**Unity noter til eksamen**

**Start():**

Start-funktionen kaldes af Unity, før gameplay begynder (dvs. før Update-funktionen bliver kørt første gang) og er et ideelt sted at lave en initialisering.

**Update():**

Update kaldes for hver frame, hvis MonoBehaviour er aktiveret.

Update er den mest almindeligt anvendte funktion til at implementere enhver form for spil script. Ikke alle MonoBehaviour-script har brug for Update.

**FixedUpdate():**

Denne funktion kaldes Fixed framerate frame, hvis MonoBehaviour er aktiveret.

FixedUpdate skal bruges i stedet for Update, når du beskæftiger dig med Rigidbody. For eksempel, når du tilføjer en “force” til en Rigidbody, skal du anvende “force” for hver frame inde i FixedUpdate i stedet for hver frame i Update.

**LateUpdate():**

Et eksempel er, hvor script-koden skal overskrive effekten af en animation (f.eks. hvis en Players hoved skal kigge op et “enemyplayer-objekt” i scenen). LateUpdate-funktionen kan bruges til sådanne situationer.

**Transform:**

Position, rotation og skala af et objekt.

Hvert objekt i en scene har en transform. Det bruges til at gemme og manipulere objektets position, rotation og scale Hver Transform kan have en parent/forældre, som giver dig mulighed for at anvende position, rotation og scale hierarkisk.

**Quaternion.identity:**

identity rotation (kan kun læses).

Denne quaternion svarer til "ingen rotation" - objektet er perfekt tilpasset verden eller akserne.

**Vector2:**

Repræsentation af 2D vektorer og punkter.

Denne struktur anvendes på nogle steder til at repræsentere 2D-stillinger og vektorer (fx teksturkoordinater i en Mesh eller teksturforskydninger i Materiale). I de fleste andre tilfælde anvendes en Vector3.

**Vector3:**

Repræsentation af 3D-vektorer og punkter.

Denne struktur bruges i hele Unity til at overføre 3D-stillinger og retninger rundt. Det indeholder også funktioner til at lave fælles vektoroperationer.

Udover de funktioner, der er angivet nedenfor, kan andre klasser også bruges til at manipulere vektorer og punkter. For eksempel er Quaternion- og Matrix4x4-klasserne nyttige til at rotere eller transformere vektorer og punkter.

**MathF.PingPong():**

PingPongs værdien t (time), så den aldrig er større end længden og aldrig mindre end 0.

Den returnerede værdi vil bevæge sig frem og tilbage mellem 0 og længde.

**Time.time:**

Tiden i begyndelsen af ​​denne ramme (Read Only). Dette er tiden i sekunder siden starten af ​​spillet.

**SceneManager:**

Scenestyring i runtime.

**[SerializeField] private vs public:**

Dette kan være forvirrende, for når du opretter en public variabel, vises den som et felt i inspektøren. Men hvis du tilføjer attributten [SerializeField] til en variabel, vises den også i inspektøren.

Variabler, der er public, kan tilgås og ændres ved koden uden for klassen.

De fleste variabler, du opretter virkelig, behøves kun inde i klassen, hvor de er erklæret, og de skal være private.

Selvom det kan være fristende at gøre variabler public, fordi Unity automatisk gør dem til at virke som et felt i inspektøren, er det meget bedre at lave en variabel privat og derefter kun bruge attributten [SerializeField], hvis du har brug for få adgang til dem gennem inspektøren. Det ser ud som om, at formålet er at gøre det privat, da værdierne kunne ændres ved et uheld ved inspektøren. Men det er meget nemmere at finde fejlen end ved alle mulige scripts på kryds og på tværs.

**Corutine():**

Når du kalder en funktion, kører den til afslutning før du vender tilbage. Dette betyder effektivt, at enhver handling, der finder sted i en funktion, skal ske inden for en enkelt frame opdatering; et funktionskald kan ikke bruges til at indeholde en proceduremæssig animation eller en række arrangementer over tid. For eksempel overveje opgaven med gradvist at reducere en objekts alfa (opacitet) værdi, indtil den bliver helt usynlig. Vi bruger Corutine til vores booster som vare i 4 sekunder.

**Instantiate():**

Kloner objektet originalt og returnerer klonen.

Denne funktion gør en kopi af et gameObject på samme måde som Duplicate-kommandoen i editoren. Hvis du kloner et GameObject, kan du også eventuelt angive sin position og rotation (disse er standard for den oprindelige GameObjects position og rotation ellers). Hvis du kloner et komponent, så er GameObject det er knyttet til, også klonet igen med en valgfri position og rotation.

Når du kloner et GameObject eller Component, vil alle childobjekter og komponenter også blive klonet med deres egenskaber, der er indstillet som dem i det oprindelige objekt. Vi har brugt Instantiate til at clone vores boss som er en prefab så den spawner en ny boss hver 3 sekund.

**Singleton():**

Singleton er et objekt der kun kan blive instantiseret 1 gang.

**Object pooling():**

Hvad er Object Pooling?

Instantiate () og Destroy () er nyttige og nødvendige metoder under gameplay. Hver kræver generelt minimal CPU-tid.

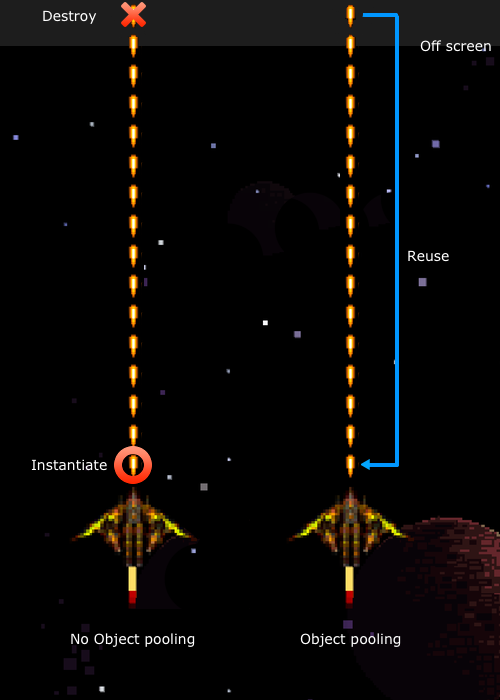
Men for objekter, der er skabt under gameplay, der har en kort levetid og bliver ødelagt i enorme tal pr. Sekund, skal CPU'en tildele betydeligt mere tid.



Desuden bruger Unity Garbage Collection til at allokere hukommelse, der ikke længere er i brug. Gentagne kald til Destroy () udløser ofte denne opgave, og det har en evne til at bremse CPU'er og indføre pauser til gameplay.

Denne adfærd er kritisk i ressourcebegrænsede omgivelser som f.eks. Mobilenheder og webapplikationer.

Objekt pooling er, hvor du forud instantiatere alle de objekter, du har brug for på et hvilket som helst bestemt tidspunkt før gameplay - fx under en læsningsskærm. I stedet for at skabe nye objekter og ødelægge gamle under gameplay genbruger dit spil genstande fra en "pool".



public class BossJump : MonoBehaviour {

private float min = 2f;

private float max = 3f;

// Use this for initialization

void Start()

{

min = transform.position.y;

max = transform.position.y + 8;

}

// Update is called once per frame

void Update()

{

transform.position = new Vector3(transform.position.x, Mathf.PingPong(Time.time \* 8, max - min) + min, transform.position.z);

}

}

Vi har to private fields, min og max med værdier. Ved Start() sætter vi min til “ingenting” og max til +8 hvor begge er på y-aksen. På Update() laver vi så en transformering på position ved y-aksen hvor vi så bruger en Vector3 som tager en x værdi, en y værdi hvor vi bruger **Mathf.PingPong som gør så den aldrig er større end længden og aldrig mindre end 0. Den returnerede værdi vil bevæge sig frem og tilbage mellem 0 og længde.** Samt en z værdi.

public class MonsterScript : MonoBehaviour {

private float min = 2f;

private float max = 3f;

// Use this for initialization

void Start()

{

min = transform.position.x;

max = transform.position.x + 1;

}

// Update is called once per frame

void Update()

{

transform.position = new Vector3(Mathf.PingPong(Time.time \* 2, max - min) + min, transform.position.y, transform.position.z);

}

}

Vi har to private fields, min og max med værdier. Ved Start() sætter vi min til “ingenting” og max til +1 hvor begge er på x-aksen. På Update() laver vi så en transformering på position ved x-aksen hvor vi så bruger en Vector3 som tager **Mathf.PingPong som gør så den aldrig er større end længden og aldrig mindre end 0. Den returnerede værdi vil bevæge sig frem og tilbage mellem 0 og længde.** En y værdi. Samt en z værdi.

public class ScoreScript : MonoBehaviour {

public static int scoreValue = 0;

Text score;

// Use this for initialization

void Start () {

score = GetComponent<Text>();

}

// Update is called once per frame

void Update () {

score.text = "Gems: " + scoreValue;

}

}

Vi bruger en public int scoreValue som i starten er 0. Også har vi en Text score. Så på Start() sætter vi score til at finde komponentet<Text>. Så på Update() sætter vi så score.text til en string samt den talte værdi fra scoreValue når vi samler en gem op.

public class KeyScript : MonoBehaviour {

[SerializeField] private string findWall;

[SerializeField] private string findKey;

void OnTriggerEnter2D(Collider2D theCollider)

{

if (theCollider.CompareTag("Player"))

{

GameObject wallFound = GameObject.Find(findWall);

GameObject keyFound = GameObject.Find(findKey);

Destroy(wallFound);

Destroy(keyFound);

}

}

}

Vi har to private fields findWall og findKey. Vi bruger så OnTriggerEnter2D (når den kollidere) så spørger vi om det er spilleren. Så finder vi den tilhørende mur og den tilhørende nøgle som den får ind fra Unity. Når de er fundet så, Destroy dem.