I Am not Legend

Af Oliver Mads Kragh

27/04/23

Projekt lavet med

Jeppe Møller Wortmann

Rasmus Riemke Rubæk

Med fokus på

Wave Spawner og Zombier

**15000 Tegn**



# 

# Resume

Jeg har lavet et spil udviklet med Jeppe og Rasmus. I spillet skal man dræbe zombierne, som spawner gradvist for at overleve. Det er muligt at skyde dem, men jo flere zombier man dræber, jo sværere er det at overleve, eftersom der spawner flere og flere.

# Problemformulering

Vi valgte netop at lave vores program fordi alle gruppemedlemmer interesserer sig for zombie-genren. Dette program inkorporerer også en stor del af de funktioner og lignende, vi har lært i de forskellige forløb i undervisningen.

# Krav til opgaven

I skal udvikle et valgfrit spil i Unity.

Undersøg hvilke elementer spillet skal indeholde og gør rede for, hvad der ligger til grund for jeres beslutninger. Spillet skal indeholde et UI (User Interface), en hovedmenu og relevante optioner i forbindelse med spillet. ***Dokumenter jeres designovervejelser og løbende ændringer*** for at kunne dokumentere udviklingen af spillet.

# Projekt beskrivelse

Vi valgte netop at lave vores program fordi alle gruppemedlemmer interesserer sig for zombie-genren. Dette program inkorporerer også en stor del af de funktioner og lignende, vi har lært i de forskellige forløb.

Vi vil udarbejde et program, der minder om Call of Duty: Zombies, men med en anden målgruppe, og mindre avanceret. Målgruppen er mere fokuseret på personer der er nye til genren.

Vi vil redegøre for programmets udvikling, heri med fokus på selve koden og opsætningen.

Vi vil undersøge, hvorvidt oplevelsen med spillet kan forbedres ved hjælp af flere forskellige sværhedsgrader.

Vi vil diskutere hvorvidt produktet kan vække en interesse for zombie-genren.

Vi vil diskutere hvorvidt spillet kan udvides med flere funktioner.

Jeppe Møller Wortmann vil i sin synopsis besvare problemstillingen med fokus på Playercontrolleren, UI og våben.

Oliver Mads Kragh vil i sin synopsis besvare problemstillingen med fokus på Wavespawner og integrering af GitHub.

Rasmus Riemke Rubæk vil i sin synopsis besvare problemstillingen med fokus på troubleshooting, zombier, og 3D-objekterne.

# Programbeskrivelse

Spilet består af en lysning i en skov med en bakke. En spiller løber her rundt med en automatisk riffel og en hånd pistol. Der dukker pludseligt mange zombier op rundt i lysningen. De bliver spawned i waves. Spilleren skal så dræbe zombierne for ikke at dø og tabe. Zombierne spawner sammen i forskellige områder rundt i banen, og der kommer gradvist flere og flere zombier. Spillet går ud på at dræbe zombier, som spawner rundt i mappet. Der bliver spawned et vis antal zombier per wave. Når alle zombierne er døde, starter en ny wave. Spillet slutter når spilleren dør. Det er i spillet muligt at vælge sværhedsgrad. Nem Normal og Svær. Når der trykkes på start knappen starter spillet. Under spillet kan der trykkes esc og så pauses spillet. Her kan man vælge at fortsætte eller stoppe spillet.

# Funktionalitet

Det skal være muligt at kunne vælge sværhedsgrad. Det skal være muligt at kunne starte spillet gennem et ui. Det skal være muligt at pause spillet. Det skal være muligt at bevæge sig rundt i map’et. Det skal være muligt at kunne skyde zombierne. Det skal være muligt at dø af zombierne. Det skal være muligt at reloade. Det skal være muligt at løbe og hoppe. Det skal være muligt at kunne bevæge sit synspunkt rundt.

# Rutediagrammer af wave spawner

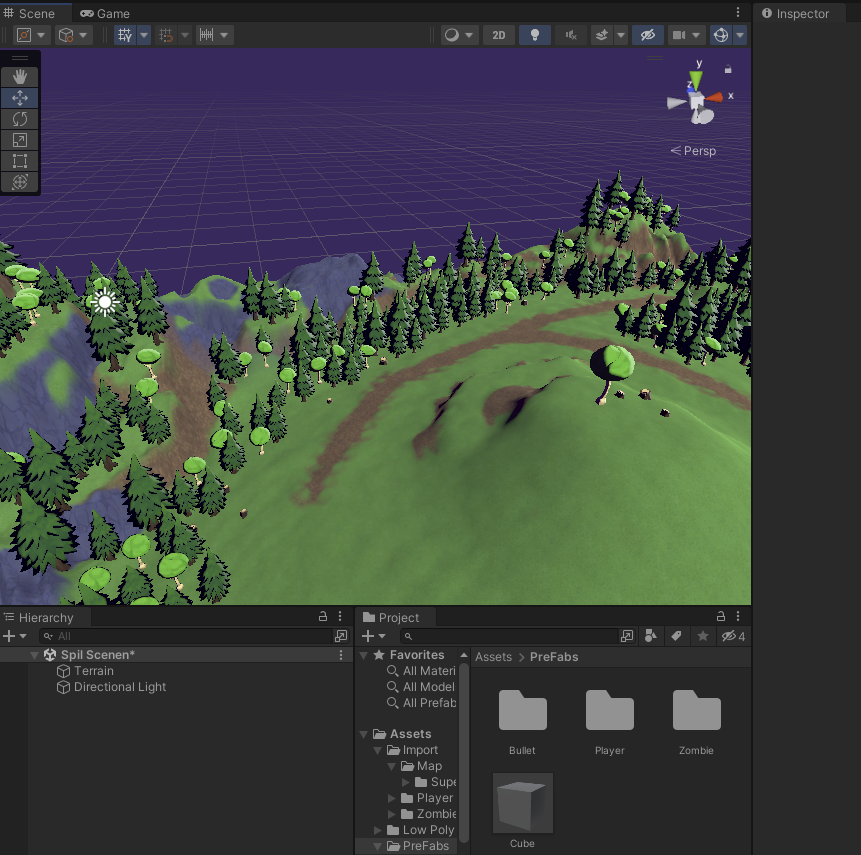
# 

Vi kan på rutediagrammet se hvordan koden bliver påvirket af spilleren. Vi starter med variablen Start hvor spillet startes. Vi har så et input i form af difficulty som spilleren vælger. Dette bliver til et dokument spillet bruger til at genere en sværhedsgrad i spillet. Så sker processen Generering af en Wave hvilket starter processen Wavetimer. Denne wavetimer bliver brugt som dokument men også som et input. Når spilleren dræber zombierne, er dette et input som bruges af processen som tæller zombier. Zombie tæller laver et dokument som bliver brugt af koden, samt en variabel som tjekkes af et binært valg. Hvis spillet ikke kører, sluttes waven. Hvis spillet kører, så kommer der endnu et binært valg. Er alle zombierne dræbt , så starter en ny wave. Hvis ikke alle zombier er dræbt, starter den endnu et binært valg. Når wave timerne er forbi, startes processen som genererer en ny wave. Hvis ikke så starter processen som tæller mængden af zombier. Altså zombierne bliver talt og så tjekker den alle forholdene igen. Når processen som laver en ny wave startes, så starter den også processen som nulstiller wavetimerne, hvilket nulstiller hele waven. Derefter startes processen start wave timer og en ny wave startes.

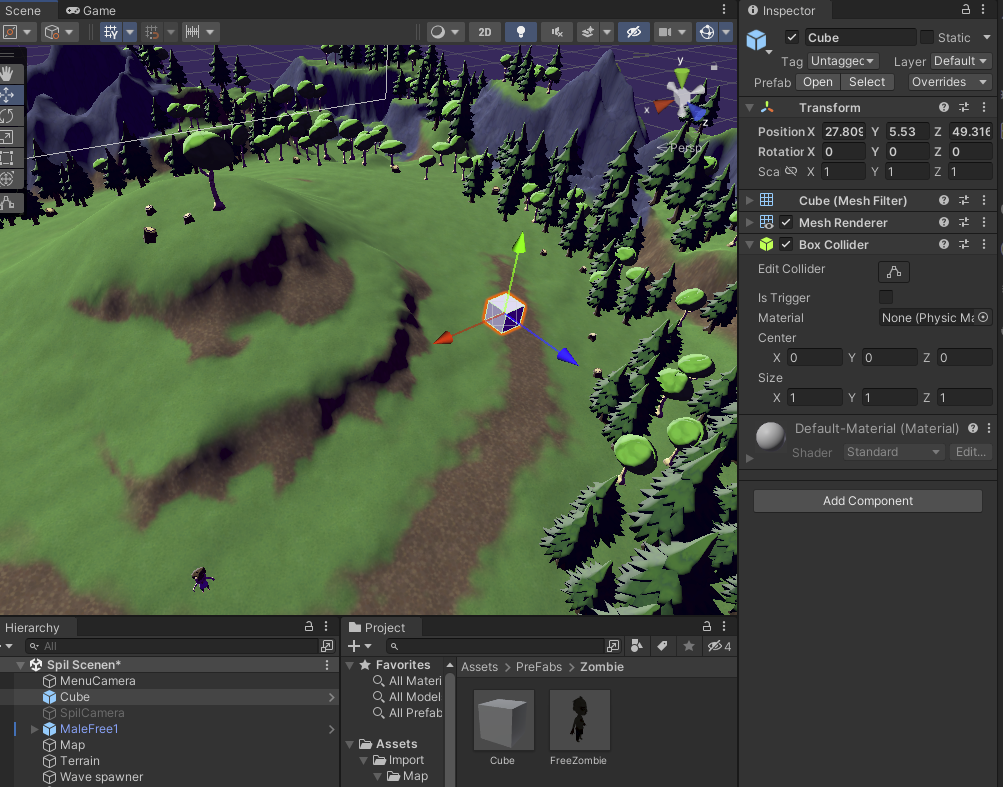
# Brugergrænseflader

**Prototyper af spillet**

Til at starte med fandt vi et map vi mente var tilpas til spillet og vi udviklede derefter løbende.



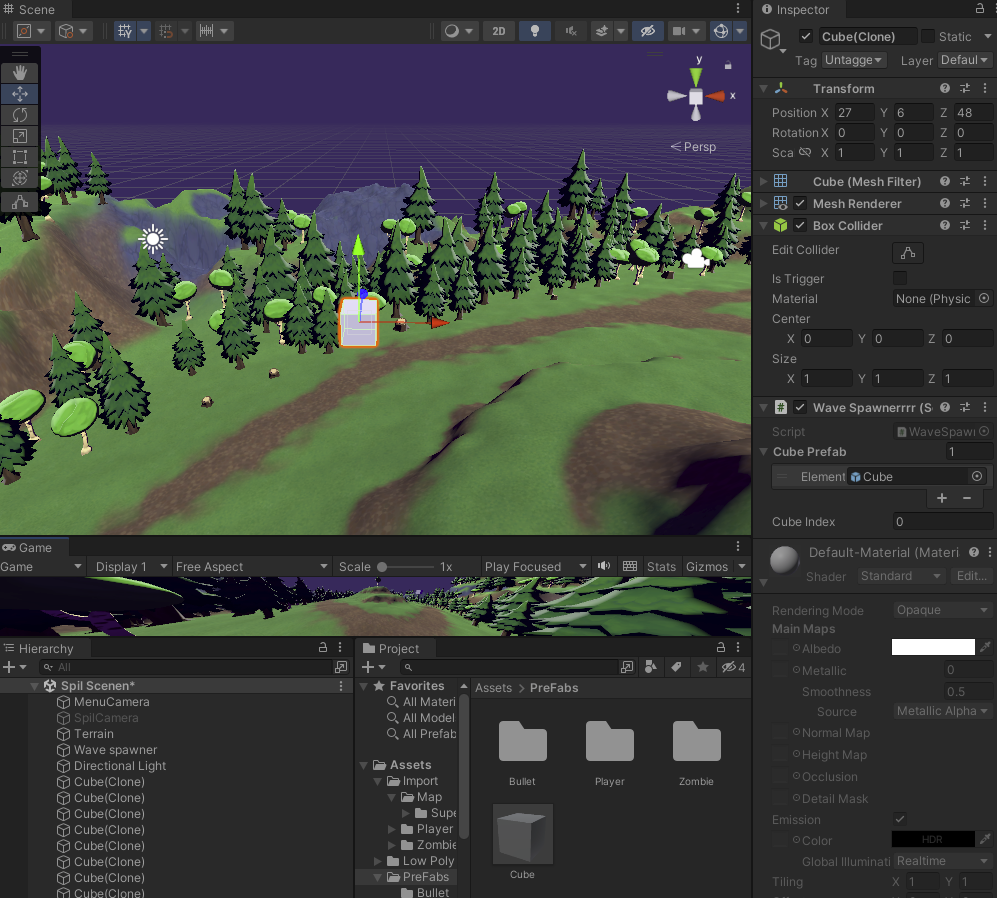
Så begyndte jeg at arbejde på wave spawner hvor jeg til at starte med prøvede at få lavet en funktion hvor vi kunne spawne et objekt som set på billedet under.



Jeg lagde et tomt objekt ind i spillets scener. Dette objekt gav jeg scriptet wavespawner. Så skrev jeg en smule kode vi lærte fra vores andre projekter om til at passe med firkanten, jeg prøvede at spawne ind.

# 

# Udvalgt kode



using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class WaveSpawnerrrr : MonoBehaviour

{

public GameObject[] cubePrefab;

public int cubeIndex;

// Update is called once per frame

void Update()

{

if(Input.GetKeyDown(KeyCode.S))

{

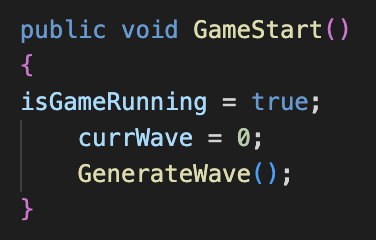
Instantiate(cubePrefab[cubeIndex],new Vector3(27,6,48),cubePrefab[cubeIndex].transform.rotation);

}

}

Vi kan her se koden og dets funktion på billedet over. Hvor hver gang spillet opdateres, skulle det tjekke, at “if” kommandoen var rigtig. Kommandoen virkede ved at hvis man trykker på S tasten så ville den spawne en cube med kommandoen “Instantiate”. Instantiate kopierer cuben fra listen cubeindex og spawner den på positionen (27, 6, 48). Dette virkede og derfor fortsatte jeg med at opbygge en spawner. Funktionen skulle være at spawne et antal zombier indenfor et område forskellige steder på mappet.

Da vi opbyggede wavespawnern havde vi ikke udviklet difficulty endnu. Dette gjorde at vi var nødt til at skrive wavespawnern om for at tage højde for difficultien.



antalE = antalE \* 1.5f;

Sådan så koden ud før og det var ikke de store problemer at implementerer. Vi havde problemer med at læse difficultien eftersom den var låst i et script. Vi implementerede så et UItomt objekt med en masse værdier som var muligt at aflæse af andre scripts. Vi kunne nu hente sværhedsgraden fra UI og det fungerede dermed sammen.

public int difficulty;

difficulty = (int)GameObject.Find("UItomt").GetComponent<UI>().difficulty;

antalE = antalE \* (int)GameObject.Find("UItomt").GetComponent<UI>().difficulty;

Vi tilføjede disse koder og ændrede koden bag antalE.

# Wavespawner Fokusområde

Jeg begyndte nu at opbygge en mindre avanceret WaveSpawner med mulighed for at ændre alle variabler vi kunne forestille os. Et krav var, at den selv skulle finde rundt i antallet der skal spawnes, hvor de skal spawnes og sværhedsgraden. Det er noget af den længste sammenhængende kode jeg har skrevet og det var også svært. Men ved at gå struktureret til værks gik det.

public List<Enemy> enemies = new List<Enemy>();

Laver en liste over objekterne enemies(fjenderne) spilleren skal dræbe, kan påvirkes i unity editoren

public int currWave;

definerer den aktive wave, kan påvirkes i unity editoren

public List<GameObject> enemiesToSpawn = new List<GameObject>();

laver en liste med fjender som skal spawnes, kan påvirkes i unity editoren

public Transform[] spawnLocation; ,

laver en liste med spawnpoints hvor fjender kan spawne, kan påvirkes i unity editoren

public int spawnIndex;

laver en liste til at holde styr på hvilket spawnpunkt der skal bruges til hvilken wave, kan påvirkes i unity editoren

public float areaRand;

laver en værdi der er mulig at ændre inde i unity, denne værdi er området zombierne kan

public int waveDuration;

Definerer varigheden af en bølge, kan påvirkes i unity editoren

private float waveTimer;

Den holder styr på tiden af en bølge

private float spawnInterval;

Intervallerne mellem fjenderne skal spawne

private float spawnTimer;

holder styr på hvor lang tid der er tilbage før en ny fjende spawner

public float antalE;

laver en værdi med antallet af fjender der skal spawnes i bølgen, kan påvirkes i unity editoren

public int difficulty;

Laver værdien difficulty, kan påvirkes i unity editoren

public List<GameObject> spawnedEnemies = new List<GameObject>();

Laver en list med fjender som er spawnet, kan påvirkes i unity editoren

private bool isGameRunning;

laver en værdi som indikerer om spillet køre. Hvis den er lig med true køre spillet indtil den er sat til falsk

public void GameStart()

Den køre koden når spillet starter

isGameRunning = true;

fortæller at spillet kører

currWave = 0;

Sætter en startværdi til currWave så spillet ved at det er den første wave

difficulty = (int)GameObject.Find("UItomt").GetComponent<UI>().difficulty;

Sætter difficulien til værdien fra UItomt som er afhængig af UI menuen Setting hvor man kan vælge sværhedsgraden

GenerateWave();

Den starter funktionen GenerateWave

void FixedUpdate()

Den opdaterer hvert frame

if (isGameRunning)

Den tjekker om spillet køre hvis det gør køre den kommandoen under

if (spawnTimer <= 0)

Den tjekker om spawnTimer er over 0, hvis dette er sandt vil koden blive eksekveret

Vector3 randomPos = spawnLocation[spawnIndex].position + new Vector3(Random.Range(-areaRand, areaRand), 0, Random.Range(-areaRand, areaRand));

definerer en position for zombierne

GameObject enemy = (GameObject)Instantiate(enemiesToSpawn[0], randomPos, Quaternion.identity);

spawner en fjende fra det første element i listen enemiesToSpawn på positionerne fra før

67enemiesToSpawn.RemoveAt(0);y

fjerner fjenden fra listen enemiesToSpawn

spawnedEnemies.Add(enemy);

tilføjer fjenden til listen spawnedEnemies

spawnTimer = spawnInterval;

nulstiller spawnTimer

else

Hvis if kommandoen ikke er sand eksekveres else kommandoen

waveTimer = 0;

Hvis der ikke er flere zombier nulstiller den waveTimer

else

Hvis waveTimern ikke er lig med 0 eksekveres denne kommando

spawnTimer -= Time.fixedDeltaTime;

For eksempel, hvis spawnInterval er 4 sekunder så tæller spawnTimer ned fra 4 sekunder indtil 0 sekunder. Når spawnTimer er 0 sekunder, vil spawneren spawne en fjændre fra listen

waveTimer -= Time.fixedDeltaTime;

når waveTimer tælle ned fra waveDuration og når 0 sekunder vil den stoppe bølgen og starte en ny

if (waveTimer <= 0 && spawnedEnemies.Count <= 0)

kommandoen tjekker om waveTimer og spawnedEnemies.Count er mindre eller ligmed nul. Så er bølgen færdig og der er ikke flere fjender

currWave++;

denne øger currWave med 1 for at starte en ny wave

spawnIndex = (spawnIndex + 1) % spawnLocation.Length;

Kommandoen ændre spawnIndex til næste ved at tilføje 1 til spawnIndex derefter tjekker den med antallet af spawnLocation for ikke at overskride dette

GenerateWave();

den starter GenerateWave funktionen

public void GenerateWave()

Starter en wave

GenerateEnemies();

Starter funktionen GenerateEnemies

spawnInterval = (int)(waveDuration + 0.01f / (int)antalE + 0.01f);

Sætter spawnInterval til tiden mellem fjenderne skal spawne baseret på waveduration og antalE

waveTimer = waveDuration;

Sætter waveTimer ligmed WaveDuration for at

public void GenerateEnemies()

Starter funktionen GenerateEnemies

antalE = antalE \* (int)GameObject.Find("UItomt").GetComponent<UI>().difficulty;

Den sætter antalE lig med antalE gange med sværhedsgraden som den finder fra UItomt dette gør jeg for at afspejle sværhedsgraden i spillet og så det er muligt at ændre denne løbende i spillet

List<GameObject> generatedEnemies = new List<GameObject>();

Den laver listen med fjender som er blevet spawnet i spillet

for (int i = 0; i < (int)antalE; i++)

Den starter en for lykke der køre antal gange ligmed antalE, den starter ved 0 (i) og slutter når den når antallet som antalE er ligmed. Det er her vigtigt at den afhænger af antalE som i forvejen afhænger af sværhedsgraden, vi behøver dermed ikke af varierer hver gang sværhedsgraden ændres

int randEnemyId = Random.Range(0, enemies.Count);

Denne funktion finder en tildfældig fjende fra listen dette gør det mugligt at have forskellige fjender som spawnes sammen, vi har dog kun en slags fjende så variabel gør ikke det store i vores spil nu.

generatedEnemies.Add(enemies[randEnemyId].enemyPrefab);

Denne tager fjenderne der er muligt at spawne som kommandoen ovenfor har valgt og tilføjer dem til listen generatedEnemies. Når kodenovenfor dermed har valgt en tilfældig fjende fra de mulige prefabs så sender denne kode dem videre på listen generatedEnemies.

enemiesToSpawn.Clear();

Når lykken er slut fjernes alt fra listen enemiesToSpawn

enemiesToSpawn = generatedEnemies;

Den sikre at der ikke er flere enemies i listen ved at overføre værdien

public void ResetSpawner()

bruges til at resete spawnerne når en wave er slut

currWave = 0;

sætter currWave ligmed 0

isGameRunning = false;

Sætter isGameRunning ligmed falsk så spillet stopper med at spawne fjender

foreach (GameObject enemy in spawnedEnemies)

alle enemy i listen spawnedEnemies bliver fjernet med denne kommando

Destroy(enemy);

Denne fjerner enemiesne fra listen

spawnedEnemies.Clear();

Den ryder listen spawnedEnemies

enemiesToSpawn.Clear();

Den ryder listen enemiesToSpawn

antalE = 1;

den sætter antallet af enemies ligmed 1

SceneManager.LoadScene(0);

den loader startsmenuen

[System.Serializable]

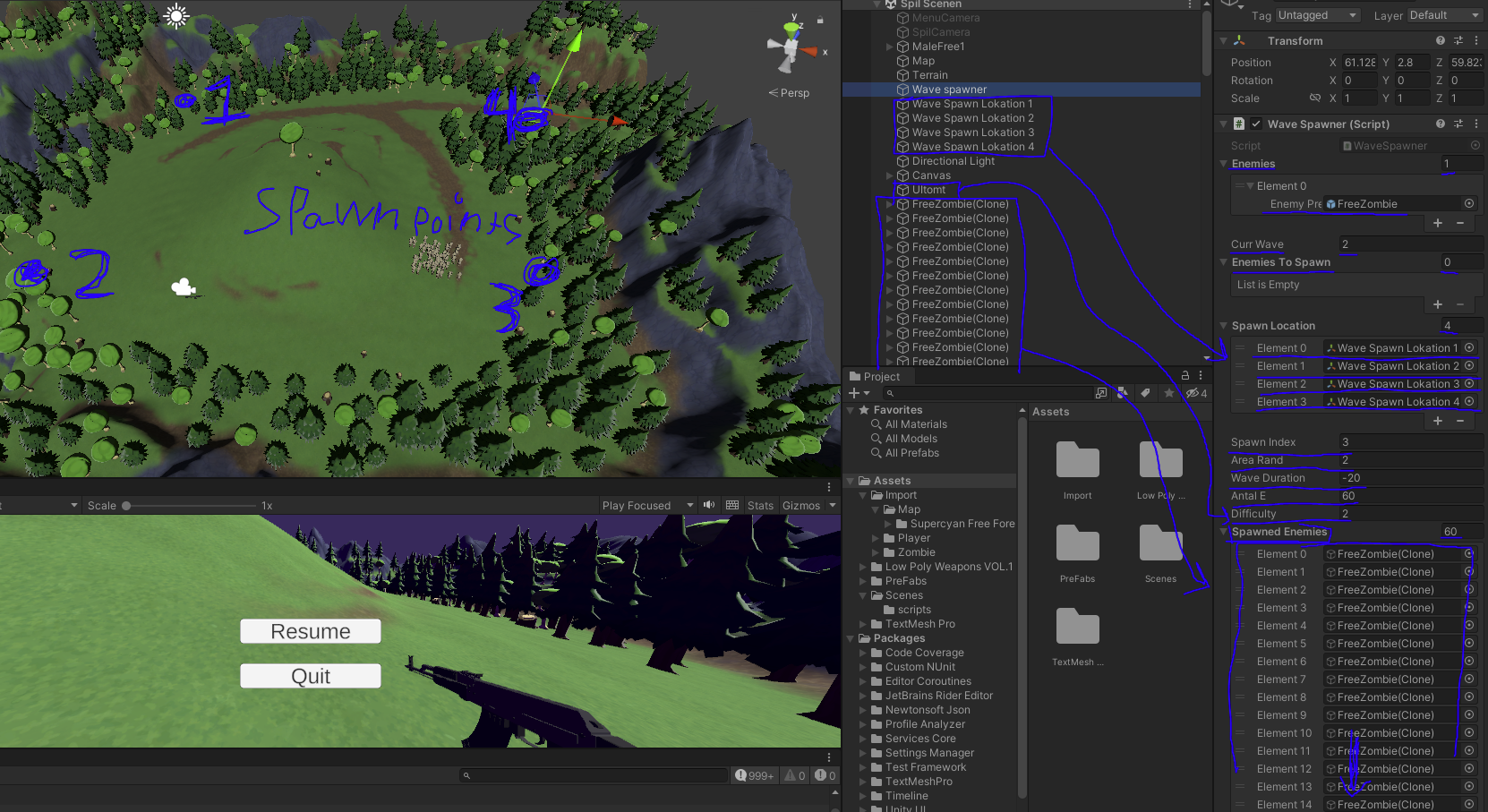
den sikre at Enemy prefaben kan redigeres i unity editoren

public class Enemy

ved at gøre Enemy public

public GameObject enemyPrefab;

Den gør prefaben public

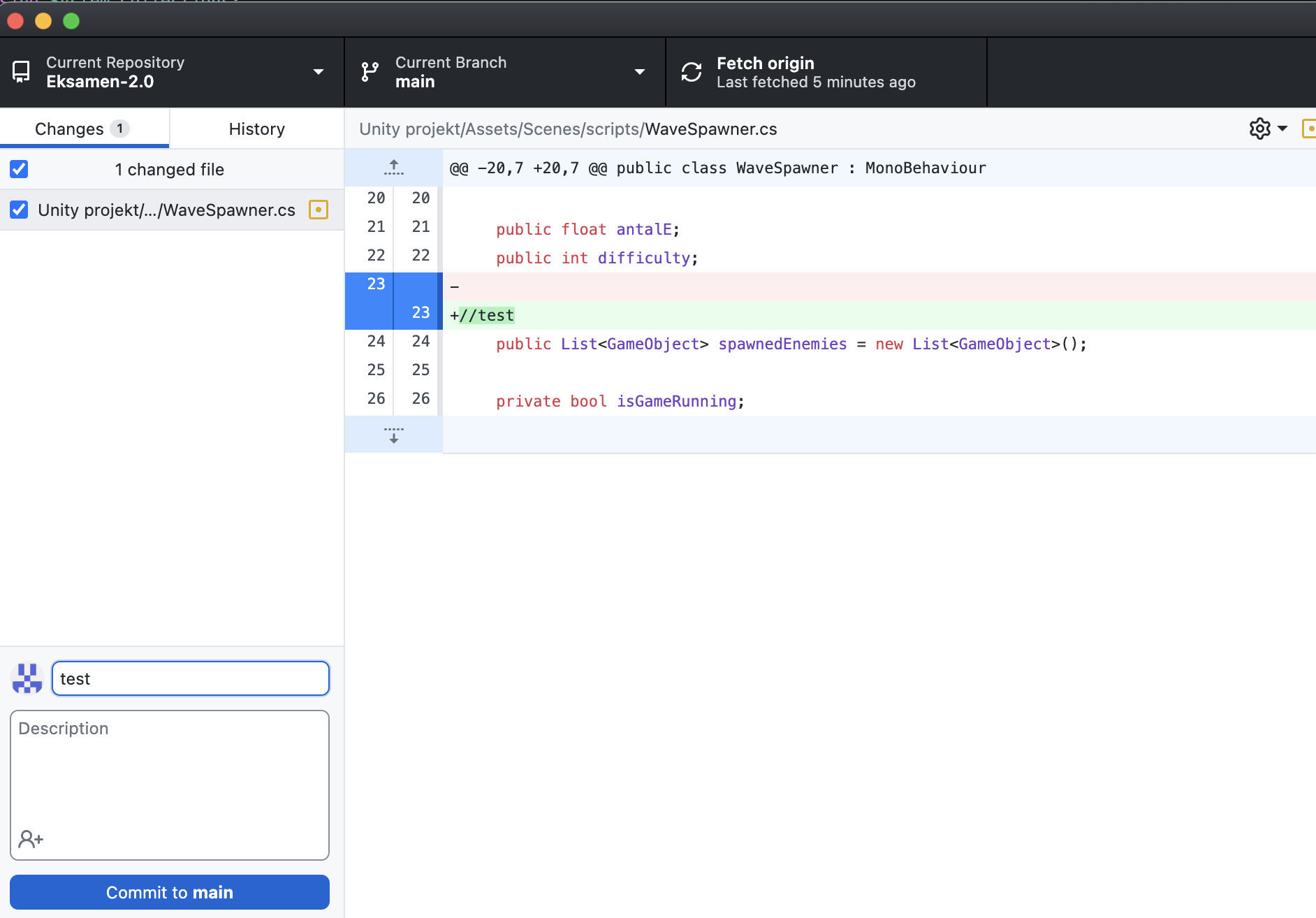


Vi kan her se koden i gang.

Først giver jeg den en enemy som vi kan se på listen “Enemies” hvor prefaben for vores zombier er lagt ind. Den ved dermed at det er det objekt den skal clone når den spawner en enemy. Vi ser samtidigt antallet i listen dette bruger vi dog ikke nu men det kan bruges hvis der er flere forskellige zombier. Under ser vi hvilken wave spilleren er ved altså den aktive wave. Så kommer listen “Enemies To Spawn” dette er listen med fjender den er ved eller mangler at spawne, eftersom “Wave Duration” er -20 så går der ikke lang tid imellem zombierne spawner. Vi kan samtidigt se på listen “Enemies To Spawn” at der er 0 antal på listen hvilket stemmer overens med at vi på screenshottet kan se at fjenderne er på vej mod spilleren. Så kommer vi til listen “Spawn Location” hvor vi kan se der er 4 lokationer fjenderne kan spawne på. Disse lokationer er tegnet ind på mappet da de er et tomt objekt med en position som koden bruger som spawnpoint for fjenderne. Vi kan her se at fjenderne er spawnet ved lokation 2 som er spawn lokation 3 hvilket stemmer overens med at vi er ved currWave 2 så tredje wave. Eftersom den tæller fra 0. Så har vi spawnIndex som holder styr på spawn lokationerne. Vi kan dermed vælge hvilken spawn lokation vi vil starte waven ved. Så er der “Area Rand” hvilket er værdien for området zombierne kan spawne på. Så er der “Antal E” som er antallet af fjender. Den startede på 15 i wave 0. Eftersom “Difficulty” er 2 som er den normale så ganges der med 2 per wave. Altså hvis der er 15 fjender i wave 0. Så spawner den 30 fjender i wave 1 og 60 fjender i wave 2 som screenshottet er taget i. Vi kan dermed se at dette stemmer overens med værdierne fra unity editorn. Vi har valgt at lave værdierne public så der var mulighed for at ændre i spillet nemt og hurtigt. Vi kunne dermed løbende ændre for sværhedsgraden, currwave eller antallet af zombier for at komme frem til et mere balanceret spil. Ved at gøre det muligt at læse og redigere i unity editoren er det også nemt at se hvad koden gør og er i gang med hvis der er fejl.

# GitHub implementering

Det startede med at jeg selv havde opsat GitHub til at gemme mine projekter med eftersom jeg både skrev på Windows og Mac. GitHub gav mulighed for cloud og local save cross platform hvilket gjorde den attraktiv for mig. Vi opsatte vores projekt på GitHub eftersom jeg kun havde god erfaring. Jeg havde dog ikke prøvet at være flere personer i samme Github save. Ved at installere GitHub Desktop app var det muligt at lave en privat repository hvor vi kunne have en mappe som var clonet med clouden. Når vi havde lavet en ændring så skal man dermed “pushe” sin opdatering med en kommentar. Så bliver den overført til clouden og synkroniseret på alle maskinerne der er sat op. Dette fungerer ved at man har en mappe som er synkroniseret med clouden. Når man så laver en ændring i mappen, så ændre den clouden og alle maskiner forbundet. Når der så laves en ændring af andre, ændres ens lokale mappe til at være synkroniseret med clouden. Man starter med at opsætte et Repository på github.com, så man kan aktiverer linket og lave en dedikeret mappe til desktop-appen. Den synkroniserer så ændringerne deri og det er så muligt at bruge alle features. Man kan så åbne scriptene igennem github-appen eller gennem unity projektet. Man skal bare huske at pushe sine updates ellers er de kun gemt lokalt. Dette kan være et problem hvis man skriver på forskellige maskiner. Hvis andre så ændrer i det samme script som man selv har ændret i, men glemmer at pushe. Kan det ødelægge ens ændringer og de er dermed ubrugelige. Dette har vi oplevet flere gange, eftersom vi har glemt at pushe efter vi har lavet ændringer.



Vi kan her se at jeg i WaveSpawner har lavet en ændring “//test” Vi kan samtidigt se hvor ændringen er lavet og dets påvirkning i programmet. Dette har været meget praktisk med en detaljeret beskrivelse af hver opdatering, hver person laver. Vi har brugt det meget til debugging. Det er også muligt at se alle ændringer under historien tappen hvilket gør at vi kan vende tilbage til en ældre opdatering hvis vi har fejl med programmet. Der er dog ikke mulighed for live edition af flere personer som vi kender fra Google Drive. Hvis flere personer ændrer det samme program, kan der risikerer at ske fejl. Vi har derfor prøvet at undgå at redigere i projektet, imens en anden sidder og ændrer noget. Vi lavede derfor en messenger gruppe hvor man skriver hvis man begynder at redigerer og når man er færdig. Vi havde derfor mulighed for at redigere programmet på farten. Dette har vi brugt meget, nogle af vores bærbar har ikke kunne trække unity programmet. Vi har derfor kunnet redigerer i scriptsne, sendt det til clouden og så vil en af de andre kunne teste ændringer i unity. Dette gør at vi har kunne rette fejl hurtigt uden at være ved en kraftig stationær. Dette har fungeret meget godt og har gjort det muligt for os at arbejde på projektet uanset hvor vi har været.

# Videreudvikling

Vi har undervejs fundet det fedt og spændende at se, hvor mange muligheder der er, når man programmerer. Det har også givet os blod på tanden i forhold til om vi kunne gøre spillet endnu bedre. Vi har mange ideer fra vores brainstorm som ikke er nået til livs endnu. I forhold til waveSpawner ville det være spændende at lave flere varianter af zombier med forskellige sværhedsgrader. Simplificering af waveSpawnern ville også være relevant, eftersom der er mange variabler som påvirker det endelige resultat. Implementering af github og mere struktur deri ville også gøre at vi ville kunne arbejde mere effektivt. Vi har bøvlet en del med github, fordi vi ikke rigtig kendte til alle funktioner. Der er mulighed for at lave en branch, dette vil sige at man ville kunne lave en ændret version af spillet uden at overwrite ens primære save. Dette ville have været nyttigt til at teste funktioner med, eftersom man ikke er nødt til at slette og skrive det man havde før. Her ville man bare kunne vælge at køre videre med den primære. Alt i alt har det været et spændende forløb med interessante udfordringer.

# Konklusion

Vi har udviklet et spil i unity med et UI, et hovedmenu og en udfordring. Jeg har forklaret udfordringer og udviklingen bag wavespawnern. Som til at starte med hvor jeg prøvede med en cube for at være sikker på funktionerne. Vi har udviklet et interessant spil for alle målgrupper eftersom der med difficulty er mulighed for at vælge hvor svært spillet skal være. Vi har også haft fokus på at bruge det vi har lært i undervisningen.

# Credits

Vi har valgt at fokuserer på koden og spillet, derfor har vi fundet graphichs’ne på Unity assets store. Mappet er lavet af “Supercyan” og hedder “Environment Pack: Free Forest Sample”[[1]](#footnote-1). Zombierne er lavet af “Supercyan” og hedder “Character Pack: Zombie Sample”[[2]](#footnote-2). Våbende er lavet af “Simon Mercuzot” og hedder “Low Poly Weapons VOL.1”[[3]](#footnote-3). Playeren er lavet af “Supercyan” og hedder “Character Pack: Free Sample”[[4]](#footnote-4). Det har fungeret super fint med at finde graphics og implementere dem. Lyd fra Pixabay og Minecraft . **15000 Tegn**

1. <https://assetstore.unity.com/packages/3d/vegetation/environment-pack-free-forest-sample-168396> [↑](#footnote-ref-1)
2. <https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/character-pack-zombie-sample-131604> [↑](#footnote-ref-2)
3. <https://assetstore.unity.com/packages/3d/props/guns/low-poly-weapons-vol-1-151980> [↑](#footnote-ref-3)
4. <https://assetstore.unity.com/packages/3d/characters/humanoids/character-pack-free-sample-79870> [↑](#footnote-ref-4)