 {
3     void Sprite(
4         EntityManager* em,
5         ComponentArray* trs,
6         ComponentArray* spr);
7 }
```

SPRITESYSTEM.H

```
1 namespace Systems
2 {
3     void Sprite(
4         EntityManager* em,
5         ComponentArray* trs,
6         ComponentArray* spr);
7 }
```

SPRITESYSTEM.CPP

```
1 void Sprite(
2     EntityManager* em,
3     ComponentArray* trs,
4     ComponentArray* spr)
5 {
6     TransformComponent* trsList = trs->GetComponents();
7     SpriteComponent* sprList = spr->GetComponents();
8
9     for (Entity e = 0U; e < MAX_ENTITIES; ++e)
10    {
11        if (!em->Validate(e)
12            || !trs->HasComponent(e)
13            || !spr->HasComponent(e))
14            continue;
15
16        // More code here
17    }
18}
```

SPRITESYSTEM.CPP

```
1 void Sprite(
2     EntityManager* em,
3     ComponentArray* trs,
4     ComponentArray* spr)
5 {
6     TransformComponent* trsList = trs->GetComponents();
7     SpriteComponent* sprList = spr->GetComponents();
8
9     for (Entity e = 0U; e < MAX_ENTITIES; ++e)
10    {
11        if (!em->Validate(e)
12            || !trs->HasComponent(e)
13            || !spr->HasComponent(e))
14            continue;
15
16        // More code here
17    }
18}
```

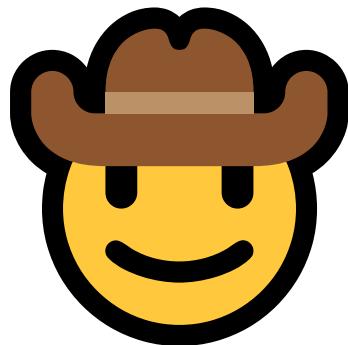
SPRITESYSTEM.CPP

```
1 void Sprite(
2     EntityManager* em,
3     ComponentArray* trs,
4     ComponentArray* spr)
5 {
6     TransformComponent* trsList = trs->GetComponents();
7     SpriteComponent* sprList = spr->GetComponents();
8
9     for (Entity e = 0U; e < MAX_ENTITIES; ++e)
10    {
11        if (!em->Validate(e)
12            || !trs->HasComponent(e)
13            || !spr->HasComponent(e))
14            continue;
15
16        // More code here
17    }
18}
```

SPRITESYSTEM.CPP

```
1 void Sprite(
2     EntityManager* em,
3     ComponentArray* trs,
4     ComponentArray* spr)
5 {
6     TransformComponent* trsList = trs->GetComponents();
7     SpriteComponent* sprList = spr->GetComponents();
8
9     for (Entity e = 0U; e < MAX_ENTITIES; ++e)
10    {
11        if (!em->Validate(e)
12            || !trs->HasComponent(e)
13            || !spr->HasComponent(e))
14            continue;
15
16        // Do stuff
17    }
18}
```

ANVÄNDNINGSEXEMPEL



```
1 struct GameState
2 {
3     EntityManager entityManager;
4
5     ComponentArray transforms = &entityManager;
6     ComponentArray sprites = &entityManager;
7 } gameState{ };
8
9 Entity testObj = INVALID_ENTITY;
```

```
1 struct GameState
2 {
3     EntityManager entityManager;
4
5     ComponentArray transforms = &entityManager;
6     ComponentArray sprites = &entityManager;
7 } gameState{ };
8
9 Entity testObj = INVALID_ENTITY;
```

```
1 struct GameState
2 {
3     EntityManager entityManager;
4
5     ComponentArray transforms = &entityManager;
6     ComponentArray sprites = &entityManager;
7 } gameState{ };
8
9 Entity testObj = INVALID_ENTITY;
```

```
1 struct GameState
2 {
3     EntityManager entityManager;
4
5     ComponentArray transforms = &entityManager;
6     ComponentArray sprites = &entityManager;
7 } gameState{ };
8
9 Entity testObj = INVALID_ENTITY;
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5     float factor = abs(sinf(totalTime));
6
7     gameState.transforms.GetComponent(testObj).pos =
8         { 400.0f + 300.0f * factor, 300.f };
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    gameState.sprites.GetComponent(testObj).myColor =
12        { color, 0, uint8_t(255 - color) };
13
14    Systems::Sprite(
15        &gameState.entityManager,
```

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5     float factor = abs(sinf(totalTime));
6
7     gameState.transforms.GetComponent(testObj).pos =
8         { 400.0f + 300.0f * factor, 300.f };
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    gameState.sprites.GetComponent(testObj).myColor =
12        { color, 0, uint8_t(255 - color) };
13
14    Systems::Sprite(
15        &gameState.entityManager,
16        &gameState.sprites
```

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5     float factor = abs(sinf(totalTime));
6
7     gameState.transforms.GetComponent(testObj).pos =
8         { 400.0f + 300.0f * factor, 300.f };
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    gameState.sprites.GetComponent(testObj).myColor =
12        { color, 0, uint8_t(255 - color) };
13
14    Systems::Sprite(
15        &gameState.entityManager,
16        &gameState.sprites
```

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5     float factor = abs(sinf(totalTime));
6
7     gameState.transforms.GetComponent(testObj).pos =
8         { 400.0f + 300.0f * factor, 300.f };
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    gameState.sprites.GetComponent(testObj).myColor =
12        { color, 0, uint8_t(255 - color) };
13
14    Systems::Sprite(
15        &gameState.entityManager,
16        &gameState.sprites
```

```
1 void Update(float dt)
2 {
3     static float totalTime = 0.0f;
4     totalTime += dt / 2.0f;
5     float factor = abs(sinf(totalTime));
6
7     gameState.transforms.GetComponent(testObj).pos =
8         { 400.0f + 300.0f * factor, 300.f };
9
10    uint8_t color = uint8_t(factor * 255.0f);
11    gameState.sprites.GetComponent(testObj).myColor =
12        { color, 0, uint8_t(255 - color) };
13
14    Systems::Sprite(
15        &gameState.entityManager,
```

RESULTAT 😎

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = new GameObject;
4     testObj->SetPosition({ 300, 300 });
5     SpriteComponent* spr = testObj->AddComponent<SpriteComponent>
6     spr->SetSpritePath("../Assets/dawdwdawda.png");
7 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = new GameObject;
4     testObj->SetPosition({ 300, 300 });
5     SpriteComponent* spr = testObj->AddComponent<SpriteComponent>
6     spr->SetSpritePath("../Assets/dawdwdawda.png");
7 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = new GameObject;
4     testObj->SetPosition({ 300, 300 });
5     SpriteComponent* spr = testObj->AddComponent<SpriteComponent>
6     spr->SetSpritePath("../Assets/dawdwdawda.png");
7 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = new GameObject;
4     testObj->SetPosition({ 300, 300 });
5     SpriteComponent* spr = testObj->AddComponent<SpriteComponent>
6     spr->SetSpritePath("../Assets/dawdwdawda.png");
7 }
```

```
1 void Init()
2 {
3     testObj = gameState.entityManager.GetEntity();
4
5     gameState.transforms.AddComponent(testObj).pos =
6         { 300.0f, 300.0f };
7
8     SpriteComponent& sprComp =
9         gameState.sprites.AddComponent(testObj);
10    sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
11 }
```

```
1 void Init()
2 {
3     testObj = new GameObject;
4     testObj->SetPosition({ 300, 300 });
5     SpriteComponent* spr = testObj->AddComponent<SpriteComponent>
6     spr->SetSpritePath("../Assets/dawdwdawda.png");
7 }
```

```
1 void Update(float dt)
2 {
3     gameState.transforms.GetComponent(testObj).pos =
4         { 400.0f + 300.0f * factor, 300.f };
5
6     uint8_t color = uint8_t(factor * 255.0f);
7     gameState.sprites.GetComponent(testObj).myColor =
8         { color, 0, uint8_t(255 - color) };
9 }
```

```
1 void Update(float dt)
2 {
3     testObj->SetPosition(
4         { 400.0f + 300.0f * factor, 300.f });
5
6     uint8_t color = uint8_t(factor * 255.0f);
7     testObj->GetComponent<SpriteComponent>()
8         ->SetColor({ color, 0, uint8_t(255 - color) });
9 }
```

```
1 void Update(float dt)
2 {
3     gameState.transforms.GetComponent(testObj).pos =
4         { 400.0f + 300.0f * factor, 300.f };
5
6     uint8_t color = uint8_t(factor * 255.0f);
7     gameState.sprites.GetComponent(testObj).myColor =
8         { color, 0, uint8_t(255 - color) };
9 }
```

```
1 void Update(float dt)
2 {
3     testObj->SetPosition(
4         { 400.0f + 300.0f * factor, 300.f });
5
6     uint8_t color = uint8_t(factor * 255.0f);
7     testObj->GetComponent<SpriteComponent>()
8         ->SetColor({ color, 0, uint8_t(255 - color) });
9 }
```

```
1 void Update(float dt)
2 {
3     gameState.transforms.GetComponent(testObj).pos =
4         { 400.0f + 300.0f * factor, 300.f };
5
6     uint8_t color = uint8_t(factor * 255.0f);
7     gameState.sprites.GetComponent(testObj).myColor =
8         { color, 0, uint8_t(255 - color) };
9 }
```

```
1 void Update(float dt)
2 {
3     testObj->SetPosition(
4         { 400.0f + 300.0f * factor, 300.f });
5
6     uint8_t color = uint8_t(factor * 255.0f);
7     testObj->GetComponent<SpriteComponent>()
8         ->SetColor({ color, 0, uint8_t(255 - color) });
9 }
```

NÅGRA SAKER ATT HA I ÅTANKE

NÅGRA SAKER ATT HA I ÅTANKE

- Det är inte antingen eller, hybrider finns och är vanliga för spelmotorer.

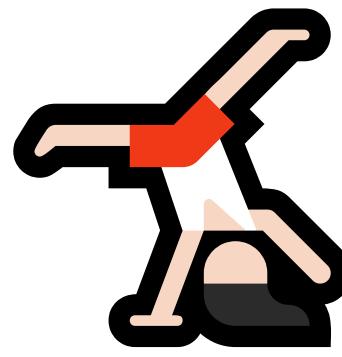
NÅGRA SAKER ATT HA I ÅTANKE

- Det är inte antingen eller, hybrider finns och är vanliga för spelmotorer.
- Viktigt att bibehålla användarvänligheten för era kollegor.

NÅGRA SAKER ATT HA I ÅTANKE

- Det är inte antingen eller, hybrider finns och är vanliga för spelmotorer.
- Viktigt att bibehålla användarvänligheten för era kollegor.
- Det finns många olika lösningar; ingen är one-size-fits-all.

RESURSER



CPPCON 2014: MIKE ACTON "DATA-ORIENTED DESIGN AND C++"

CppCon 2014: Mike Acton "Data-Oriented Design and C++"



A SIMPLE ENTITY COMPONENT SYSTEM (ECS) [C++]-

AUSTIN MORLAN

ENTT

FRÅGOR

