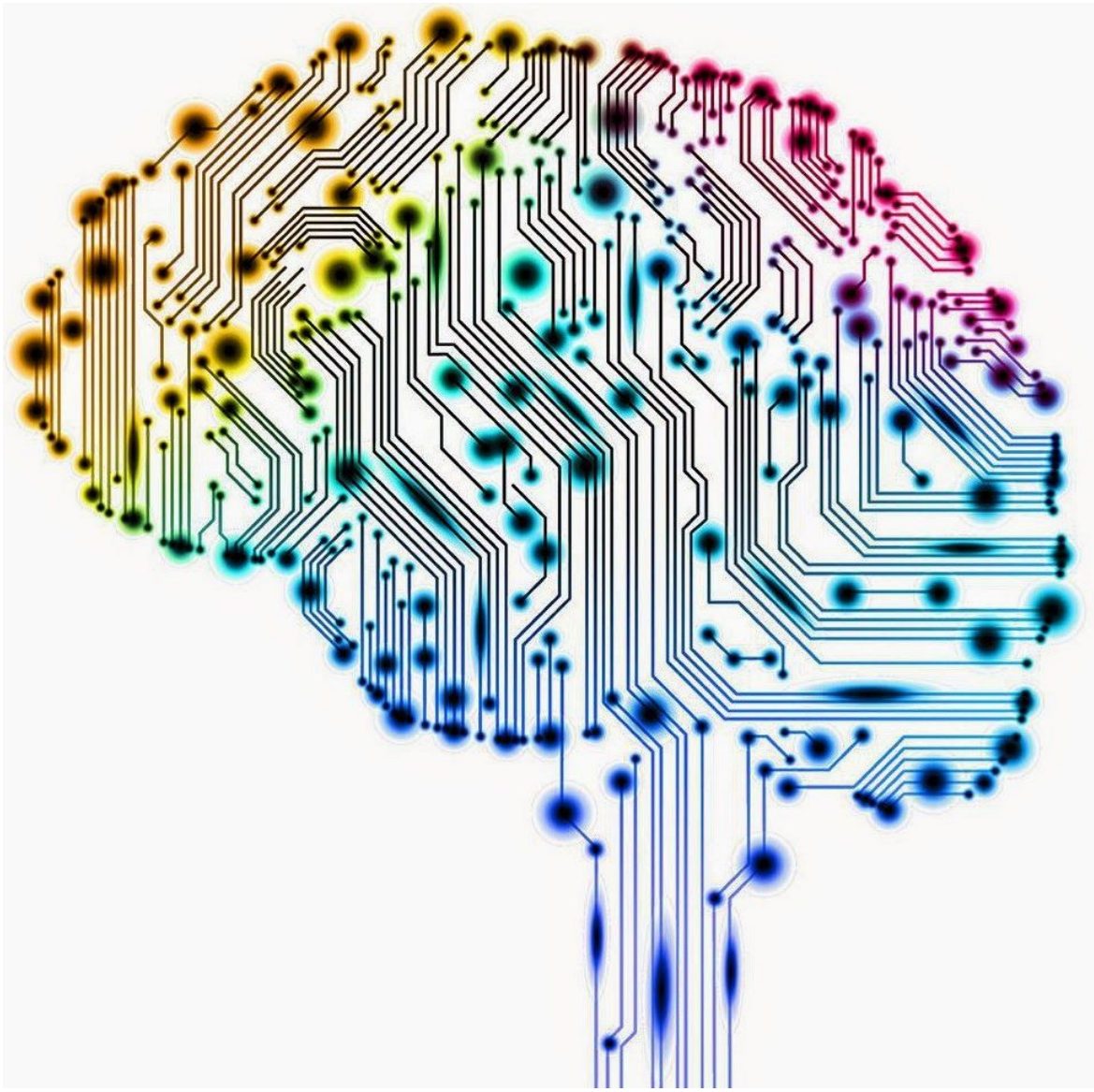
Game Development

Kernmodule 3: Artificial Intelligence

Jesse van Vliet
3012446



Docenten: V. Muijrers en S. Alkemade
Datum: 2 april 2017

Inhoud

Opdrachten

Boids

Includes files: Boid.cpp - Flock.cpp - Fractal.cpp - Weapon.cpp

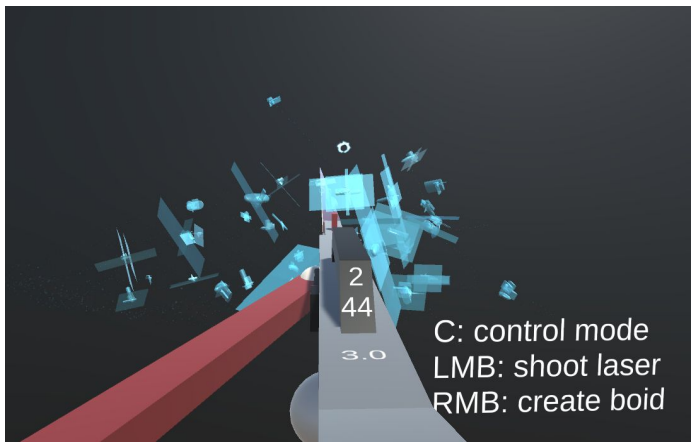
Concept

Een simpele interactieve scene waarbij je met wapen die een laser schiet interacteert met boids in de scene

- De boids vluchten van jouw laser.
- De boids vluchten van exploderende boids.
- De boids zijn bestuurbaar omdat de speler overduidelijk de macht heeft over alles in de game.
- Het zijn fractals.

De basis 3 boid regels zijn toegepast met 2 extra regels.

- Rule 1: Find Center of Mass and move towards.
- Rule 2: Don't hit other boids.
- Rule 3: Match velocity.
- Rule 4: Move boids based on player control.
- Rule 5: Run away from exploding boids and the lasers.



Sources & helpful links:

- http://www.kfish.org/boids/pseudocode.html
- https://processing.org/examples/flocking.html

Code (Unity - C#)

```
private Vector3 Rule1() {
    Vector3 centerOfMass = Vector3.zero;

    for (int i = 0; i < flock.boids.Length; i++) {
        float dist = Vector3.Distance(flock.boids[i].transform.localPosition, this.transform.localPosition);

        if (dist > 0 && dist < maxDist) {
            centerOfMass += flock.boids[i].transform.localPosition;
        }
    }

    return ((centerOfMass / (flock.boids.Length - 1)) - this.transform.localPosition).normalized;
}
```

```
private Vector3 Rule2() {
    Vector3 velocity = Vector3.zero;

    for (int i = 0; i < flock.boids.Length; i++) {
        float dist = Vector3.Distance(flock.boids[i].transform.localPosition, this.transform.localPosition);

        if (dist > 0 && dist < maxDist) {
            velocity += flock.boids[i].GetComponent<Rigidbody>().velocity;
        }
    }
    return (velocity / (flock.boids.Length - 1)).normalized;
}
```

```
private Vector3 Rule3() {
    Vector3 velocity = Vector3.zero;

    for (int i = 0; i < flock.boids.Length; i++) {
        float dist = Vector3.Distance(flock.boids[i].transform.localPosition, this.transform.localPosition);

        if (dist > 0 && dist < maxDist) {
            velocity -= (flock.boids[i].transform.localPosition - this.transform.localPosition).normalized / dist;
        }
    }
    return (velocity / (flock.boids.Length - 1)).normalized;
}
```

```
private Vector3 Rule4() {
    Vector3 addingVector = center.transform.position;
    return addingVector;
}
```

```
private Vector3 Rule5() {
    Vector3 moveTo = Vector3.zero;

    if (hideFrom != null) {
        for (int i = 0; i < hideFrom.Length; i++) {
            Vector3 dirTo = this.transform.position - hideFrom[i].transform.position;
            dirTo = dirTo.normalized;
            moveTo = dirTo;
        }
    }
    return moveTo * 10f;
}
```

Code (Unreal Engine - C++)

```
FVector ABoid::RuleOne() {
    FVector CenterOfMass = FVector();

    for (TActorIterator<ABoidsList> ActorItr(GetWorld()); ActorItr; ++ActorItr) {
        ABoidsList *BL = *ActorItr;

        for (int i = 0; i < BL->BoidsList.Num(); i++) {

            FVector distVector = FVector();

            float dist = distVector.Distance(BL->BoidsList[i]->GetActorLocation(), GetActorLocation());

            if (dist > 0 && dist < mDist) {
                CenterOfMass += BL->BoidsList[i]->GetActorLocation();
            }
        }
        CenterOfMass = (CenterOfMass / (BL->BoidsList.Num() - 1)) - GetActorLocation();
    }
    return CenterOfMass.GetSafeNormal();
}
```

```
FVector ABoid::RuleTwo() {
    FVector Vc = FVector();

    for (TActorIterator<ABoidsList> ActorItr(GetWorld()); ActorItr; ++ActorItr) {
        ABoidsList *BL = *ActorItr;

        for (int i = 0; i < BL->BoidsList.Num(); i++) {

            FVector distVector = FVector();

            float dist = distVector.Distance(BL->BoidsList[i]->GetActorLocation(), GetActorLocation());

            if (dist > 0 && dist < mDist) {
                Vc += BL->BoidsList[i]->GetVelocity();
            }
        }
        Vc = (Vc / (BL->BoidsList.Num() - 1));
    }
    return Vc.GetSafeNormal();
}
```

```
FVector ABoid::RuleThree() {
    FVector Vc = FVector();

    for (TActorIterator<ABoidsList> ActorItr(GetWorld()); ActorItr; ++ActorItr) {
        ABoidsList *BL = *ActorItr;

        for (int i = 0; i < BL->BoidsList.Num(); i++) {

            FVector distVector = FVector();

            float dist = distVector.Distance(BL->BoidsList[i]->GetActorLocation(), GetActorLocation());

            if (dist > 0 && dist < mDist) {
                FVector Temp = FVector();
                Temp = (BL->BoidsList[i]->GetActorLocation() - GetActorLocation()).GetSafeNormal();
                Vc -= Temp / dist;
            }
        }
        Vc = (Vc / (BL->BoidsList.Num() - 1));
    }
    return Vc.GetSafeNormal();
}
```

Procedural Generation

Includes files:

- AI:
    - FSM.cs - IState.cs - State.cs (FSM + State base + State interface)
        - State_Attack.cs (State implementation)
        - State_Defend.cs (State implementation)
        - State_Dodge.cs (State implementation)
        - State_FindOpponent.cs (State implementation)
        - State_FindState.cs (State implementation)
        - State_Flee.cs (State implementation)
        - State_MoveToOpponent.cs (State implementation)
            - SetAttacking.cs (Animator state script)
            - SetDefending.cs (Animator state script)
- Level:
    - ArenaGenerator.cs (Generates arena)
    - MapGenerator.cs (Generates world)
    - ObjectToFloor.cs (Sets arena objects to world floor)
- Camera:
    - LookAtCamera.cs (TextMeshes looking at camera)
    - MoveCamera.cs (Cinematic styled movement)
- Swords:
    - CreateSwords.cs (Creates an amount of swords)
    - WeaponGeneration.cs (Sword Gen for showcase)
    - WeaponGenerationForGameplay.cs (Sword Gen for arena)
- Databases:
    - SwordDatabase.cs (Database of models/materials, values)
    - NameDatabase.cs (Database of names, generating names)
- Other:
    - MoneyManager.cs (Gameplay)
    - SetOnBtnClickValues.cs (Gameplay)
    - MoveToClickPoint.cs (Debug)
    - SetCullingModeFix.cs (Animator rendering fix)
    - SceneManaging.cs (User interface)

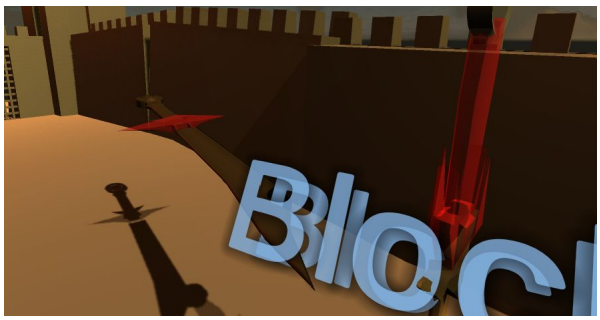
Concept

Een cinematic arena styled brawler en betting game waarbij alles procedurally generated wordt. Een soort gladiator style pit fight.

Random werelden, Random arena's en random zwaarden met randomized stats gebaseerd op de models/materials waarvan de sword gemaakt is.

Gemixt met een computer ai die de swords bestuurd en tegen elkaar op laat vechten.

Het resultaat is een awesome random arena fighting game waar je op de computers kan betten en rijk worden!



Er is een 2e versie beschikbaar met LeapMotion support voor camera controls.

[YouTube link naar sword fighting met LeapMotion.]

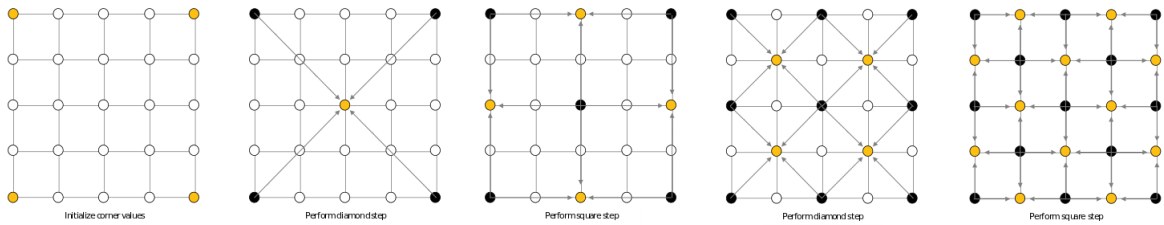
Code & algorithmes

World Generation algorithme: Diamond Square (Pagina 7 t/m 10)

Zelf vond ik het niet zo interessant om een tutorial te gaan volgen van het internet en misschien lichtelijk de code aanpassen of iets dergelijks. Dus ik heb het zoveel mogelijk gedaan met theorie en zelf toepassen van het algoritme.

Ik heb wel een tutorial gevolgd *(de unity documentation + een answers.unity3d link)* over een mesh aanmaken in runtime d.m.v. code.

Het diamond square algoritme wordt erg goed uitgelegd met screenshots en dit was voornamelijk waar ik naar heb gekeken met veel tijd om te proberen met code.



In MapGenerator.cs in het project wordt de volledige code vertoond:

https://github.com/Thovex/Artificial-Intelligence/blob/master/Procedural%20Generation%20Arena%20Game/Assets/Scripts/Level/MapGenerator.cs

De volgorde van terrain generation:

```
private void CreateTerrain() {
    GetVertexCount();
    InitVertexArrays();
    CalculateVertsUvsTris();
    SetEdgeVerts();
    PerformDiamondSquare();
    SetMesh(CreateMesh());
}
```

De totale vertex count berekenen (based op hoe vaak je de mesh divide).

```
private void GetVertexCount() {
    worldVertCount = (worldDivisions + 1) * (worldDivisions + 1);
}
```

De vertex arrays, uvs en tri's initialiseren zodat deze vervolgens verandert kan worden.

```
private void InitVertexArrays() {
    worldVerts = new Vector3[worldVertCount];
    uvs = new Vector2[worldVertCount];
    tris = new int[worldDivisions * worldDivisions * 6];
}
```

Het bepalen van alle vertices, UVs en SetTris() aanroepen.

```
private void CalculateVertsUvsTris() {
    float halfSize = worldSize * 0.5f;
    float divisionSize = worldSize / worldDivisions;

    int triOffset = 0;

    for (int i = 0; i <= worldDivisions; i++) {
        for (int j = 0; j <= worldDivisions; j++) {
            worldVerts[i * (worldDivisions + 1) + j] = new Vector3(-halfSize + j * divisionSize, 0.0f, halfSize - i * divisionSize);
            uvs[i * (worldDivisions + 1) + j] = new Vector2((float)i / worldDivisions, (float)j / worldDivisions);
            if (i < worldDivisions && j < worldDivisions) {
                int topLeft = i * (worldDivisions + 1) + j;
                int botLeft = (i + 1) * (worldDivisions + 1) + j;
                SetTris(triOffset, topLeft, botLeft);
                triOffset += 6;
            }
        }
    }
}
```

Het aanmaken en bepalen van de tris in de quads.

```
private void SetTris(int triOffset, int topLeft, int botLeft) {
    tris[triOffset] = topLeft;
    tris[triOffset + 1] = topLeft + 1;
    tris[triOffset + 2] = botLeft + 1;
    tris[triOffset + 3] = topLeft;
    tris[triOffset + 4] = botLeft + 1;
    tris[triOffset + 5] = botLeft;
}
```

De buitenste edge vertices bepalen. En vervolgens hun height randomizen.

```
private void SetEdgeVerts() {
    worldVerts[0].y = Random.Range(-worldHeight, worldHeight);
    worldVerts[worldDivisions + 1].y = Random.Range(-worldHeight, worldHeight);
    worldVerts[worldVerts.Length - 1].y = Random.Range(-worldHeight, worldHeight);
    worldVerts[worldVerts.Length - 1 - worldDivisions].y = Random.Range(-worldHeight, worldHeight);
}
```

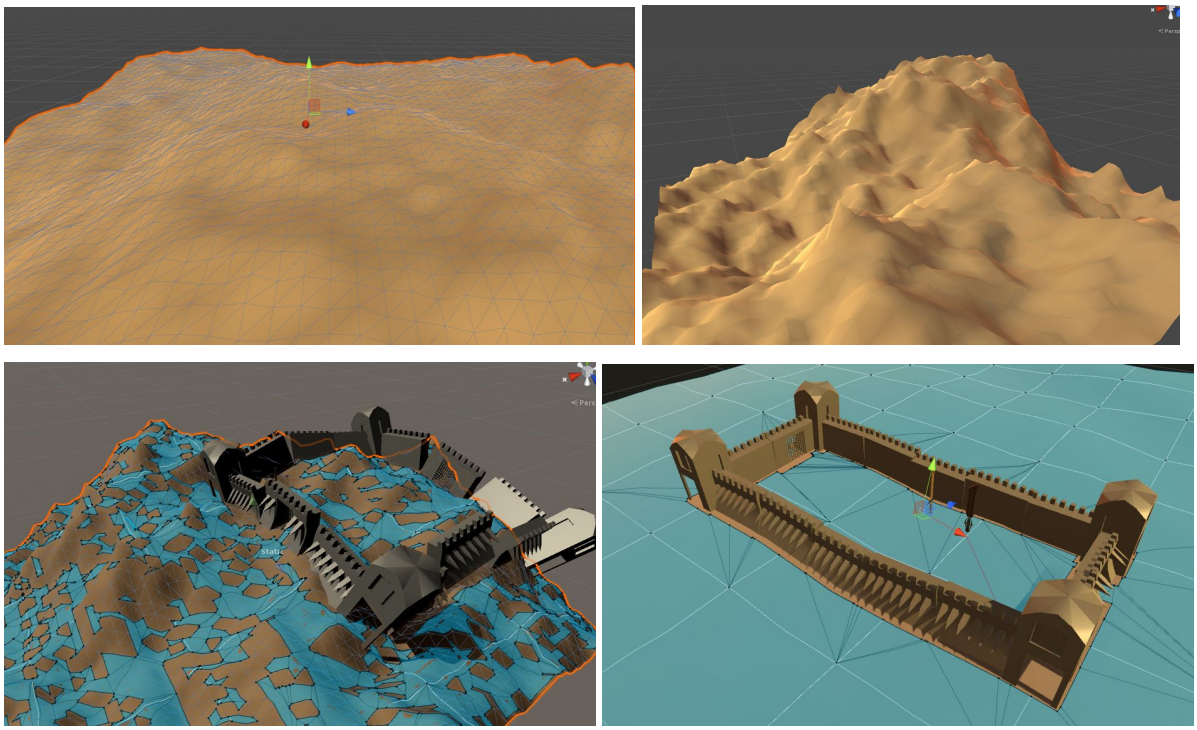
De “square” functie van de diamond square aanroepen. Hier krijg je rijen en kolommen van squares in je mesh waar vervolgens de diamond square op wordt aangeroepen - om vervolgens de square weer te herhalen tot het einde (het aantal mesh divisions bepaalt het einde).

```
private void PerformDiamondSquare() {
    int iterations = (int)Mathf.Log(worldDivisions, 2);
    int numSquares = 1;
    int squareSize = worldDivisions;

    for (int i = 0; i < iterations; i++) {
        int row = 0;
        for (int j = 0; j < numSquares; j++) {
            int col = 0;
            for (int k = 0; k < numSquares; k++) {
                PerformDiamond(row, col, squareSize, worldHeight);
                col += squareSize;
            }
            row += squareSize;
        }
        numSquares *= 2;
        squareSize /= 2;
        worldHeight *= 0.5f;
    }
}
```

De diamond functie uitvoeren op een square in de mesh.

```
private void PerformDiamond(int row, int col, int size, float offset) {
    int halfSize = (int)(size * 0.5f);
    int topLeft = row * (worldDivisions + 1) + col;
    int botLeft = (row + size) * (worldDivisions + 1) + col;

    int mid = (row + halfSize) * (worldDivisions + 1) + col + halfSize;
    worldVerts[mid].y = (worldVerts[topLeft].y + worldVerts[topLeft + size].y + worldVerts[botLeft].y + worldVerts[botLeft + size].y) * .25f + Random.Range(-offset, offset);

    worldVerts[topLeft + halfSize].y = (worldVerts[topLeft].y + worldVerts[topLeft + size].y + worldVerts[mid].y) / 3 + Random.Range(-offset, offset);
    worldVerts[mid - halfSize].y = (worldVerts[topLeft].y + worldVerts[botLeft].y + worldVerts[mid].y) / 3 + Random.Range(-offset, offset);
    worldVerts[mid + halfSize].y = (worldVerts[topLeft + size].y + worldVerts[botLeft + size].y + worldVerts[mid].y) / 3 + Random.Range(-offset, offset);
    worldVerts[botLeft + halfSize].y = (worldVerts[botLeft].y + worldVerts[botLeft + size].y + worldVerts[mid].y) / 3 + Random.Range(-offset, offset);
}
```

Result:



Een wereld die customizable is in breedte, lengte, hoogte en de hoeveelheid. Ook heb ik zelf uitgespit hoe in de (toen: unity 5.6 beta) de nieuwe navmesh generation werkt during realtime en heb daar mee geëxperimenteerd en succesvol toegepast.

Sources:

- https://en.wikipedia.org/wiki/Diamond-square_algorithm
- http://stackoverflow.com/questions/2755750/diamond-square-algorithm
- http://answers.unity3d.com/questions/139808/creating-a-plane-mesh-directly-from-code.html

Arena Generation: by Vector2 (Pagina 11 t/m 12)

Dit was een iets ander soort “random generation” er wordt wel specifiek gekeken waar alles gespawnd gaat worden en zorgen dat er altijd 2 gates en 4 towers zijn. Maar de size van de arena heeft geen limiet verder.

Veel instantiation, altijd 2 gates, altijd 4 towers en unlimited walls in between met de correcte posities en rotaties, dit wordt berekend door mesh size - dus de walls kunnen ook grotere of kleinere models krijgen en dan zou de code gewoon hetzelfde blijven.

```
void CreateBase () {

    for (int i = 0; i < arenaSize.x; i++) {
        if (i == Mathf.FloorToInt(arenaSize.x / 2)) {

            gates[0] = Instantiate(gate, new Vector3(transform.position.x, transform.positio
            gates[1] = Instantiate(gate, new Vector3(transform.position.x - (arenaSize.y * s

        }
        else {
            Instantiate(wall, new Vector3(transform.position.x, transform.position.y, transf
            Instantiate(wall, new Vector3(transform.position.x - (arenaSize.y * sWall + sTow
        }

        if (i == arenaSize.x - 1) {
            Instantiate(tower, new Vector3(transform.position.x, transform.position.y, trans
            Instantiate(tower, new Vector3(transform.position.x, transform.position.y, trans
            Instantiate(tower, new Vector3(transform.position.x - (arenaSize.y * sWall + sTo
            Instantiate(tower, new Vector3(transform.position.x - (arenaSize.y * sWall + sTo
        }
    }

    for (int j = 0; j < arenaSize.y; j++) {

        GameObject g = Instantiate(wall, new Vector3((transform.position.x - (j * sWall + sT
        Instantiate(wall, new Vector3((transform.position.x - (j * sWall + sTower * 1.3f)),

        if (j == Mathf.FloorToInt(arenaSize.y / 2f)) {
            Camera.main.transform.parent = g.transform;
            Camera.main.transform.position = new Vector3(g.transform.position.x, g.transform
            Camera.main.transform.rotation = Quaternion.Euler(new Vector3(25f, -90f, 0f));
        }
    }
}
```

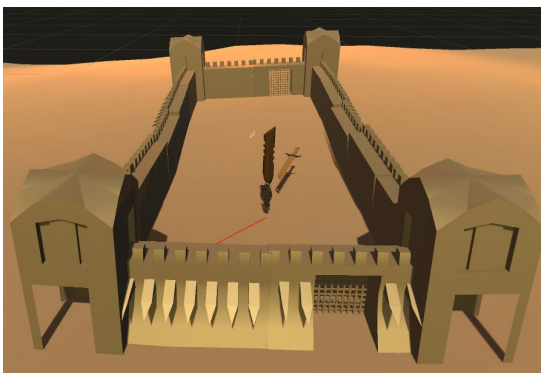
De walls worden perfect gezet aan de hand van de mesh van de wereld zelf door middel van 4 raycasts aan alle zeiden van een muur model, naar beneden gecast tot een hit.point op de mesh. Deze 4 values worden samengevoegd door 4 cross products te normalizen.

```
Vector3 upDir = (
     Vector3.Cross(rayHitBackRight.point - Vector3.up, rayHitBackLeft.point - Vector3.up) +
     Vector3.Cross(rayHitBackLeft.point - Vector3.up, rayHitFrontLeft.point - Vector3.up) +
     Vector3.Cross(rayHitFrontLeft.point - Vector3.up, rayHitFrontRight.point - Vector3.up) +
     Vector3.Cross(rayHitFrontRight.point - Vector3.up, rayHitBackRight.point - Vector3.up)
    ).normalized;

transform.rotation = Quaternion.LookRotation(-upDir, -transform.forward);
transform.Rotate(Vector3.right, 90f);

if (Physics.Raycast(rayMiddle, out rayHitMiddle, Mathf.Infinity)) {
    transform.position = rayHitMiddle.point;
    Destroy(this);
}
```

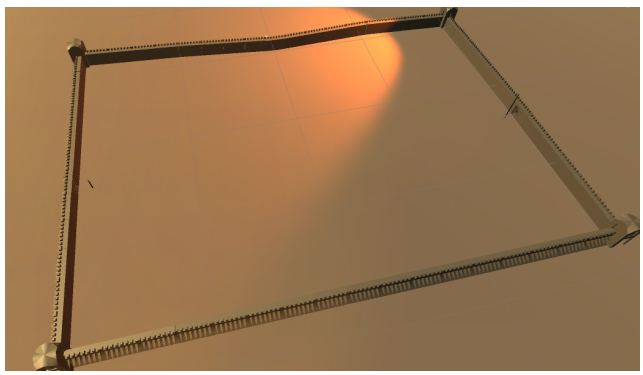
En vervolgens het object snappen aan de vloer beneden.

Result:

(2 x 5)



(12 x 17)



Sword Generation: by database (Pagina 13 t/m 16)

Om het interessant te maken moeten er veel variaties in zwaarden en statistieken zijn. Dit heb ik door middel van een sword database gedaan en vervolgens zwaarden in/aan elkaar gecodeerd.

We starten met een

```
private void Start() {
    StartGeneratingSword();
```

Een sword heeft x aantal onderdelen nodig. Als het een eerste onderdeel is dan wordt het object het base object en kan hij op de (0, 0) spawnen en creëert dit object zijn eigen onderdeel.

```
private void StartGeneratingSword() {
    for (int i = 0; i < SwordDatabase.Instance.swordSize; i++) {
        GameObject aPart;

        if (part.Count < 1) {
            aPart = GeneratePartData(transform.position, gameObject, i);
        } else {
            Vector3 newPos = CalculateTopVertex(part[i - 1]);
            aPart = GeneratePartData(newPos, part[i - 1], i);
        }

        if (Random.value < 0.5f) {
            aPart.transform.rotation = Quaternion.Euler(0, 180, 0);
        }

        part.Add(aPart);
    }
}
```

Het onderdeel creëren wordt hier gedaan, er wordt een mesh en material gekozen voor het specifieke onderdeel (index 0 is een pommel, index 1 is een hilt, enzovoorts.)

```
private GameObject GeneratePartData(Vector3 startPos, GameObject parent, int index) {

    int meshPicker = Random.Range(0, SwordDatabase.Instance.itemMeshes[index].ItemMeshes.Count);
    int materialPicker = Random.Range(0, SwordDatabase.Instance.itemMaterials.Count);

    Mesh selectedMesh = SwordDatabase.Instance.itemMeshes[index].ItemMeshes[meshPicker].mesh;
    Material selectedMaterial = SwordDatabase.Instance.itemMaterials[materialPicker].material;

    GameObject obj = GenerateGameObjectData(selectedMesh, selectedMaterial);
    obj.transform.position = startPos + parent.transform.position + offset;
    obj.transform.parent = parent.transform;

    CustomizeSettings(index, obj);

    materialTypes.Add(selectedMaterial.ToString());

    meshValue += (SwordDatabase.Instance.itemMeshes[index].ItemMeshes[meshPicker].value * obj.tra
    materialValue += SwordDatabase.Instance.itemMaterials[materialPicker].value;

    return obj;
}
```

Dit object pakt zijn eigen models/meshes.

```
private GameObject GenerateGameObjectData(Mesh theMesh, Material theMaterial) {
    GameObject newObj = new GameObject(theMesh.name);
    newObj.AddComponent<MeshFilter>().mesh = theMesh;
    newObj.AddComponent<MeshRenderer>().material = theMaterial;
    return newObj;
}
```

En vervolgens worden er custom settings toegepast.

```
private void CustomizeSettings(int index, GameObject obj) {
    switch (index) {
        case 0: // Pommel Settings
            break;
        case 1: // Hilt Settings
            obj.transform.localScale = new Vector3(obj.trans
            break;
        case 2: // Guard Settings
            obj.transform.localScale = new Vector3(obj.trans
            break;
        case 3: // Sword Settings
            obj.transform.position -= offset * 1.5f;

            obj.transform.localScale = new Vector3(obj.trans

            if (Random.value < 0.5f) {
                obj.transform.rotation = Quaternion.Euler(0,
            }

            break;
    }
}
```

Dit is belangrijk doordat bijvoorbeeld een sword altijd iets naar beneden in de guard moet zitten. Maar ik kan hier ook leuke features zetten zoals een blade omdraaien omdat hij toch altijd 180 graden kan draaien en alsnog goed zit (Een gebogen zwaard heeft dan al 2 variaties).

Vervolgens wordt de part toegevoegd en kan de generator verder naar het volgende object, die zet hij bovenop het vorige object door middel van de top vertex berekenen van de mesh.

```
private Vector3 CalculateTopVertex(GameObject obj) {
    Vector3[] verts = obj.GetComponent<MeshFilter>().sharedMesh.vertices;
    Vector3 topVertex = new Vector3(0, float.NegativeInfinity, 0);
    for (int i = 0; i < verts.Length; i++) {
        Vector3 vert = transform.TransformPoint(verts[i]);
        if (vert.y > topVertex.y) {
            topVertex = vert;
        }
    }
    topVertex = new Vector3(obj.GetComponent<MeshRenderer>().bounds.center.x, topVertex.y, obj.GetComponent<MeshRenderer>().bounds.center.z);
    return topVertex;
}
```

Vervolgens aan het einde krijg je een volledig zwaard aan elkaar terug.

Result:

Sword Stats: by database (Pagina 17 t/m 19)

Ook moeten de zwaarden veel variaties hebben in hun stats om in een battle arena neergezet kunnen worden.

Het totaalplaatje daarvan ziet er zo uit:

```
itemValue = meshValue + materialValue;

if (CheckMaterialCombo()) {
    itemValue = itemValue * 1.5f;
}

if (Random.value < .05f) {
    itemName = NameDatabase.Instance.GenerateShittyName();
}
else {
    itemName = NameDatabase.Instance.GenerateName();
}

itemLength = CalculateLocalBounds();
itemWeight = CalculateWeight();
itemDurability = CalculateDurability();
itemValue = itemValue * (itemDurability / 100f);

SuperSpecificStuff();

itemDamage = itemValue * itemLength * itemDurability * itemWeight / 10000f;
```

De lengte wordt berekend door de bounds berekenen van het zwaard geheel.

```
private float CalculateLocalBounds() {
    Quaternion currentRotation = transform.rotation;
    transform.rotation = Quaternion.Euler(0f, 0f, 0f);

    Bounds bounds = new Bounds(transform.position, Vector3.zero);

    foreach (Renderer renderer in GetComponentsInChildren<Renderer>()) {
        bounds.Encapsulate(renderer.bounds);
    }

    Vector3 localCenter = bounds.center - transform.position;
    bounds.center = localCenter;


    transform.rotation = currentRotation;

    return itemLength = bounds.size.y / 10f;
}
```

Het gewicht wordt bepaald door een (minder clean code)

```
private float CalculateWeight() {
    itemWeight = 10;

    foreach (string s in materialTypes) {
        if (s == "Copper (UnityEngine.Material)") {
            itemWeight = itemWeight * 55.987f;
        }
        else if (s == "Bronze (UnityEngine.Material)") {
            itemWeight = itemWeight * 54.100f;
        }
        else if (s == "Steel (UnityEngine.Material)") {
            itemWeight = itemWeight * 49.421f;
        }
        else if (s == "Silver (UnityEngine.Material)") {
            itemWeight = itemWeight * 65.491f;
        }
        else if (s == "Gold (UnityEngine.Material)") {
            itemWeight = itemWeight * 120.683f;
        }
        else if (s == "Ruby (UnityEngine.Material)") {
            itemWeight = itemWeight * 30.683f;
        }
        else if (s == "Rune (UnityEngine.Material)") {
            itemWeight = itemWeight * 155.423f;
        }

        itemWeight = itemWeight / 50f;
    }
    return itemWeight;
}
```

Door gegevens van materiaal gewicht per object op het zwaard.

http://www.hibid.co.uk/metcalc/metcalc.html

Ook de durability van het zwaard wordt zo bepaald maar dan met random waardes of er durability van het wapen af is of niet (des te beter het materiaal des te minder kans er durability af is).

De waarde van het zwaard is:

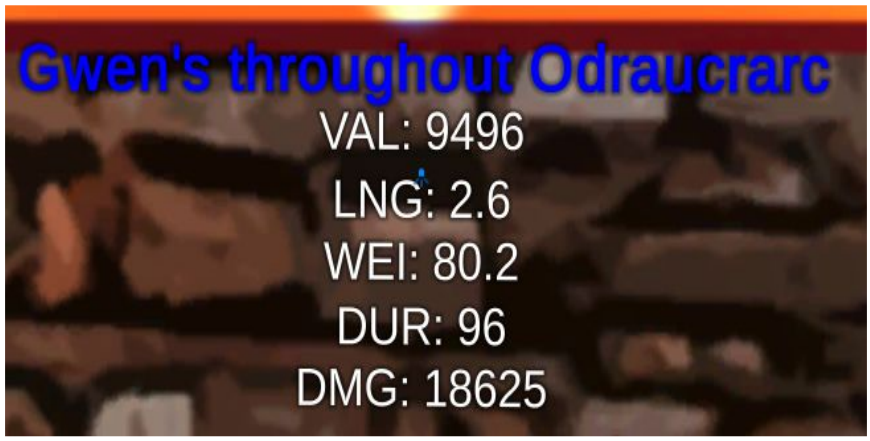
```
itemValue = meshValue + materialValue;
```

En de damage:

```
itemDamage = itemValue * itemLength * itemDurability * itemWeight / 10000f;
```

Result:

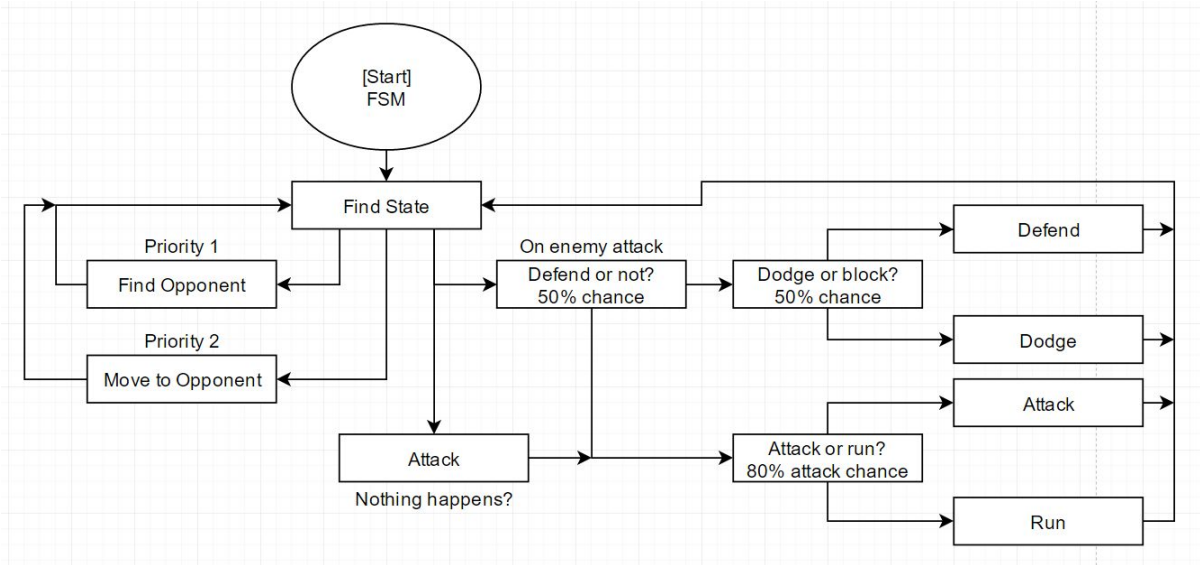

Amazing accompanied by Wruerul
VAL: 12095
LNG: 1.9
WEI: 13.1
DUR: 100
DMG: 3070


Gwen's throughout Odraucrarc
VAL: 9496
LNG: 2.6
WEI: 80.2
DUR: 96
DMG: 18625


Jesse's amidst Legbiter
VAL: 128584
LNG: 1.8
WEI: 5.5
DUR: 96
DMG: 12015
Highest Value

Sword Fighting: by Finite State Machine (Pagina 20 t/m 24)

Het vechten tussen de 2 zwaarden wordt gedaan met een finite state machine. Hieronder zie je een flowchart van de verschillende states en hoe de ai daar komt, de hele routine van de ai.



Dit wordt gedaan door een FSM die een lijst van states heeft die bepaalt worden in de inspector van Unity zelf.

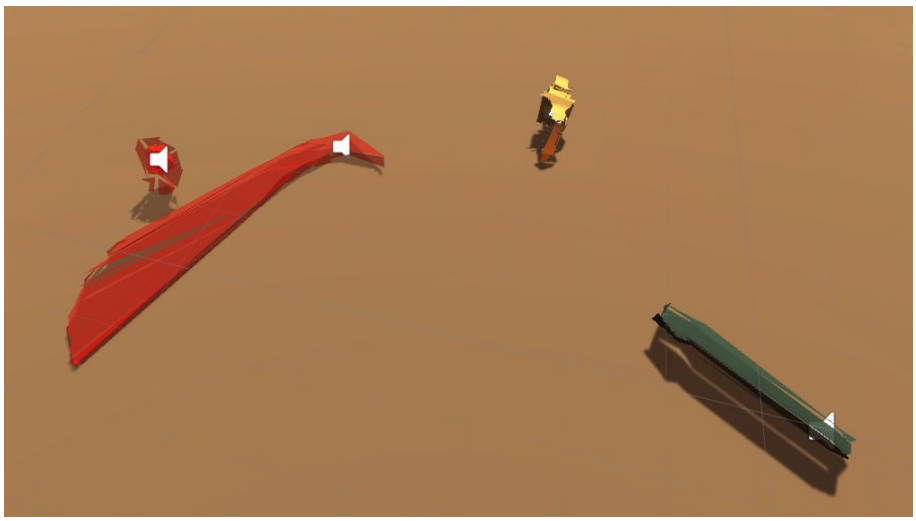
```
private void Start() {
    GetComponents();

    fsmStates = new Dictionary<string, State>();

    GetStates();
    SetStats();

    State idle = null;
    if (fsmStates.TryGetValue("Idle", out idle)) {
        currentState = idle;
        currentState.EnterState(this);
    }

}
```

Vervolgens wordt er in de statemachine zelf bijgehouden wat de:

- Stats van de speler zijn;
- Stats van de vijand zijn;
- Hoeveelheid de ai heeft aangevallen of verdedigd;
- Zijn huidig doelwit;
- Overige doelwitten (Support voor meer dan 2 swords);
- NavMesh agent;
- Animator;
- States waar de ai in kan;
- Audioclips;
- RigidBody;

Als de speler onder 0 hitpoints komt te zitten zal de FSM de speler vernietigen door middel van alle onderdelen van het zwaard los te koppelen en te damagen.

```
private void AdjustMeshVertices(GameObject g) {
    g.AddComponent<Rigidbody>();
    g.transform.parent = null;

    Mesh m = g.GetComponent<MeshFilter>().mesh;
    Vector3[] oldVertices = m.vertices;
    Vector3[] newVertices = new Vector3[oldVertices.Length];

    for (int k = 0; k < oldVertices.Length; k++) {
        newVertices[k] = new Vector3(oldVertices[k].x + Random.Range(-.1f, .1f), oldVertices[k].y + Random.Range(-.1f, .1f), oldVertices[k].z + Random.Range(-.1f, .1f));
    }

    m.vertices = newVertices;
    g.layer = 0;
    g.GetComponent<Rigidbody>().mass = weaponStats.itemWeight;
}
```



Als de speler zichzelf succesvol verdedigt of de vijand aanvalt zal er audio spelen en een textmesh in de lucht spawnen die wegfade na verloop van tijd:

```
private void OnCollisionEnter(Collision coll) {
    if (coll.gameObject.GetComponent<FSM>() != null) {
        FSM collFSM = coll.gameObject.GetComponent<FSM>();

        if (isAttacking) {
            if (!collFSM.isDefending) {
                collFSM.health -= damage;
                timesSuccesHits++;
                Instantiate(textMeshes[0], coll.contacts[0].point, Quaternion.identity);
            }
            else {
                collFSM.timesSuccesBlocks++;
            }
        }

        if (isDefending) {
            timesSuccesBlocks++;
            Instantiate(textMeshes[1], coll.contacts[0].point, Quaternion.identity);
        }

        audioSource.PlayOneShot(audioClips[Random.Range(0, audioClips.Length)]);

    }
}
```

Verder checkt heeft de statemachine functionaliteit zoals checken of hij in range is van zichzelf, hij kan er naar toe lopen en rotaten naar de vijand.

```
public bool IsInMeleeRangeOf(Transform target) {
    float distance = Vector3.Distance(transform.position, target.position);
    return distance < meleeRange;
}

public void MoveTowards(Transform target) {
    agent.SetDestination(target.position);
}

public void RotateTowards(Transform target) {
    Vector3 direction = (target.position - transform.position).normalized;
    Quaternion lookRotation = Quaternion.LookRotation(direction);
    transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime * rotationSpeed);
}
```

Dit wordt aangeroepen in verschillende states - bijvoorbeeld een attack wilt altijd rotaten naar de vijand om goed te slaan. Maar een block wilt dat ook om goed te blocken.

De states zelf zijn implementaties van de base state:

```
public abstract class State : MonoBehaviour, IState {
    public string stateName;

    public abstract void EnterState(FSM fsm);
    public abstract void UpdateState();
    public abstract void ExitState();
}
```

Die een interface IState implementeert.

```
public interface IState {
    void EnterState(FSM fsm);
    void UpdateState();
    void ExitState();
}
```

Een voorbeeld hiervan is de attack state die een attack animation afspeelt. De attack state heeft een StateBehaviour script om te zorgen dat hij juist op attack zal gaan in het midden v.d. Animatie. Ook zal deze tijdens de state naar de vijand toekijken om te hitten, vervolgens switcht hij naar de FindState state om een nieuwe state te zoeken.

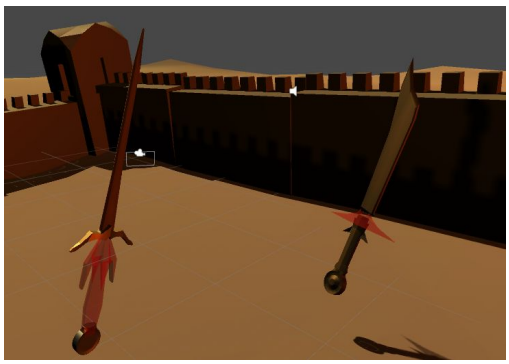
```
public class State_Attack : State {
    private FSM fsm;

    public override void EnterState(FSM _fsm) {
        Debug.Log("Entered: " + this.GetType());
        fsm = _fsm;

        if (Random.value < .5f) {
            fsm.anim.SetTrigger("Attack1");
        } else {
            fsm.anim.SetTrigger("Attack2");
        }

        Invoke("ExitState", 2f);
    }

    public override void UpdateState() {
        if (fsm.currentTarget != null) {
            fsm.RotateTowards(fsm.currentTarget.transform);
        }
    }

    public override void ExitState() {
        Debug.Log("Exitting: " + this.GetType());
        fsm.SwitchState("Idle");
    }
}
```

De volledige implementatie van de Find State flowchart ziet er zo uit:

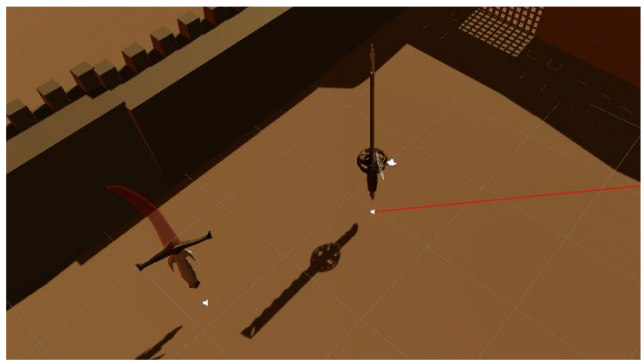
https://github.com/Thovex/Artificial-Intelligence/blob/master/Procedural%20Generation%20Arena%20Game/Assets/Scripts/AI/States/State_FindState.cs

Result:

Dodge

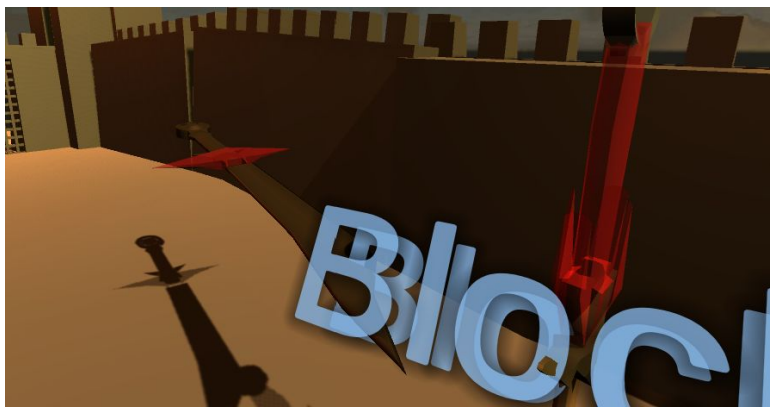


Flee



Attack

Block



Toevoegingen (met extra tijd)

Als ik meer tijd had voor dit project zou ik het vooral houden op het focussen van het optimaliseren van code want er zijn stukken code waar het natuurlijk een chaos is. Ik zou het meer als een geheel laten aanvoelen voordat ik door zou gaan met features toevoegen.

Na deze optimalisatie zou ik verschillende classes toevoegen. Denk aan swords, maces, halberds e.d. Met verschillende type AIs.

Ook zou ik swords willen kunnen laten teamen met elkaar zodat er 2v2, 4v4, 8v8 battles zouden komen in arena's met een heleboel diversiteit.

Een hash toevoegen aan een zwaard lijkt me ook erg interessant (dus dat elk zwaard een echte uniek nummer krijgt door alle stappen weer langs te gaan en aan de hashcode toevoegen).

Eventueel een eigen zwaard creëren en inzetten in de game om te gaan vechten in een soort tournament bracket ronde die gesimuleerd wordt.

De implementatie van het betten op swords zou zeker ook beter kunnen.

Over de kernmodule

De kernmodule zelf was echt super interessant en heb echt zoveel verschillende sorting algoritmes, artificial intelligence types, world generation algoritmes horen langskomen.

Dit maken was echt fantastisch en motiveerde me heel erg om telkens extra te gaan werken aan het project.

Dus all-in-all een 10/10 kernmodule blok, jammer van de drukte van andere vakken maar ik heb zeer zeker mijn tijd hierin kunnen stoppen.

Geschatte tijd gewerkt
Zelfstudie: 5-10u
Robocode: 5u
Research onderwerpen: 10u
Opdracht 1 boids: 15-20u
Opdracht 2 generation: 65-75u
Documentatie: ~5u
Presentatie: 2u

Links en extra's

GitHub opdracht 1 boids:
https://github.com/Thovex/Artificial-Intelligence/tree/master/Boids

GitHub opdracht 2 generation arena:
https://github.com/Thovex/Artificial-Intelligence/tree/master/Procedural%20Generation%20Arena%20Game

GitHub opdracht 2 voor *LeapMotion*:
https://github.com/Thovex/Artificial-Intelligence/tree/master/Procedural%20Generation%20Arena%20Game%20(LeapMotion)

Restaurant AI FSM zelfstudie:
https://github.com/Thovex/Artificial-Intelligence/tree/master/Restaurant%20AI

Flanking AI FSM zelfstudie:
https://github.com/Thovex/Artificial-Intelligence/tree/master/Flanking%20AI

Fractals zelfstudie:
https://github.com/Thovex/Artificial-Intelligence/tree/master/Fractals

Presentatie Kernmodule Game Development
https://docs.google.com/presentation/d/1ZMyWAMWjmMRwWaU1WFqLNheSq0qLpyn5gwHoBAauOAk/edit?usp=sharing

LeapMotion Opdracht 2 Gameplay:
https://www.youtube.com/watch?v=NJfRv5Y2VIo

GitHub Artificial Intelligence:
https://github.com/Thovex/Artificial-Intelligence

Andere sources staan onder desbetreffende onderwerpen in het document zelf.

Models, materials & scripts (met uitzondering van TextMeshPro, Classic Skybox, LeapMotion en LeapMotion Modules) zelf gemaakt.