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

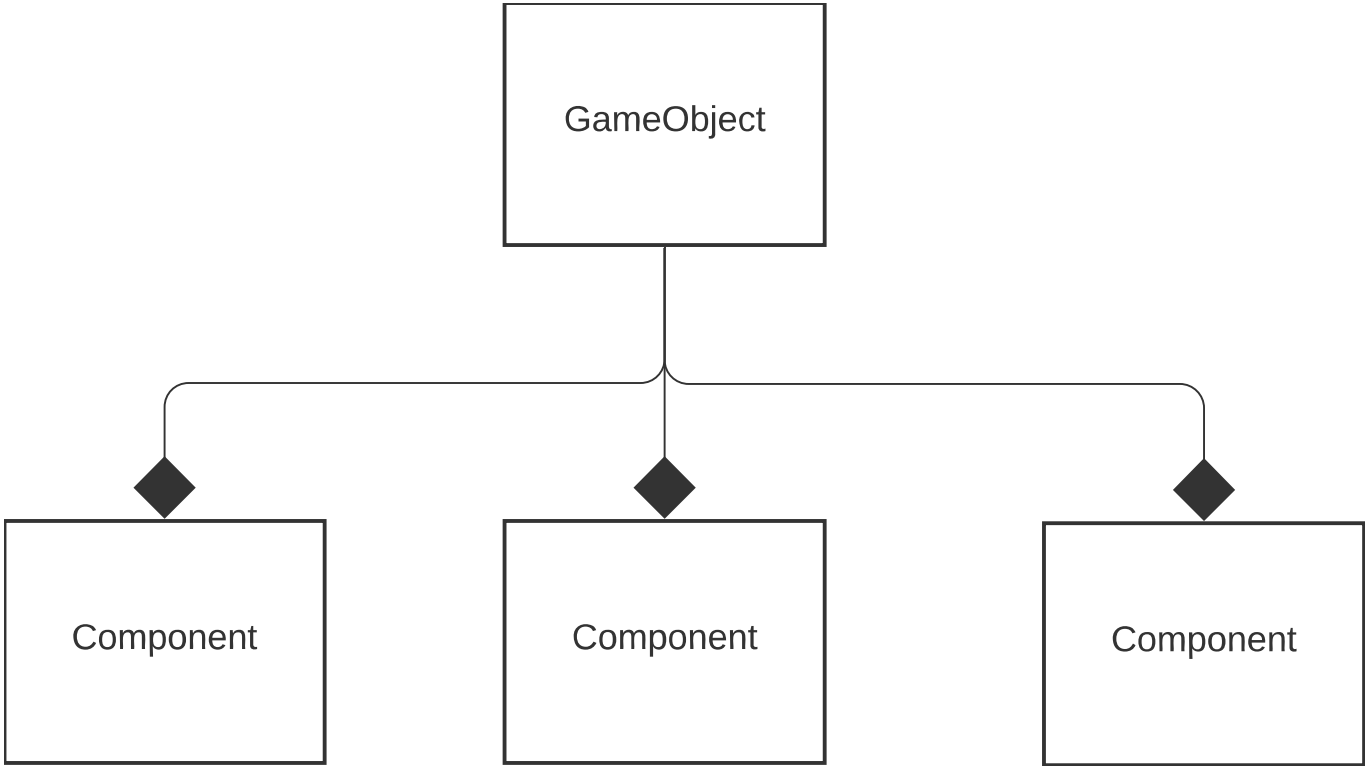
VAD ÄR ETT KOMPONENTSYSTEM?

- 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

```
class PlayerCharacter :
public Transform, public Sprite,
public PhysicsBody, public BoxCollider
{
public:
  PlayerCharacter();
  ~PlayerCharacter();

  void Init();
  void Update(float dt);
  void Render();

  // ...

private:
```

COMPOSITION OVER INHERITANCE

```
class PlayerCharacter :
public Transform, public Sprite,
public PhysicsBody, public BoxCollider
{
public:
  PlayerCharacter();
  ~PlayerCharacter();

  void Init();
  void Update(float dt);
  void Render();

  // ...

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.

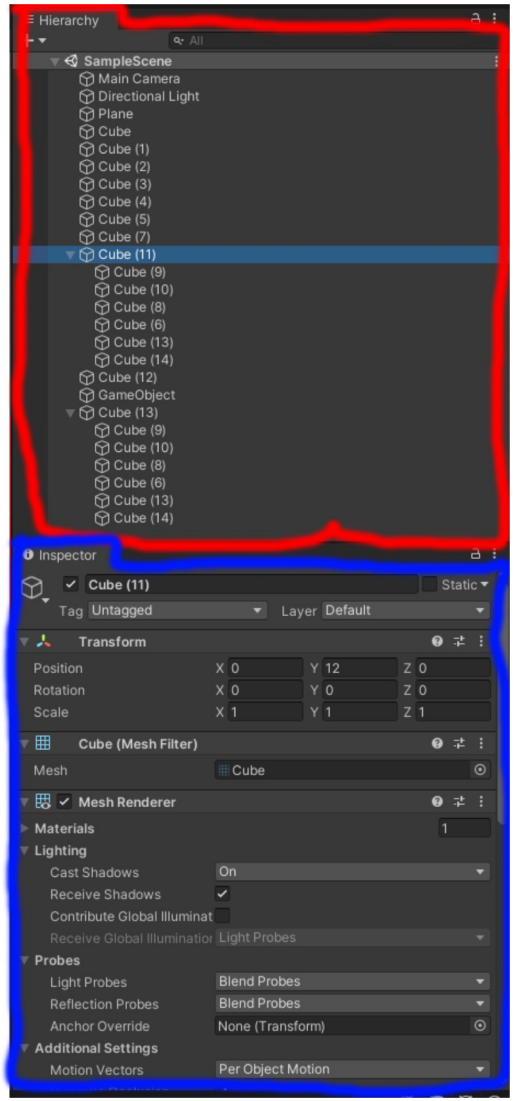
DYNAMISK KOMPOSITION, ISTÄLLET FÖR STATISK.

- 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

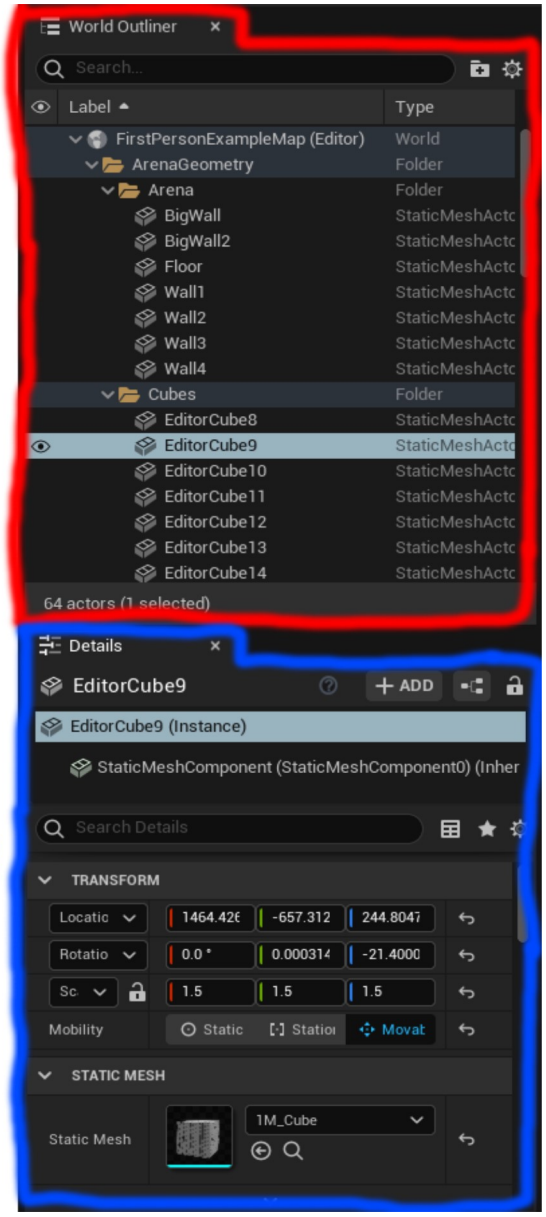
Components



EXEMPEL

Actors

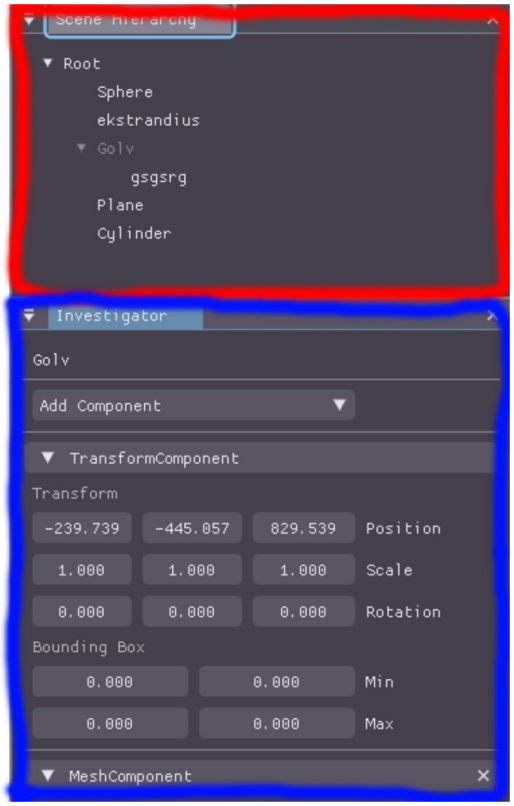
Components



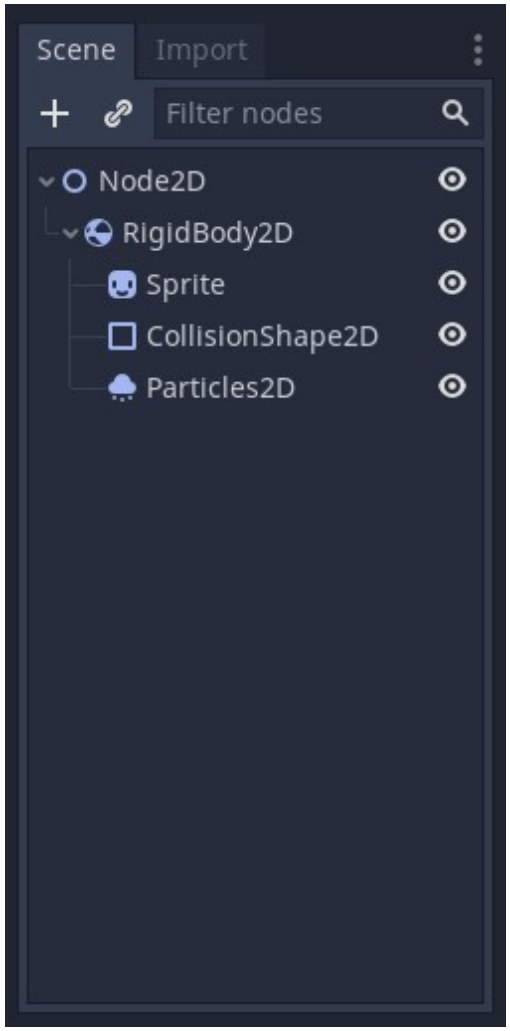
EXEMPEL

Entities

Components



EXEMPEL


Scene
Import
Filter nodes
Node2D
RigidBody2D
Sprite
CollisionShape2D
Particles2D

VAD ÄR INTE ETT KOMPONENTSYSTEM?

NÅGRA EXEMPEL:

ARV

```
class PlayerCharacter :
public Transform, public Sprite,
public PhysicsBody, public BoxCollider
{
public:
  PlayerCharacter();
  ~PlayerCharacter();

  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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
  {
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

GAMEOBJECT.CPP

```cpp
#include "GameObject.h"

#include "Component.hpp"

std::vector GameObject::ourGameObjects;

GameObject::GameObject()
  : myComponents{}
{
  ourGameObjects.push_back(this);
}

GameObject::~GameObject()
{
  for (Component* comp : myComponents)
```

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 };
```

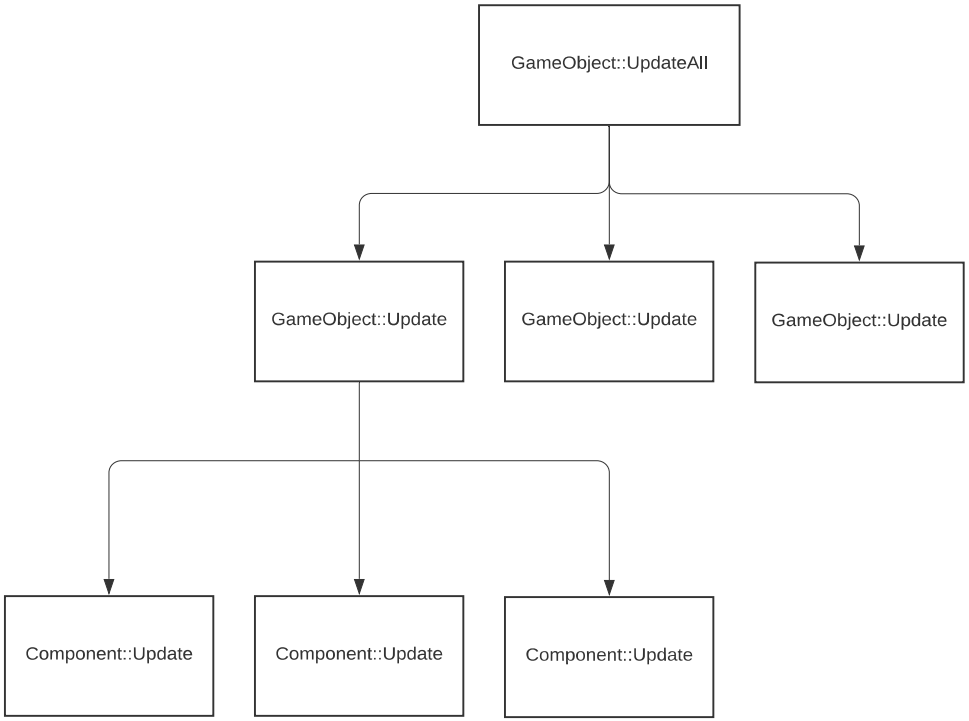
COMPONENT.HPP

```
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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

SPRITECOMPONENT.CPP

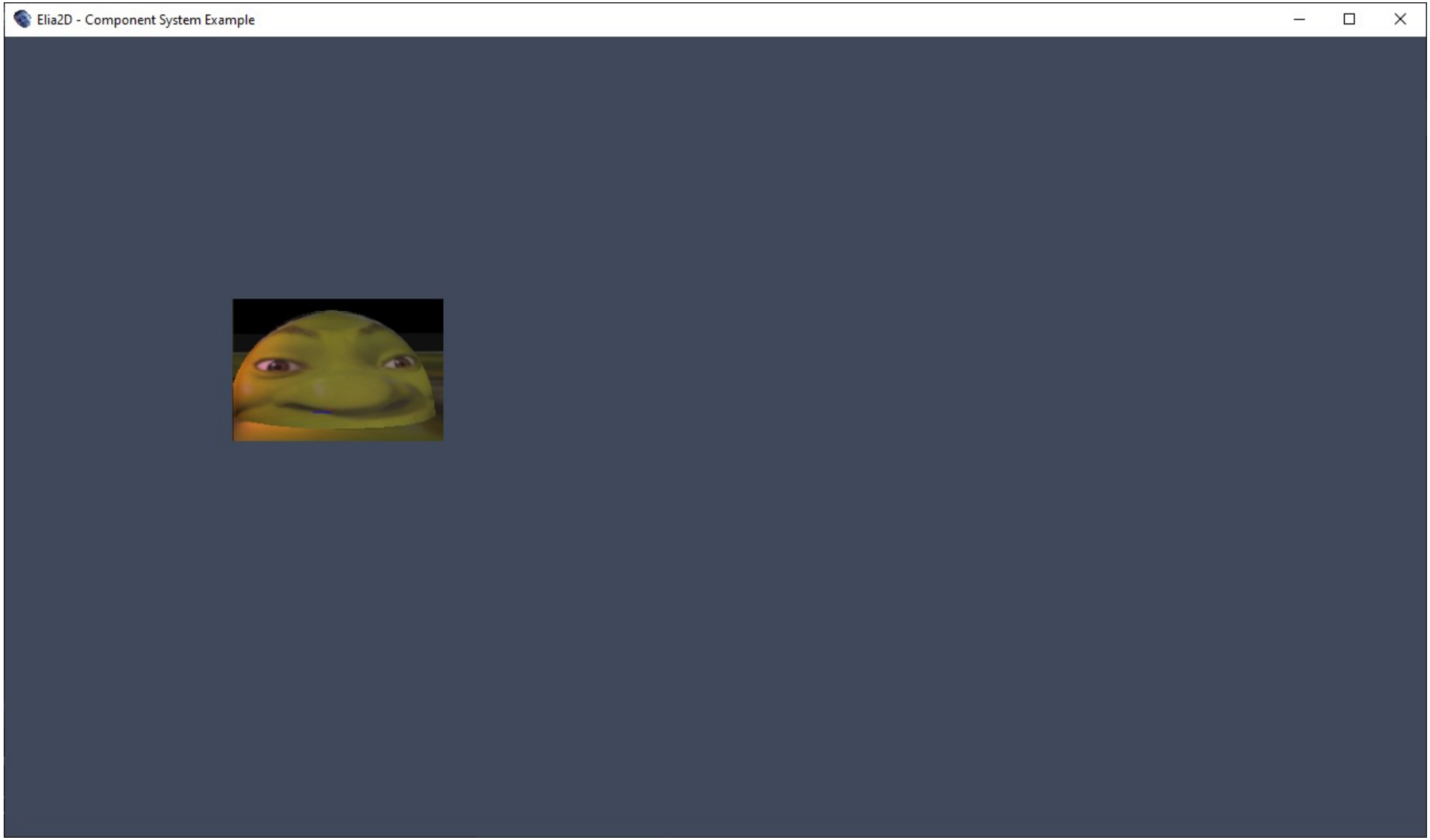
```cpp
#include "SpriteComponent.h"

#include "GameObject.h"

SpriteComponent::SpriteComponent()
  : myTexture(0U)
  , mySize({ 0, 0 })
  , myColor({ 255, 255, 255 })
  , myTexturePath("")
{}

SpriteComponent::~SpriteComponent()
{
  if (myTexture)
  {
```

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

- Komponentsystem - 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	Key	Key	Key	Key	Key	Key	Key
Value	Value	Value	Value	Value	Value	Value	Value

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);
```

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);
```

ENTITYMANAGER.H

```
class EntityManager
{
public:
  /* Constructors, Destructor, Assignment Operators */
  EntityManager();
  ~EntityManager() = default;

  EntityManager(const EntityManager& other);
  EntityManager(EntityManager&& other) noexcept;
  EntityManager& operator=(const EntityManager& other);
  EntityManager& operator=(EntityManager&& other);

  /* Public Interface */
  Entity GetEntity();
  void ReturnEntity(Entity e);
```

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);
```

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);
```

ENTITYMANAGER.H

```
class EntityManager
{
public:
  /* Constructors, Destructor, Assignment Operators */
  EntityManager();
  ~EntityManager() = default;

  EntityManager(const EntityManager& other);
  EntityManager(EntityManager&& other) noexcept;
  EntityManager& operator=(const EntityManager& other);
  EntityManager& operator=(EntityManager&& other);

  /* Public Interface */
  Entity GetEntity();
  void ReturnEntity(Entity e);
```

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
```

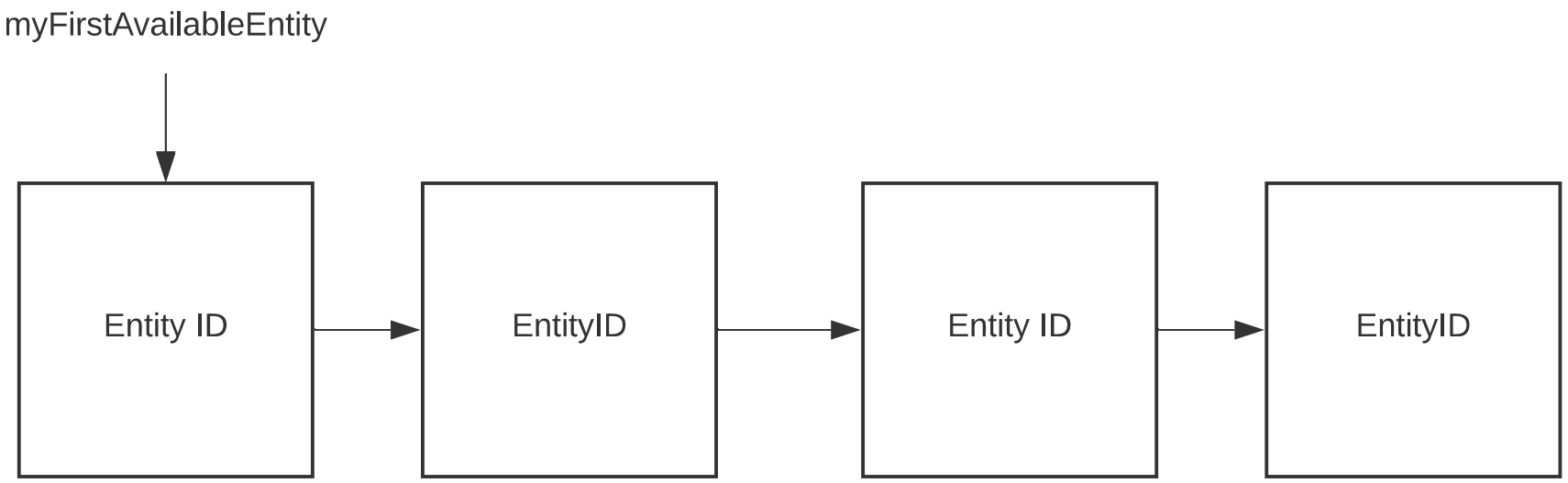
INITIALISERA LÄNKADE LISTAN

```
 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
```

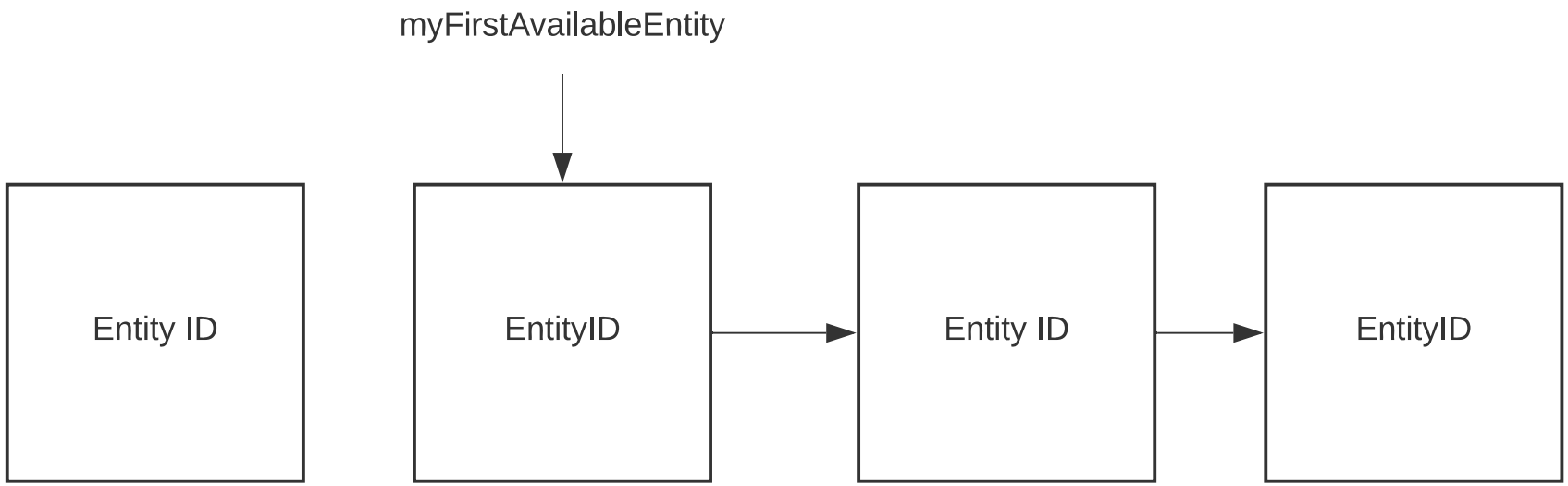
INITIALISERA LÄNKADE LISTAN

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

```

GETENTITY()


myFirstAvailableEntity
Entity ID
EntityID
Entity ID
EntityID

GETENTITY()


myFirstAvailableEntity
Entity ID
EntityID
Entity ID
EntityID

GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

```

GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

Entity Manager::GetEntity(const EntityManager& other)
```

GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

```

GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

Entity Manager::GetEntity(const EntityManager& other)
```

GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

```

GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

```

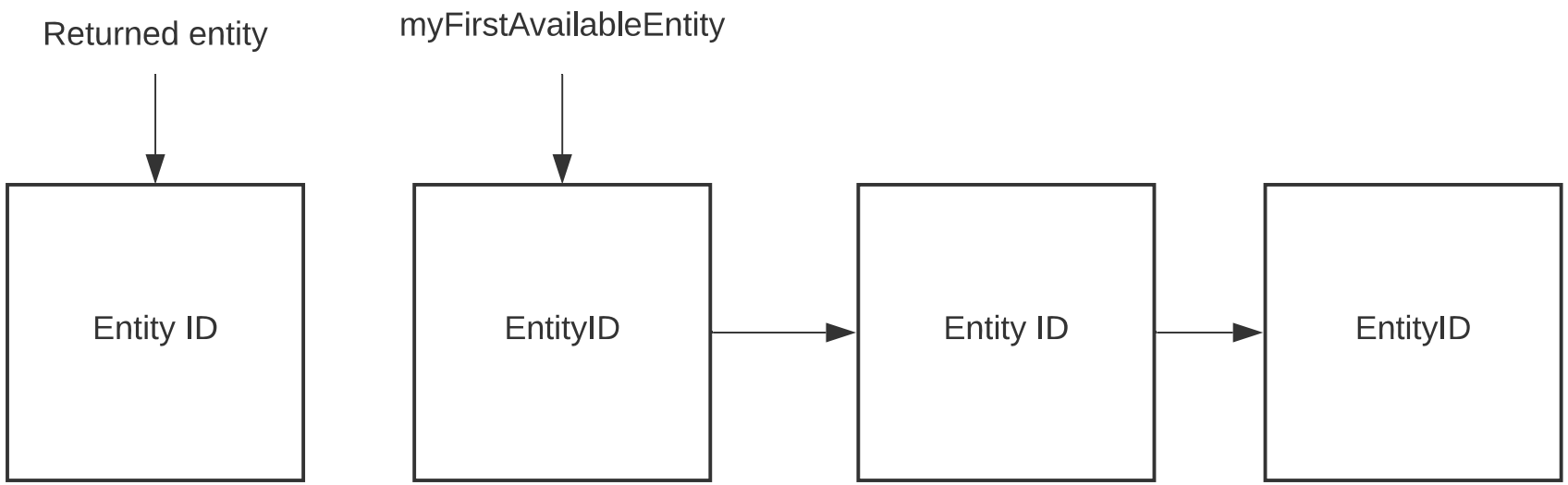
GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager&other)
```

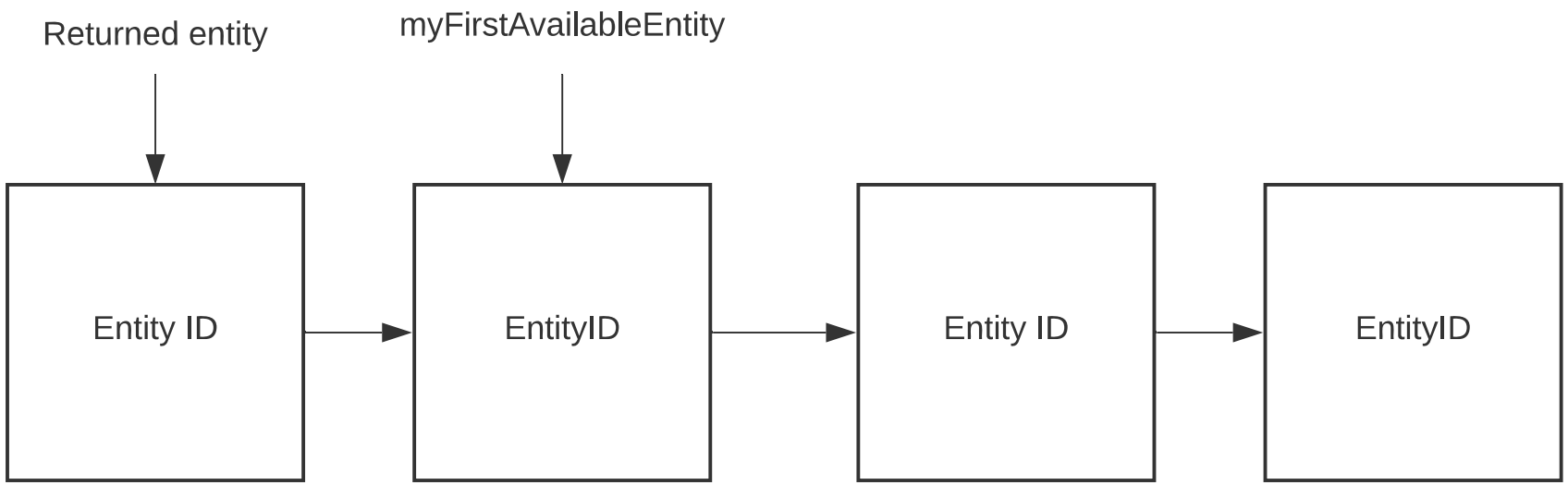
GETENTITY()

```cpp
#include "EntityManager.h"

#include

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

```

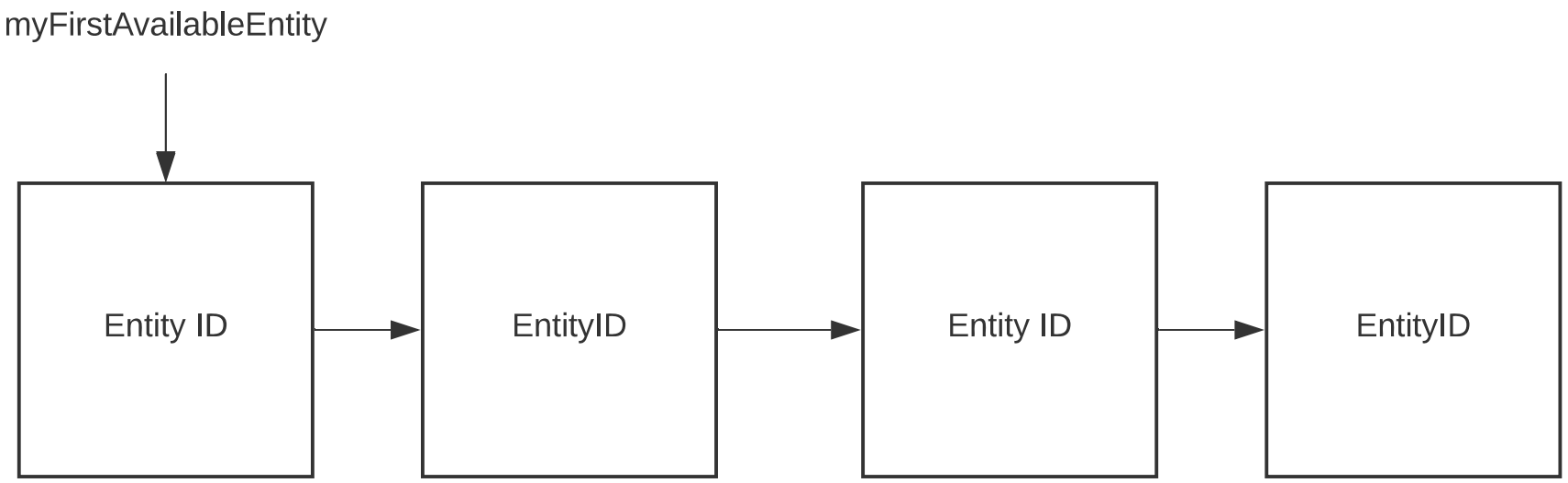
RETURNENTITY()



RETURNENTITY()



RETURNENTITY()



RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RETURNENTITY()

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

RESTERANDE FUNKTIONER

```cpp
#include "EntityManager.h"

#include <assert.h>

EntityManager::EntityManager()
  : myFirstAvailableEntity(0)
  , myAvailableEntitiesLL{}
  , myOccupiedEntities{}
{
  for (Entity e = 0U; e < MAX_ENTITIES; ++e)
  {
    myAvailableEntitiesLL[e] = e + 1;
  }
}

EntityManager::EntityManager(const EntityManager& other)
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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   bool HasComponent(Entity e) 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   bool HasComponent(Entity e) 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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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 */
```

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
```

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
```

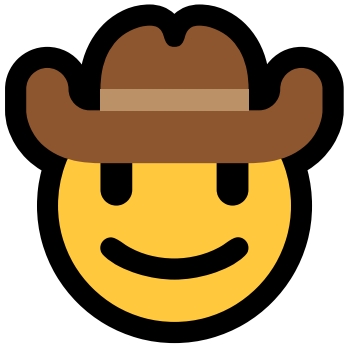
SPRITESYSTEM.CPP

```
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
```

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
```

ANVÄNDNINGSEXEMPEL



```
struct GameState
{
  EntityManager entityManager;

  ComponentArray transforms = &entityManager;
  ComponentArray sprites = &entityManager;
} gameState{};

Entity testObj = INVALID_ENTITY;
```

```
struct GameState
{
  EntityManager entityManager;

  ComponentArray transforms = &entityManager;
  ComponentArray sprites = &entityManager;
} gameState{};

Entity testObj = INVALID_ENTITY;
```

```
struct GameState
{
  EntityManager entityManager;

  ComponentArray transforms = &entityManager;
  ComponentArray sprites = &entityManager;
} gameState{};

Entity testObj = INVALID_ENTITY;
```

```
struct GameState
{
  EntityManager entityManager;

  ComponentArray transforms = &entityManager;
  ComponentArray sprites = &entityManager;
} gameState{};

Entity testObj = INVALID_ENTITY;
```

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

```
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 }
```

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

```
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,
```

```
void Update(float dt)
{
  static float totalTime = 0.0f;
  totalTime += dt / 2.0f;
  float factor = abs(sinf(totalTime));

  gameState.transforms.GetComponent(testObj).pos =
    { 400.0f + 300.0f * factor, 300.f };

  uint8_t color = uint8_t(factor * 255.0f);
  gameState.sprites.GetComponent(testObj).myColor =
    { color, 0, uint8_t(255 - color) };

  Systems::Sprite(
    &gameState.entityManager,
```

```
void Update(float dt)
{
  static float totalTime = 0.0f;
  totalTime += dt / 2.0f;
  float factor = abs(sinf(totalTime));

  gameState.transforms.GetComponent(testObj).pos =
    { 400.0f + 300.0f * factor, 300.f };

  uint8_t color = uint8_t(factor * 255.0f);
  gameState.sprites.GetComponent(testObj).myColor =
    { color, 0, uint8_t(255 - color) };

  Systems::Sprite(
    &gameState.entityManager,
```

```
void Update(float dt)
{
  static float totalTime = 0.0f;
  totalTime += dt / 2.0f;
  float factor = abs(sinf(totalTime));

  gameState.transforms.GetComponent(testObj).pos =
    { 400.0f + 300.0f * factor, 300.f };

  uint8_t color = uint8_t(factor * 255.0f);
  gameState.sprites.GetComponent(testObj).myColor =
    { color, 0, uint8_t(255 - color) };

  Systems::Sprite(
    &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,
```

RESULTAT 😎

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

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

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

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

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

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

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

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

```
void Init()
{
  testObj = gameState.entityManager.GetEntity();

  gameState.transforms.AddComponent(testObj).pos =
    { 300.0f, 300.0f };

  SpriteComponent& sprComp =
    gameState.sprites.AddComponent(testObj);
  sprComp.myTexture = CreateTexture("../Assets/dawdwdawda.png"
}
```

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

```
void Update(float dt)
{
  gameState.transforms.GetComponent(testObj).pos =
    { 400.0f + 300.0f * factor, 300.f };

  uint8_t color = uint8_t(factor * 255.0f);
  gameState.sprites.GetComponent(testObj).myColor =
    { color, 0, uint8_t(255 - color) };
}
```

```
void Update(float dt)
{
  testObj->SetPosition(
    { 400.0f + 300.0f * factor, 300.f });

  uint8_t color = uint8_t(factor * 255.0f);
  testObj->GetComponent<SpriteComponent>()
    ->SetColor({ color, 0, uint8_t(255 - color) });
}
```

```
void Update(float dt)
{
  gameState.transforms.GetComponent(testObj).pos =
    { 400.0f + 300.0f * factor, 300.f };

  uint8_t color = uint8_t(factor * 255.0f);
  gameState.sprites.GetComponent(testObj).myColor =
    { color, 0, uint8_t(255 - color) };
}
```

```
void Update(float dt)
{
  testObj->SetPosition(
    { 400.0f + 300.0f * factor, 300.f });

  uint8_t color = uint8_t(factor * 255.0f);
  testObj->GetComponent<SpriteComponent>()
    ->SetColor({ color, 0, uint8_t(255 - color) });
}
```

```
void Update(float dt)
{
  gameState.transforms.GetComponent(testObj).pos =
    { 400.0f + 300.0f * factor, 300.f };

  uint8_t color = uint8_t(factor * 255.0f);
  gameState.sprites.GetComponent(testObj).myColor =
    { color, 0, uint8_t(255 - color) };
}
```

```
void Update(float dt)
{
  testObj->SetPosition(
    { 400.0f + 300.0f * factor, 300.f });

  uint8_t color = uint8_t(factor * 255.0f);
  testObj->GetComponent<SpriteComponent>()
    ->SetColor({ color, 0, uint8_t(255 - color) });
}
```

NÅGRA SAKER ATT HA I ÅTANKE

NÅGRA SAKER ATT HA I ÅTANKE

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

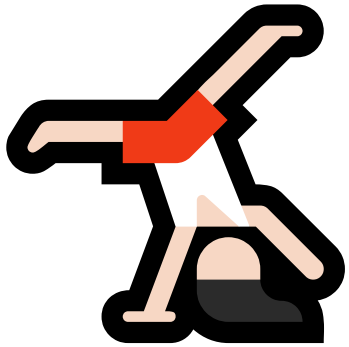
NÅGRA SAKER ATT HA I ÅTANKE

- 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++"

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

ENTT

FRÅGOR