Ekskams Projekt Programering - B

HTX ROSKILDE TEKNISK SKOLE – 3.4Et billede, der indeholder rød, orange

Automatisk genereret beskrivelse

6. maj 2022

Jakkarin C. Anthonsen

# Synopsis

Projektet er der lavet en AI indeni en FPS-videospil Prototype, som er lavet med hovedformålet at lave et program med en eller anden form for automatiseret proces. Ideen er at tilføje en slags form for en fjende NPC, som kan går efter spilleren.

Prototypen er udformet sådan at reagere efter hvordan spilleren interagere sig indeni spillet.

Dette er skrevet i C-Sharp script, og der bruges Unity som Game Engine.

Projektet er et eksamensprojekt i programmering B for app og model-udviklings klassen i Roskilde Teknisk Skole.

Indholdsfortegnelse

[1 Synopsis 1](#_Toc103239472)

[3 Indledning 2](#_Toc103239473)

[4 Problemformulering 3](#_Toc103239474)

[5 Programbeskrivelse 3](#_Toc103239475)

[6 Prototypens Funktionalitet 4](#_Toc103239476)

[7 Brugergrænseflader 4](#_Toc103239477)

[7.1.1 Spilleren 4](#_Toc103239478)

[8 Planlægning og opsætning 5](#_Toc103239479)

[8.1 MoSCoW Diagram 5](#_Toc103239480)

[8.2 Rutediagrammer 6](#_Toc103239481)

[9 Udvalgt Kode 7](#_Toc103239482)

[9.1 AI/Fjende 7](#_Toc103239483)

[9.1.1 Patrolling 9](#_Toc103239484)

[9.1.2 ChasePlayer 10](#_Toc103239485)

[9.1.3 AttackPlayer 10](#_Toc103239486)

[9.2 Fjende-Spawner 12](#_Toc103239487)

[10 Test af programmet 13](#_Toc103239488)

[11 Konklusion 13](#_Toc103239489)

[12 Bilag 14](#_Toc103239490)

[13 Referencer 15](#_Toc103239491)

# Indledning

I dette projekt er der lavet en prototype af et FPS-spil. Projektets fokus har været at implementere en fjende AI.

Hovedmålet for dette projekt var været at programmet skulle automatiseres.

Spil prototypen er lavet indeni et Game Engine kaldt for Unity. Hele programmet som bliver, vist er skrevet i C-Sharp/C#.

# Problemformulering

Hvordan kan man opbygge en fjende AI indeni et FPS-videospil, der kan går efter spilleren?

Hvordan kan man lave koden, så AI’en kan gå rundt tilfældigt når den ikke kan se spilleren og løbe efter spilleren nå de er tæt nok.

# Programbeskrivelse

Programmet er en prototype af en ”FPS Zombie” skyde spil.

Her bliver der udforsket på hvordan man kan implementere en AI, som skal fungere som en fjende. I dette projekt har der været fokus på hvordan det er muligt at kunne lave et program som er automatiseret.

I spillet går ud på at skyde zombier, som hele tiden kommer efter spilleren. Spilleren har nogle bestemte våben de kan bruge. Zombierne spawner hele tiden endeløst. Spillet handler om at kunne overleve i så lang tid nå muligt.

Dog er zombierne i spillet kun en rød kapsel som går rundt indenfor spil området.

Et billede, der indeholder rød, orange

Automatisk genereret beskrivelse

# Prototypens Funktionalitet

I spil prototypen har vi brugeren som er spilleren indeni spillet og AI’en som er den NPC i spillet.

Spilleren har den mulighed at de kan bevæge sig omkring spil området og skyde rundt.

Fjende AI’en skal kunne spawne indeni spil forskellige punkter i spil området og lede efter spilleren.

Når AI’en er tæt nok på spilleren skal vi løbe efter spilleren og angribe dem hvis de er tæt nok på.

Spilleren kan dræbe AI’en ved at skyde på dem.

# Brugergrænseflader

### Spilleren

Et billede, der indeholder tekst, himmel, udendørs

Automatisk genereret beskrivelse

Disse er interaktionsmulighederne for brugeren.

Interaktionerne er baseret på hvad man typisk ville kunne i de fleste FPS spil. Interaktions muligheder for brugen er defineret ud denne tabel med Input og Output.

|  |  |
| --- | --- |
| Input | Output |
| WASD | Gå frem, tilbage, venstre, højre |
| Shift | Løbe |
| Ctrl | Dykke |
| Mouse0 eller Mus klik venstre | Skyde |

# Planlægning og opsætning

Dette er den planlagte funktioner i spillet i med at bagtænke om hvad der var relevant og opnåelig.

## MoSCoW Diagram

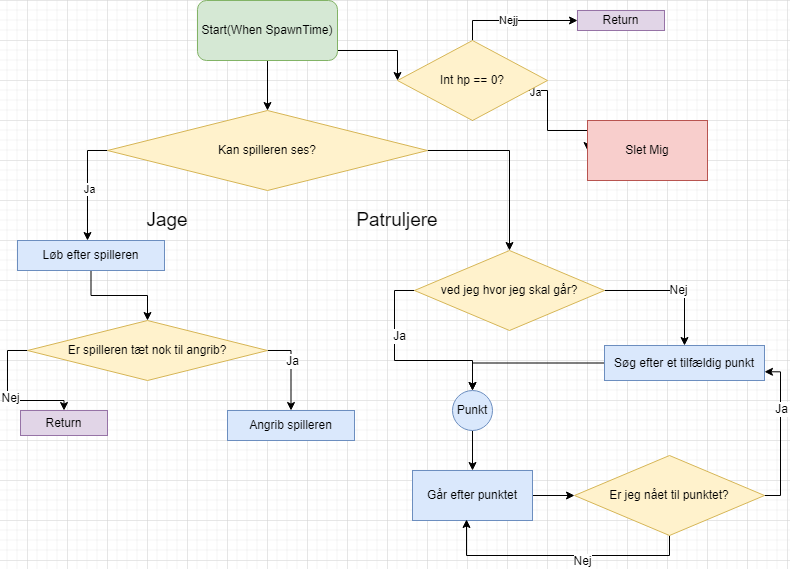
MoSCoW diagrammet blev brugt til at identificere hvilke dele at programmet som var essentielt i forhold til den udførsel der kræves for dette projekt.

metoden er et projektstyrings-værktøj, til prioritering of projekter og andre former for opgaver. Derfor var dette praktisk til planlægning for dette projekt.

## Rutediagrammer

Igennem processen har der været overvejelser for hvordan koden skal opbygges.

Dette er en visuel repræsentation af AI’en opførsel. Hvordan AI’en opfør sig omhandler på hvor tæt spilleren er på den. Rutediagram er lavet i ” app.diagrams.net”, som er et gratis program lavet til at lave forskellige diagrammer som dette.



# Udvalgt Kode

Disse er de udvalgte koder i programmet.

## AI/Fjende

Fjende AI’et er der lavet et script, som er tilføjet til AI’et.

Her bruges ’UnityEngine.AI”. UnityEngine.AI er moderklassen for NavMesh-relaterede klasser, så den bruges til at flytte Nav-agenter rundt i en scene.

Derefter er de nødvendige variabler tilføjet. AI’et har 3 hoved funktioner, ”Patroling”, ”Chase” & ”Attack”

|  |
| --- |
| using UnityEngine;  using UnityEngine.AI;  public class EnemyAI : MonoBehaviour  {  public NavMeshAgent agent;  public Transform player;  public LayerMask whatIsGround, whatIsPlayer;  //Patroling  public Vector3 walkPoint;  bool walkPointSet;  public float walkPointRange;  //Attack  public float timeBetweenAttack;  bool alreadyAttacked;  //States  public float sightRange, attackRange;  public bool playerInSightRange, playerInAttackRange |

Dette Awake funktion er for at Programmet identificere hvad spilleren er og hvilke platform AI’et fjenden kan gå i.

|  |
| --- |
| private void Awake()  {  player = GameObject.Find("PlayerObj").transform;  agent = GetComponent<NavMeshAgent>();  } |

For dette ”Update” Funktion tjekker fjenden hele hvilke State fjenden er i.

Der er skrevet 3 if statements.

* Fjenden er i ”patrol” man ikke kan se spilleren.
* Fjenden Løber efter spilleren når de kan se dem.
* Fjenden angriber spilleren, når er tæt nok på.

Disse ’if’ statements aktivere specifikke void funktioner som bliver forklaret lidt senere.

Der 2 funktioner som identificerer om Fjenden er tæt når på til at kunne se spilleren og når de er tæt nok til at angribe.

|  |
| --- |
| // Update is called once per frame  void Update()  {  // Check for sight and attack range.  playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);  playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);  //The player is not in sight or in range. Enemy Patrolling  if (!playerInSightRange && !playerInAttackRange) Patroling();  // The player is in sight, but not in attack range, Enemy Chase the player.  if (playerInSightRange && !playerInAttackRange) ChasePlayer();  //The player is in sight and in attack range, Enemy attack.  if (playerInSightRange && playerInAttackRange) AttackPlayer();  } |

### Patrolling

Dette er funktionen for ”Patrolling”. AI’et går efter en tilfældig retning for at lede efter spilleren. I denne funktion er der 3 if statements.

Hvis funktionen ik allerede har en pointen den går hen til, så skal den søge efter en tilfældig position, ved at køre ”SearchWalkPoint”, som er en anden void funkion der er lavet uden for dette.

Når den har fundet en position, så vil AI’en går efter den position. Den gør det ved at bruge NavMeshAgent, som er en Component i Unity brugt til at bevæge en spille objekt til en bestemt position.

Efter AI’en er gået kommet til den position, sætter den walkPointSet boolean til at være falsk for at genstarte funktionen.

|  |
| --- |
| private void Patroling()  {  if (!walkPointSet) SearchWalkPoint();  if (walkPointSet)  agent.SetDestination(walkPoint);  Vector3 distanceToWalkPoint = transform.position - walkPoint;  //Walkpoint reached  if (distanceToWalkPoint.magnitude < 1f)  walkPointSet = false;  } |

Måden programmet beregner den tilfældige position AI’en kan gå hen til, bruges en anden funktion ’SeachWalkPoint’. distancen bliver defineret som en ”WalkPointRange” for at det er muligt at justere den.

|  |
| --- |
| private void SearchWalkPoint()  {  //Calculate random point in range.  float randomZ = Random.Range(-walkPointRange, walkPointRange);  float randomX = Random.Range(-walkPointRange, walkPointRange);  walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);  if (Physics.Raycast(walkPoint, -transform.up, 2f, whatIsGround))  walkPointSet = true;  } |

### ChasePlayer

Når AI’en skal går efter spilleren, bliver definitionen af den position den skal gå efter sat til at være spillerens position.

|  |
| --- |
| //Chase the player  private void ChasePlayer()  {  agent.SetDestination(player.position);  } |

### AttackPlayer

Fjenden kan kun aktiveres når den er tæt nok på spilleren, som bliver vurderet ved Update funktionen.

Fjenden vil kigge på spilleren imens den stadig vil går efter spilleren.

For at fjenden kun kan angribe for et bestemt tidsrum, er der lavet en boolean ’AlreadtAttacked’, dette bliver sandt når den har angribet og bliver genstartet via ’ResetAttack’ funktionen.

Hvornår den genstartes bliver defineres via invoke funtionen. Tiden for hvert angreb bliver defineret via variablet ’TimeBetweenAttack’, som på den måde kan man justere på tiden.

|  |
| --- |
| private void AttackPlayer()  {  agent.SetDestination(transform.position);  transform.LookAt(player);  if (!alreadyAttacked) // An attack here that could be anything.  {  //  ///Todo Enemy Punch  //  alreadyAttacked = true;  Invoke(nameof(ResetAttack), timeBetweenAttack);  }  }  // Resetting the attack  private void ResetAttack()  {  alreadyAttacked = false;  } |

For at det er muligt for spilleren at dræbe AI’en blev der lavet et andet script til at styre det. Ideen var at udvide koden mere, som er derfor dette blev tilføjet i stedet for kode det I det samme script.

Her er der lavet et simpelt script som får fjenden til at forsvinde efter spilleren af dræbt AI’en.

Der er lavet en float, som den variabel for deres HP(health). Og en ’if’ statement for hvis den falder ned til 0, så vil Unity slette spil objektet.

|  |
| --- |
| public float health = 50f;  public void TakeDamage(float amount)  {  health -= amount;  if (health <= 0f)  {  DestroyEnemy();  }  }  private void DestroyEnemy()  {  Destroy(gameObject);  } |

## Fjende-Spawner

Dette script er tilføjet til et tomtspil objekt, kaldt for ”SpawnManager”. Der har været planlagt at AI’en kom efter spilleren i forskellige bølger.

Scriptet spawner fjenderne efter tid, i forskellige bestemte positioner. Positionerne er defineret ved at lave forskellige tomme spilobjekter og referere til dem.

Denne funktion bliver hele tiden kørt i Update funktionen.

Først er bliver der defineret at spawn punkterne er tilfældige ved ”RandomSpawnSpot”. Dette bliver gjort ved at bruge ”Random.range”.

Derefter bliver der brugt en ’if’ statement for at hvis tiden er lige med et bestemt interval, skal den spawne fjenderne.

Spillet spawner fjenderne spawner fjender med brug af ”Instantiate”, hvor fjenden bliver defineret som en spil objekt, og den valgte position som er tilfældig.

Efter spillet er spawnet en fjende bliver timeren genstartet igen til nul.

|  |
| --- |
| void SpawningEnemy()  {  randomSpawnSpot = Random.Range(0, spawnSpots.Length);  if (timer >= SpawnTimeInSecs)  {  x = Random.Range(0, maxX);  Instantiate(Enemy, spawnSpots[randomSpawnSpot].transform);  randomSpawnSpot++;  timer = 0;  }  } |

# Test af programmet

For at teste programmet er der sammenlignet den forventede output den resulteret output.

Dette er et udvidet skema fra brugergrænse overfladen. Skemaet tester dog kun det spilleren kan gøre.

|  |  |  |
| --- | --- | --- |
| Bruger Input | Forventet Output | Faktisk Output |
| WASD |  |  |
| Ctrl | Spilleren dukker sig | Spilleren Dukker sig, men figuren bliver også mindre. |
| Venstre Mus klik | Spilleren skyder, man kan se ”Muzzle Flash” og hvor hende skudet bliver ramt. | Spilleren skyder, og rammer for det meste fjenden men ikke altid |
| Shift | Spilleren Løber |  |

Det kan stadig ses når brugeren skyder efter AI’en fra en lang distance, er det ikke altid at skuet rammer.

# Perspektivering

En ting som kunne være optimeret har været koden, da kan i stedet kunne bruge en Enum, så det måske er nemmere at udvikle AI’en ved at tilføje flere funktioner eller justering til dem.

Der har også været overvejelser at kode AI’en Patch finding selv, dog ville dette være et anden projekt og besværliggøre processen.

I denne version af prototypen er der valgt at sætte fokus på de basale funktioner der krævede ud fra MoSCoW diagrammet.

Prototypen kunne der sikker være lavet pænere og lave nogle flere funktioner til at gøre brugerens game feel bedre. Eksempelvis flere bevægelser i våbenet når vi bevæger sig eller andre ting som gør det tydeligt at brugeren har et våben.

En anden overvejelse har også været at kan kunne implementere en slags bølge system noget liggende til ”Call of Duty – Zombies” hvor der kommer flere zombier efter hvert runde.

Der kunne også implementere flere variationer af fjender, dog var hovedparten af implementere en AI.

# Konklusion

Formålet har været at lave et automatiseret program. Målet har været at lave en form for AI som der opfør sig ud fra spilleren. Tingene fungerer og AI’en kan interageres med brugeren, og følger funktionerne som er skrevet. Spilleren kan nu også dræbe AI’en ved at skyde den.

For at gøre prototypen til at ligne mere en spil, kunne der måske har brugt mere tid på dette, men det vurderes at der ikke ville være store arbejde eller relevant for dette projekt.

Dog vil prototypen have sinde begrænsninger da der kun har været fokus på hvad der skulle implementeres. Derfor har spil prototypen ikke en reel start eller en slut, som der kræves på en videospil.

Projektet kan derfor konkluderes som en succes.

# Bilag

Programmet kan hentes i Github ved dette link:

<https://github.com/jayc500045/Last_Programming_Project_Unity>

## Bilag 1 – Script i Enemy AI

using UnityEngine;

using UnityEngine.AI;

public class EnemyAI : MonoBehaviour

{

public NavMeshAgent agent;

public Transform player;

public LayerMask whatIsGround, whatIsPlayer;

//Patroling

public Vector3 walkPoint;

bool walkPointSet;

public float walkPointRange;

//Attack

public float timeBetweenAttack;

bool alreadyAttacked;

//States

public float sightRange, attackRange;

public bool playerInSightRange, playerInAttackRange;

// Start is called before the first frame update

private void Awake()

{

player = GameObject.Find("PlayerObj").transform;

agent = GetComponent<NavMeshAgent>();

}

// Update is called once per frame

void Update()

{

// Check for sight and attack range.

playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);

playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);

//The player is not in sight or in range. Enemy Patrolling

if (!playerInSightRange && !playerInAttackRange) Patroling();

// The player is in sight, but not in attack range, Enemy Chase the player.

if (playerInSightRange && !playerInAttackRange) ChasePlayer();

//The player is in sight and in attack range, Enemy attack.

if (playerInSightRange && playerInAttackRange) AttackPlayer();

}

//Patrolling

private void Patroling()

{

if (!walkPointSet) SearchWalkPoint();

if (walkPointSet)

agent.SetDestination(walkPoint);

Vector3 distanceToWalkPoint = transform.position - walkPoint;

//Walkpoint reached

if (distanceToWalkPoint.magnitude < 1f)

walkPointSet = false;

}

private void SearchWalkPoint()

{

//Calculate random point in range.

float randomZ = Random.Range(-walkPointRange, walkPointRange);

float randomX = Random.Range(-walkPointRange, walkPointRange);

walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);

if (Physics.Raycast(walkPoint, -transform.up, 2f, whatIsGround))

walkPointSet = true;

}

//Chase the player

private void ChasePlayer()

{

agent.SetDestination(player.position);

}

private void AttackPlayer()

{

agent.SetDestination(transform.position);

transform.LookAt(player);

if (!alreadyAttacked) // An attack here that could be anything.

{

//

///Todo Enemy Punch

//

alreadyAttacked = true;

Invoke(nameof(ResetAttack), timeBetweenAttack);

}

}

// Resetting the attack

private void ResetAttack()

{

alreadyAttacked = false;

}

}

## Bilag 2 – Script i Enemy Spawner

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemySpawner : MonoBehaviour

{

// Start is called before the first frame update

[Header("Refferences")]

public Transform[] spawnSpots;

public GameObject Enemy;

private int randomSpawnSpot;

private Transform PagePos;

[Header("Zombie Spawning calcs")]

public int SpawnTimeInSecs = 5;

public int maxX = 10; //Max Enemy in game.

float timer = 0;

int x = 0;

void Start()

{

//The Timer always starts with 0.

timer = 0;

randomSpawnSpot = Random.Range(0, spawnSpots.Length);

}

// Update is called once per frame

void Update()

{

timer += Time.deltaTime;

SpawningEnemy();

}

void SpawningEnemy()

{

randomSpawnSpot = Random.Range(0, spawnSpots.Length);

if (timer >= SpawnTimeInSecs)

{

x = Random.Range(0, maxX);

Instantiate(Enemy, spawnSpots[randomSpawnSpot].transform);

randomSpawnSpot++;

timer = 0;

}

}

}

# Referencer

Disse er brugte kilder for hjælp og referencer til at strukturere koden.

Spawner using empty objects

<https://forum.unity.com/threads/spawning-object-at-gameobjecs-position.794232/>

How to spawn something within a time

<https://answers.unity.com/questions/1502490/how-to-make-a-spawner.html>

Enemy AI

<https://www.youtube.com/watch?v=UjkSFoLxesw>

How to make all kinds of guns

<https://www.youtube.com/watch?v=bqNW08Tac0Y&t=206s>

First person movement

<https://www.youtube.com/watch?v=f473C43s8nE>