## Platformgenerering

En essentiel del af dette spil er den funktionalitet, som står for at skabe de objekter, spilleren bevæger sig på. Fokus lå på at skabe en autogenereret bane, som var tilfældigt udformet ved hvert gennemspil. Formålet var at holde spillet interessant, da banen ikke bliver indøvet. Samtidig effektiviserer det arbejdet ift. at opsætte hver enkelt bane, da alternativet er at placere hvert platform manuelt i samtlige baner. Selvom kompleksiteten ved at autogenerere platformene er højere, så er det stadig en mere effektiv proces.

I spillet består banen af tre forskellige typer platforme; start platformen, de mellemliggende platforme og den afsluttende platform.

* Startplatformen, som er den første platform i scenen, er altid tilstede, da den sikrer spillerobjektets startposition.
* De næste objekter er de mellemliggende platforme, som udgør størstedelen af banen. Disse bliver autogenereret sideløbende med spillerens fremgang i banen. De kommer herudover i tre forskellige typer, lav, medium og høj.
* Den sidste platform i banen er slutplatformen, hvor spillet afsluttes, når spilleren når denne.

### Tilfældig generering af platforme

De platform-relaterede komponenter i selve scenen består af to objekter. Den ene er startplatformen, som er et simpelt objekt uden et script pålagt og de standard ’Transform’-, ’Box Collider’- samt ’Mesh Renderer’-komponenter.

Det andet objekt er PlatformSpawn, som står for at oprette de kommende platforme, som spilleren hopper på. PlatformSpawn-objektet er igen et simpelt objekt, da det ligger uden for spillerens syn, men har to tilhørende scripts, SpawnScriptDistance og SpawnEnemy. SpawnEnemy er et script, som står for at oprette fjender i banen, og vil derfor ikke blive redegjort for i dette afsnit, hvor SpawnScriptDistance afhandler banebygningen.

Dette script har to formål. Ét er at generere den række objekter, hvor den enkelte platform tilfældigt vælges ud fra de tre tilgængelige modeller. Herunder gøres der også brug af en teknik, kaldet ’object pooling’. Det andet er at oprette slutplatformen når den sidste autogenereret platform er sendt afsted.

Kigger man nærmere på scriptet gør det brug af 4 metoder samt de indbyggede Unity-metoder, Start() og Update().

Start()-metoden indeholder kun to kald til metoden FillPool() og SpawnPlatform(), hvor FillPool er metoden, som opretter en liste med objekterne, der udgør de autogenerede platforme. Efter FillPool-metoden kaldes SpawnPlatform, som opretter objektet i scenen, hvis det ikke er oprettet på forhånd.

FillPool-metoden er den metode, som opsætter ’object pooling’.

’Object pooling’ er en teknik, som bruges til at optimere brugen af større mængder objekter af den samme type. Et typisk eksempel af denne situation er brug af projektiler, hvor hvert projektil er et objekt. Den simple løsning er at oprette et nyt objekt, når skyde-metoden køres, og ødelægge objektet, når det ikke bruges mere. Ulempen ved denne metode er, at det kræver CPU-kraft at oprette et objekt i hukommelsen. Dette kan især være et problem for mobile enheder, som har meget begrænset processorkraft. I stedet for at oprette og ødelægge objekterne efter behov, så opretter man alle objekterne (af samme type) i hukommelsen, når scriptet bliver kaldt første gang. Objekterne bliver herefter sat til aktive, når de bliver brugt, og sat til inaktive, når de ikke bruges mere.

Som ses i scriptet *figur x*, oprettes først en liste af objekter, *list*.

Herefter oprettes hvert enkelt objekt i hukommelsen med tilhørende koordinater og rotationer i scenen, som er angivet i ’transform.position’ og ’Quaternion.identity’. Derefter sættes objektet til inaktiv, hvorefter det tilføjes til listen.

private void FillPool() {

list = new List<GameObject>();

for (int i = 0; i < pooledAmount; i++) {

for (int n = 0; n < obj.Length; n++) {

GameObject go = (GameObject)Instantiate(obj[n], transform.position,

Quaternion.identity);

go.SetActive(false);

list.Add(go);

}

}

}

Attributten ’pooledAmount’ bestemmer antallet af hver type objekt, som pooles. Dette er nødvendigt, da det skal være muligt at tilføje flere forskellige typer objekter.

Update-metoden er den metode, der bliver kaldt hvert frame i scenen, og er derfor hovedmetoden i dette script. Det første, som sker i metoden, er et kald til metoden ’isAllObjectsSpawned’, som tjekker på antallet af objekter placeret i scenen. Matcher dette antallet af objekter, som der er forventet, sætter den attributten ’\_allObjectsSpawned’ til ’true’. Det bruges herefter i en ’if’-løkke, som kalder metoderne, ’SpawnPlatform’ eller ’SpawnUnique’, som henholdsvis placerer det næste objekt i scenen eller afslutningsplatformen afhængig af ’\_ allObjectsSpawned’.

void Update () {

isAllObjectsSpawned();

float distance = Vector3.Distance(this.transform.position, lastObject.transform.position);

if(\_allObjectsSpawned == false) {

if(distance >= distanceBetweenObjects)

SpawnPlatform();

}

else {

if (distance >= distanceBetweenObjects)

SpawnUnique();

}

}

Update-metoden har endvidere en attribut, ’distance’, som er afstanden imellem det nuværende objekt og det sidst placeret objekt. Dette tal bruges til at tjekke minimumsgrænsen imellem de placerede objekter, og bliver samlet med attributten, ’distanceBetweenObjects’.

SpawnPlatform-metoden er den metode, som vælger et tilfældigt inaktivt objekt i listen og sætter det til ’aktivt’. Mere specifikt vælger metoden et tilfældigt objekt, tjekker om det er inaktivt eller ej. Viser det sig at være inaktivt sættes der gang i iterationen.

private void SpawnPlatform() {

bool found = false;

while (!found) {

int i = Random.Range(0, list.Count);

if (!list[i].activeInHierarchy) {

\_spawnedObjects++;

lastObject = list[i];

list[i].transform.position = this.transform.position;

list[i].SetActive(true);

found = true;

}

}

}

’\_spawnedObjects’ tælles op, det nyoprettede objekt gemmes i variablen ’LastObject’ og sættes til ’aktivt’. Sidst sættes variablen ’found’ til ’true’ så løkken afsluttes.

Den sidste funktion i dette script er SpawnUnique, som er den metode, som placerer slutplatformen i spillet, hvorefter der slukkes for objektet. Dette script bliver kun kaldt, når der ikke er flere inaktive objekter i listen.

private void SpawnUnique() {

Instantiate(uniqueObject, transform.position, Quaternion.identity);

this.gameObject.SetActive(false);

}