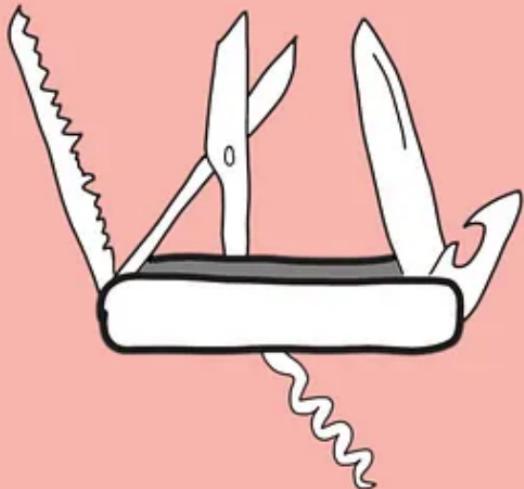


SRP

Single Responsibility Principle



VS



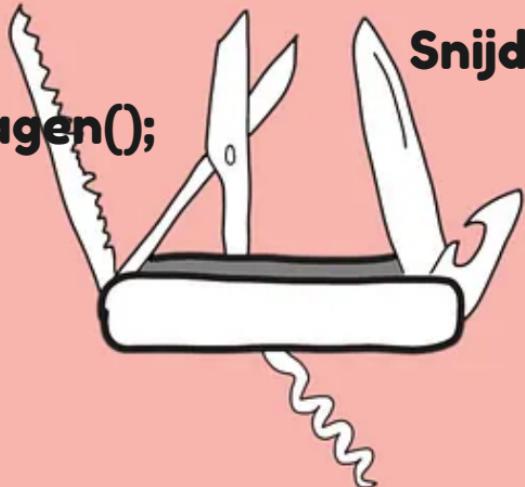
De "S" in SOLID

Het Single Responsibility Principle (SRP) is een van de SOLID-principes in objectgeoriënteerd ontwerp.

Het betekent dat een "Class" slechts één verantwoordelijkheid mag hebben voor gedrag binnen een systeem.

Dit principe bevordert modulaire en onderhoudsvriendelijke code. De code kan eenvoudiger worden gelezen, getest en gewijzigd zonder neveneffecten.

Knippen();

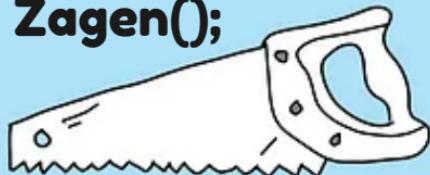


Zagen();

Snijden();

VS

Zagen();



Snijden();



Knippen();

Single Responsibility?

Welke verantwoordelijkheden heeft een smartphone?



Welke verantwoordelijkheden heeft een koffiemachine?



Welke verantwoordelijkheden heeft een brandalarm?



Tower.cs

Tower.cs > ...

```
1  using UnityEngine;          Wat zijn hier de verantwoordelijkheden?
2
3  public class Tower : MonoBehaviour
4  {
5      public int damage = 10;
6      public float attackSpeed = 1.0f;
7
8      private Enemy target;
9
10     private void Update()
11     {
12         // Zoek naar vijanden in de buurt
13         Collider[] colliders = Physics.OverlapSphere(transform.position, 5.0f);
14         foreach (Collider collider in colliders)
15         {
16             if (collider.CompareTag("Enemy"))
17             {
18                 target = collider.GetComponent<Enemy>();
19                 break;
20             }
21         }
22
23         if (target != null)
24         {
25             // Val de vijand aan
26             if (Time.time >= attackSpeed)
27             {
28                 Attack();
29                 attackSpeed = Time.time + 1.0f;
30             }
31         }
32     }
33
34     private void Attack()
35     {
36         // Voer aanval uit en verlaag de gezondheid van de vijand
37         target.TakeDamage(damage);
38     }
39 }
40
```



EnemyDetection.cs X

EnemyDetection.cs > ...

```
1  using UnityEngine;
2
3  2 references
4  public class EnemyDetection : MonoBehaviour
5  {
6
7      2 references
8      private Enemy target;
9
10     1 reference
11     public Enemy GetNearestEnemy(float detectionRange)
12     {
13
14         Collider[] colliders = Physics.OverlapSphere(transform.position, detectionRange);
15         foreach (Collider collider in colliders)
16         {
17
18             if (collider.CompareTag("Enemy"))
19             {
20                 target = collider.GetComponent<Enemy>();
21                 return target;
22             }
23         }
24         return null; // Geen vijanden gevonden
25     }
26 }
```



TowerAttack.cs X

TowerAttack.cs > ...

```
1  using UnityEngine;
2
3  0 references
4  public class TowerAttack : MonoBehaviour
5  {
6
7      1 reference
8      public int damage = 10;
9
10     2 references
11     public float attackSpeed = 1.0f;
12
13     2 references
14     private EnemyDetection enemyDetection;
15
16     3 references
17     private Enemy currentTarget;
18
19     0 references
20     private void Start()
21     {
22
23         enemyDetection = GetComponent<EnemyDetection>();
24     }
25
26     0 references
27     private void Update()
28     {
29
30         currentTarget = enemyDetection.GetNearestEnemy(5.0f);
31
32         if (currentTarget != null)
33         {
34
35             if (Time.time >= attackSpeed)
36             {
37
38                 Attack();
39                 attackSpeed = Time.time + 1.0f;
40             }
41         }
42     }
43
44     1 reference
45     private void Attack()
46     {
47
48         // Voer aanval uit en verlaag de gezondheid van de vijand
49         currentTarget.TakeDamage(damage);
50     }
51 }
```



```
nonSRP > C# Tower.cs > ...
1  using UnityEngine;
0 references
2  public class Tower : MonoBehaviour
3  {
4      1 reference
5      |  public int damage = 10;
6      |  2 references
7      |  public float attackSpeed = 1.0f;
8      |  3 references
9      |  private Enemy target;
10     |  0 references
11    >  private void Update() ...
12        1 reference
13    30  private void Attack()
14    {
15        // Voer aanval uit en verlaag de gezondheid van de vijand
16        target.TakeDamage(damage);
17    }
18        0 references
19    35  private void Upgrade(int steps)
20    {
21        // Hier komt code om mijn toren te upgraden
22    }
23        0 references
24    39  private void TowerDies()
25    {
26        // Hier komt code om mijn toren te vernietigen als de health op is
27    }
28        0 references
29    43  private void SellTower()
30    {
31        // Hier komt code om mijn toren te verkopen en resources teug te krijgen
32    }
33    }
```

Voltoet dit plan aan SRP?

✓ SCRIPTS

✓ nonSRP

C Tower.cs

✓ SRP

C EnemyDetection.cs

C KillTower.cs

C SellTower.cs

C TowerAttack.cs

C UpgradeTower.cs



SHOP

Modulair

ShootArrow()
Upgrade()
Sell()
KillTower()
Reload()



Archery Tower

ShootCannon()
Upgrade()
Sell()
KillTower()
Reload()



Cannon Tower

ShootBeam()
Upgrade()
Sell()



Magic Tower

ShootFlame()
Upgrade()
Sell()
KillTower()



Fire Tower

1

- +

1

- +

1

- +

1

- +

BUY TOWER

BUY TOWER

BUY TOWER

BUY TOWER

Onderhoudbaar



Het DRY-principe staat voor "**Don't Repeat Yourself**".

Het houdt in dat je herhaling van code moet vermijden door gemeenschappelijke logica te abstraheren en te hergebruiken.

Oftewel: **Schrijf dezelfde code niet meerdere keren!**

Organiseer code zodanig dat je deze op één plek kunt onderhouden en overal kunt gebruiken.

Dit verbetert de leesbaarheid, onderhoudbaarheid en vermindert de kans op fouten in je code.

Enkele tips

Arrays en Lists gebruiken

Functies en Methoden Gebruiken

Gebruik van Parameters

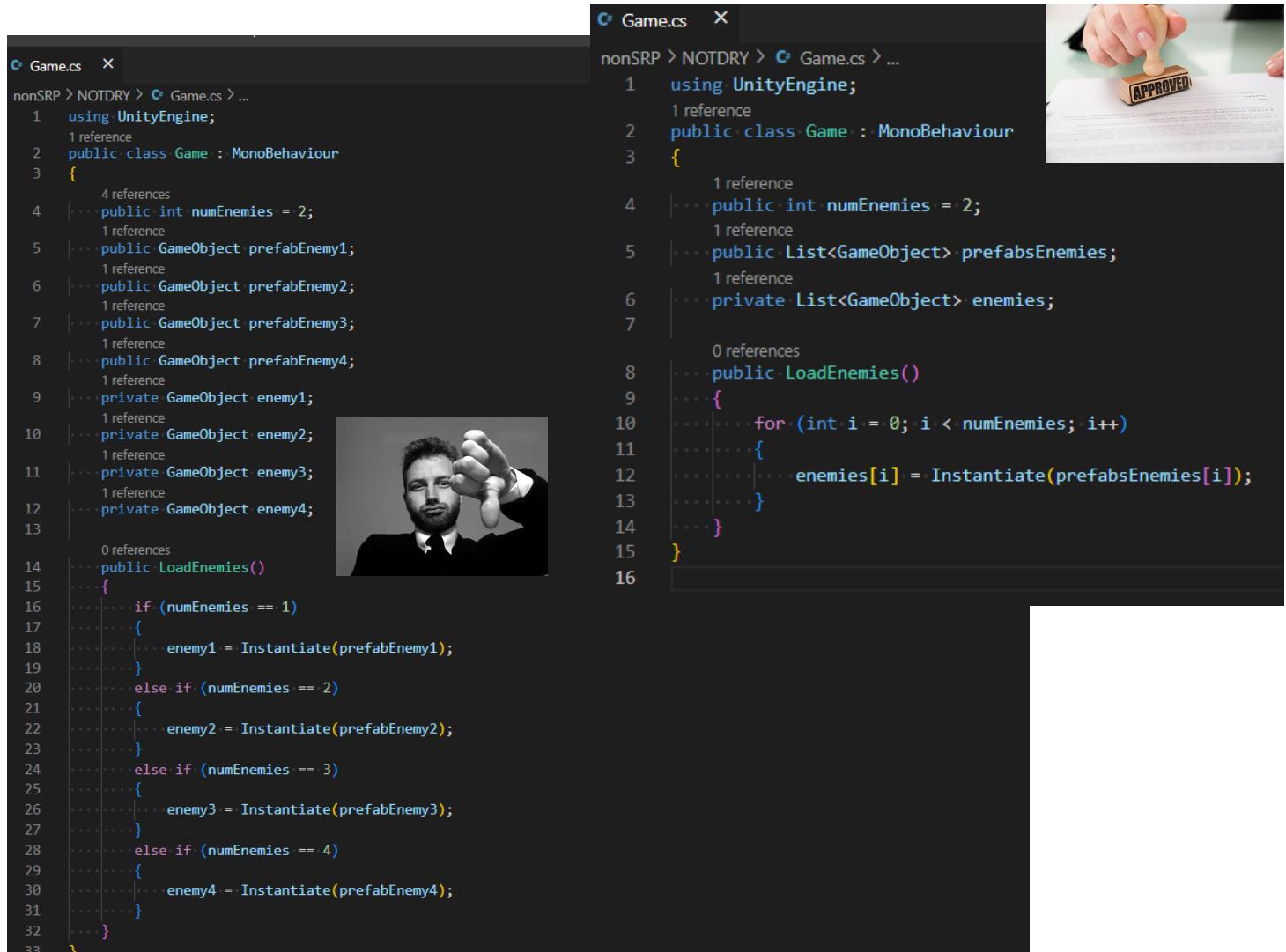
Herbruikbare Functies en Klassen

Bibliotheekfuncties Gebruiken

Inheritance & Abstraction

Arrays en Lists gebruiken:

Gebruik Arrays en of Lists als je een verzameling creeert van hetzelfde type met hetzelfde doel.



```
Game.cs  X
nonSRP > NOTDRY > Game.cs > ...
1  using UnityEngine;
1 reference
2  public class Game : MonoBehaviour
3  {
4      4 references
5      ... public int numEnemies = 2;
6      1 reference
7      ... public GameObject prefabEnemy1;
8      1 reference
9      ... public GameObject prefabEnemy2;
10     1 reference
11     ... public GameObject prefabEnemy3;
12     1 reference
13     ... public GameObject prefabEnemy4;
14
15     0 references
16     ... public LoadEnemies()
17     {
18         if (numEnemies == 1)
19         {
20             enemy1 = Instantiate(prefabEnemy1);
21         }
22         else if (numEnemies == 2)
23         {
24             enemy2 = Instantiate(prefabEnemy2);
25         }
26         else if (numEnemies == 3)
27         {
28             enemy3 = Instantiate(prefabEnemy3);
29         }
30         else if (numEnemies == 4)
31         {
32             enemy4 = Instantiate(prefabEnemy4);
33         }
34     }
35
36     1 reference
37     ... public int numEnemies = 2;
38     1 reference
39     ... public List<GameObject> prefabsEnemies;
40     1 reference
41     ... private List<GameObject> enemies;
42
43     0 references
44     ... public LoadEnemies()
45     {
46         for (int i = 0; i < numEnemies; i++)
47         {
48             enemies[i] = Instantiate(prefabsEnemies[i]);
49         }
50     }
51
52     }
53 }
```

Functies en Methoden Gebruiken:

Scheid herhaalde logica in afzonderlijke functies of methoden. Hierdoor kun je dezelfde code hergebruiken zonder duplicatie.

Gebruik van Parameters:

Gebruik functieparameters om dynamische waarden door te geven aan herbruikbare functies in plaats van dezelfde code te kopiëren en te plakken met kleine variaties.

```
private void AttackEnemy(Enemy enemy)
{
    int damageToApply = damage;

    if (enemy.isBoss)
    {
        damageToApply = damage * 2; // Dubbele schade aan bazen
    }
    else if (enemy.isMinion)
    {
        damageToApply = damage / 2; // Halve schade aan handlangers
    }

    enemy.TakeDamage(damageToApply);
}
```

Herbruikbare Functies en Klassen:

Bouw herbruikbare functies, klassen of componenten voor veelvoorkomende taken om herhaling te verminderen.

```
Projectile.cs X Tower.cs X Enemy.cs
reuseable > Projectile.cs ...
1 public class Projectile
2 {
3     public string Name { get; private set; }
4     public float Speed { get; private set; }
5     public int Damage { get; private set; }
6
7     public Projectile(string name, float speed, int damage)
8     {
9         Name = name;
10        Speed = speed;
11        Damage = damage;
12    }
13
14    public void Fire(Vector3 startPosition, Vector3 targetPosition)
15    {
16        // Implementeer de logica om het projectiel van startPosition naar targetPosition te bewegen
17        // Dit kan een lineaire beweging zijn, een boogbaan, enz., afhankelijk van het projectieltype
18        // Voer ook logica uit om schade toe te brengen aan het doelwit (bijv. een vijand)
19    }
20 }
```

```
Projectile.cs X Tower.cs X Enemy.cs
reuseable > Tower.cs ...
1 public class Tower
2 {
3     private Projectile projectile;
4
5     public Tower(string projectileName, float projectileSpeed, int projectileDamage)
6     {
7         projectile = new Projectile(projectileName, projectileSpeed, projectileDamage);
8     }
9
10    public void Attack(Enemy target)
11    {
12        // Bereken startPosition en targetPosition
13        Vector3 startPosition = transform.position;
14        Vector3 targetPosition = target.transform.position;
15
16        // Vuur het projectiel af
17        projectile.Fire(startPosition, targetPosition);
18    }
19 }
20 |
```

```
Projectile.cs X Tower.cs X Enemy.cs
reuseable > Enemy.cs ...
1
2     public class Enemy
3     {
4         private Projectile projectile;
5
6         public Enemy(string projectileName, float projectileSpeed, int projectileDamage)
7         {
8             projectile = new Projectile(projectileName, projectileSpeed, projectileDamage);
9         }
10
11        public void Attack(Player target)
12        {
13            // Bereken startPosition en targetPosition
14            Vector3 startPosition = transform.position;
15            Vector3 targetPosition = target.transform.position;
16
17            // Vuur het projectiel af
18            projectile.Fire(startPosition, targetPosition);
19        }
20 }
```

Bibliotheekfuncties Gebruiken:

Maak gebruik van bestaande bibliotheekfuncties en -methoden in plaats van dezelfde functionaliteit opnieuw te implementeren.

The screenshot shows the Unity Documentation website for Version 2022.3. The navigation bar includes links for Manual, Scripting API, a search bar, and a logo. The main content area is for the **GameObject** class, which is described as a base class for all entities in Unity Scenes. It notes that many variables have been removed and suggests using GetComponent<Renderer>() instead. The **Properties** section lists various properties like activeInHierarchy, activeSelf, isStatic, layer, scene, sceneCullingMask, tag, and transform, each with a brief description.

Unity Documentation

Manual [Scripting API](#)

Search scripting...

Version: 2022.3

[FrameTiming](#)

[FrameTimingManager](#)

[FrictionJoint2D](#)

[FrustumPlanes](#)

GameObject

[GeometryUtility](#)

[Gizmos](#)

[GL](#)

[Gradient](#)

[GradientAlphaKey](#)

[GradientColorKey](#)

[Graphics](#)

[+ GraphicsBuffer](#)

[+ GraphicsBufferHandle](#)

[Grid](#)

[+ GridBrushBase](#)

[+ GridLayout](#)

[+ GUI](#)

[+ GUIContent](#)

[+ GUIElement](#)

[+ GUILayout](#)

[+ GUILayoutOption](#)

[+ GUILayoutUtility](#)

[+ GUISettings](#)

GameObject

class in [UnityEngine](#) / Inherits from [Object](#) / Implemented in [UnityEngine.CoreModule](#)

[Leave full screen](#)

[SWITCH TO MANUAL](#)

Description

Base class for all entities in Unity Scenes.

Note: Many variables in the [GameObject](#) class have been removed. To access `GameObject.renderer` in csharp, for example, use `GetComponent<Renderer>()` instead.

See Also: [Component](#).

Properties

activeInHierarchy	Defines whether the <code>GameObject</code> is active in the Scene.
activeSelf	The local active state of this <code>GameObject</code> . (Read Only)
isStatic	Gets and sets the <code>GameObject</code> 's <code>StaticEditorFlags</code> .
layer	The layer the <code>GameObject</code> is in.
scene	Scene that the <code>GameObject</code> is part of.
sceneCullingMask	Scene culling mask Unity uses to determine which scene to render the <code>GameObject</code> in.
tag	The tag of this <code>GameObject</code> .
transform	The <code>Transform</code> attached to this <code>GameObject</code> .

Inheritance en Abstraction:

Maak abstracties zoals interfaces of basisklassen om gemeenschappelijke gedragspatronen te definiëren en zorg ervoor dat concrete implementaties deze abstracties volgen.

```
1 public abstract class Tower
2 {
3     1 reference
4     public string Name { get; protected set; }
5     1 reference
6     public int Damage { get; protected set; }
7     1 reference
8     public float Range { get; protected set; }
9
10    0 references
11    public Tower(string name, int damage, float range)
12    {
13        Name = name;
14        Damage = damage;
15        Range = range;
16    }
17
18    0 references
19    public abstract void Attack(Enemy target);
20
21    0 references
22    public virtual void Upgrade()
23    {
24        // Upgrade de toren (kan in subklassen worden geïmplementeerd)
25    }
26}
```

??? ! ? ! ? !



volgende week meer...!