PELITEKNIIKAN PORTFOLIO

JUHO HALONEN M1894

JYVÄSKYLÄN AMMATTIKORKEAKOULU

TICORPORATE DEMOLAB

KEVÄT 2020

1. Johdanto

Tämä dokumentti sisältää luomani pääsisällöt Virtual Shock- nimisessä projektissa ohjelmoijana työskennellessäni 2020 kevätlukukauden Ticorporate demolabin aikana. Käsittelen tässä dokumentissa projektia varten tekemiäni toiminnallisuuksia ja ratkaisuja jotka ovat mielestäni olleet merkittävimpiä projektin itsensä sekä oman oppimiseni kannalta. Käsittelen myös haasteita joita ilmeni työn aikana ja ratkaisujani niihin. Näiden lisäksi käsittelen myös hieman sivutyötäni project ownerina ja siihen liittyviä kokemuksiani.

2. Oppimistavoitteeni projektissa

Olen listannut alle asioita joita halusin oppia tai kehittää tässä projektissa työskennellessäni:

- C# Ohjelmointitaitojeni kehittäminen
- Unityn käyttötaitojeni kehittäminen
- Projektityöskentelytaitojeni kehittäminen
- VR-pelin suunnittelu ja kehitys
- Project Ownerin roolin ymmärtäminen ja oppiminen
- Scrum prosessin oppiminen

3. Toteutunut Oppiminen

Projektimme loppuvaiheessa näin jälkikäteen katsottuna olen mielestäni hyvin saavuttanut asettamani tavoitteet ja pidän opintojaksoa omalta kannaltani erittäin hyvin onnistuneena. Varsinkin ohjelmointitaitoni on huomattavasti parantunut ja olen entistä itsevarmempi koodia tuottaessani. Pääsen helposti alkuun uusien komponenttien ja kokonaisuuksien luonnissa, entistä enemmän jopa ilman minkäänlaista tutoriaalia.

Olen opintojakson oppimistavoitteideni lisäksi oppinut myös seuraavista asioista:

- C#:n event subscription systeemin käyttö
- Unityn SteamVR pluginin käyttö
- Unityn animatorin state machinen käyttö
- Unityn animaatio eventtien käyttö
- Parikoodaus
- Tekoälyn suunnitteleminen ja toteuttaminen
- Versionhallinnan käyttö
- Zenhubin käyttö

4. Merkittävimmät luomani kokonaisuudet

Olen projektin toisena ohjelmoijana toimiessani luonut ison osan pelimme toiminnallisuudesta. Näistä merkittävimmät projektin ja oppimiseni kannalta ovat seuraavat:

- Tekoäly pelin viholliselle
- Melee ja äänimekaniikat fyysisille esineille pelissä

Näiden lisäksi olen tehnyt peliin useita pienempiä komponentteja, joista osan olen tehnyt yksin ja osan parikoodauksena toisen ohjelmoijamme kanssa. Esittelen myös näistä mielestäni merkkittävimmät tässä dokumentissa.

Tekoäly

Yleiskuvaus

Suunnittelimme peliimme vihollisen jonka tarkoitus on vaeltaa ympäristössä ja hyökätä pelaajan havaitessaan. Olin yksin vastuussa vihollisen tekoälyn ohjelmoinnista. Ylivoimaisesti suurin osa ajastani projektissa työskennellessäni kului tekoälyn luomiseen. Tekoälyä luodessani ilmeni lukuisia ongelmia ja haasteita, joista suurimpaan osaan löysin toimivan ratkaisun.

Kuvaus tekoälyn toiminnasta pelissä

Tein tekoälylle erilaisia käyttäytymistiloja joiden välillä se vaihtelee tilanteesta riippuen.

Neutraalissa tilassa vihollinen osaa vaeltaa itsenäisesti ympäristössä. Suunnittelimme pelin ympäristön niin että vihollinen pääsee jokaiseen avoimeen paikkaan peliympäristössä.

Pelaajan havaitessaan vihollinen lähestyy pelaajaa ja siirtyy hiljalleen aggressiiviseen tilaan jossa se yrittää päästä lyömään pelaajaa. Hukatessaan pelaajan vihollinen siirtyy tilaan jossa se etsii pelaajaa. Tällöin vihollinen suorittaa normaalin vaellusrutiininsa, mutta liikkuu sekä havaitsee pelaajan nopeammin.

Jos vihollinen havaitsee pelaajan ennenkuin pelaaja on nähnyt vihollista, siirtyy vihollinen hiiviskelytilaan, jossa se yrittää hakeutua ääniä päästämättä pelaajan lähettyville.

Vihollinen reagoi myös pelaajan aiheuttamiin ääniin ympäristössä esim. Aseenlaukaukset, askeläänet, tavaroiden kolahtamiset pintoihin ym. Äänen kuullessaan vihollinen siirtyy suoraan etsintätilaan.

Vihollisella on kehossaan erilaisia osumakohtia, joihin osumalla pelaaja voi aiheuttaa viholliseen erilaisia määriä vahinkoa. Esimerkiksi pääosuma tekee kolminkertaista vahinkoa ja selkään osuminen kaksinkertaista.

Toteutus

Rakenne

Hyödynsin teköalyn toteutuksessa C#:n event systeemiä. Teköälyllä on yksi pääscripti (AI_Master), joka sisältää kaikki delegatet, event handlerit, eventit, sekä myös tärkeimmät tekoälyn toimintaan tarvittavat bool muuttujat.

Kaikki varsinainen toiminnallisuus on hajoitettu useaan eri scriptiin, joiden metodeja on subscribattu pääscriptin eventteihin. Täten kaikki scriptit ovat yhteydessä vain pääscriptiin joka keskustelee sitten muiden scriptien kanssa. Tämä selkeyttää koodia, sillä jokaiselle tekoälyn toiminnolle on oma scriptinsä ja nämä oheisscriptit tarvitsevat viitauksen vain pääscriptiin eikä toisiinsa.

Myös toimintojen yhdistäminen on eventtien ansiosta vaivattomampaa, sillä yhteen eventtiin voidaan subscribea monta eri metodia useasta scriptistä, jotka sitten kaikki ajetaan kyseistä eventtiä kutsuttaessa.

Ohessa on listattu kaikki tekoälyn scriptit mukana lyhyt selostus sen toiminnosta:

- **AI_Master** sisältää event handlerit, eventit ja tärkeimmät bool muuttujat
- **AI_Animation** kautta ajetaan vihollisen animaatiot
- **AI_Attack** sisältää vihollisen hyökkäyksen toiminnallisuuden
- **AI_Audio** kautta soitetaan vihollisen äänet
- **AI_DestinationReached** kertoo pääscriptille kun vihollinen on saavuttanut kohteensa
- **AI_Detection** sisältää toiminnallisuuden jolla vihollinen havaitsee pelaajan
- **AI_Footsteps** soittaa vihollisen askeläänet (olisi voinut yhdistää AI_Audioon)
- **AI_Health** hoitaa vihollisen terveyden vähennyksen sen ottaessa vahinkoa
- **AI_Listener** sisältää toiminnallisuudet joita vihollinen tarvitsee reagoidakseen ääniin
- **AI_NavPause** pysäyttää vihollisen liikkeen tarpeen tullen (esim. vahinkoa ottaessa)
- **AI_Pursue** sisältää toiminnallisuudet joita vihollinen tarvitsee jahdataakseen pelaajaa
- **AI_Roam** sisältää toiminnallisuudet joilla vihollinen saadaan vaeltamaan peliympäristössä
- **AI_TakeDamage** laskee vihollisen saaman vahingon määrän riippuen osumakohdasta

Esimerkki Event systeemistä

```
97  public void CallEventDie()
98  {
99      if (EventDie != null)
100     {
101         EventDie();
102     }
103     walking = false;
104     running = false;
105     isDead = true;
106
107     if (DatabaseController.instance != null)
108     {
109         DatabaseController.instance.kills++;
110     }
111 }
```

Esimerkkinä kun kutsutaan EventDie- eventtiä, tapahtuu seuraavat asiat:

```
97  void DisableThis()
98  {
99      //when the AI dies, this component will be disabled
100     this.enabled = false;
101 }
102 }
103
```

Kaikki oheis scriptit disabloidaan jotta ne eivät syö resursseja turhaan.

```
93  void AudioDeath()
94  {
95      //play one of the death sounds at random
96      int random = Random.Range(0, 2);
97      if (random == 1)
98      {
99          audio.PlayOneShot(audioDeath, volumeScale: 1f);
100     }
101     else
102     {
103         audio.PlayOneShot(audioDeath2, volumeScale: 1f);
104     }
105
106     //disable the component after a delay to ensure that the audio can play
107     Invoke(methodName: "DisableThis", time: 1f);
108 }
```

Ai_Audio script soittaa vihollisen kuolemis audion.

```
155     void AnimationDeath()
156     {
157         if (animator != null)
158         {
159             if (animator.enabled)
160             {
161                 if (animator.GetBool(name: "isRoaming"))
162                 {
163                     animator.SetBool(name: "isRoaming", value: false);
164                 }
165 
166                 if (animator.GetBool(name: "isPursuing"))
167                 {
168                     animator.SetBool(name: "isPursuing", value: false);
169                 }
170 
171                 //reset all triggers to ensure that no other animation will obstruct the death animation
172                 animator.ResetTrigger(name: "Attack");
173                 animator.ResetTrigger(name: "Aggro");
174                 animator.ResetTrigger(name: "normDamage");
175                 animator.ResetTrigger(name: "backDamage");
176                 animator.ResetTrigger(name: "critDamage");
177 
178                 //disable the animator after a delay to ensure that the death animation can play
179                 Invoke(methodName: "DisableThis", time: 1f);
180 
181                 animator.SetTrigger(name: "Die");
182             }
183         }
184     }
```

AI_Animation scripti käynnistää vihollisen kuolemisanimaation.

Vaellusmekaniikka

```
38      void Update()
39      {
40          if (Time.time > nextCheck)
41          {
42              nextCheck = Time.time + checkRate;
43 
44              if (aiMaster.target == null && !aiMaster.onRoute && !aiMaster.navPaused)
45              {
46                  SetWanderDestination();
47              }
48 
49              if (aiMaster.target == null && aiMaster.onRoute && !aiMaster.navPaused)
50              {
51                  //if it takes longer than 30 seconds for the ai to reach its wandertarget, set a new destination.
52                  //This is to prevent the AI from getting stuck trying to move where it cant move to
53                  if (Time.time > roamLimitNext)
54                  {
55                      roamLimitNext = Time.time + roamTimelimit;
56                      SetWanderDestination();
57                  }
58              }
59          }
60      }
61 
62      void SetWanderDestination()
63      {
64          if (RandomWanderTarget(centre: aiTransform.position, wanderRange, result: out wanderTarget))
65          {
66              if (!aiMaster.searchState)
67              {
68                  aiMaster.CallEventIsWalking();
69                  navMeshAgent.speed = 1;
70              }
71              else
72              {
73                  aiMaster.CallEventIsRunning();
74                  navMeshAgent.speed = 5;
75              }
76              aiMaster.onRoute = true;
77              aiMaster.roaming = true;
78              if (aiMaster.onRoute && (aiMaster.walking || aiMaster.running))
79              {
80                  navMeshAgent.SetDestination(wanderTarget);
81                  navMeshAgent.isStopped = false;
82              }
83          }
84      }
```

```
bool RandomWanderTarget(Vector3 centre, float range, out Vector3 result)
{
    //finds a random target location for the nav mesh agent to move to
    Vector3 randomPoint = centre + Random.insideUnitSphere * range;
    //if the position cant be reached, check if the navmesh can get near it
    if (NavMesh.SamplePosition(randomPoint, out navHit, maxDistance: 1.0f, areaMask: NavMesh.AllAreas))
    {
        result = navHit.position;
        return true;
    }
    else
    {
        result = centre;
        return false;
    }
}
```

AI_Roam scripti saa vihollisen vaeltamaan satunnaisesti ympäristössä. Se etsii randomilla vihollisen ympäriltä pisteen, jota kohti vihollinen lähtee liikkumaan. Samaan aikaan pääscriptin onRoute bool muuttuja laitetaan trueksi, jotta vihollisen täytyy päästä kohteeseensa ennenkuin sille annetaan uusi kohde.

```
void Update()
{
    if (Time.time > nextCheck)
    {
        nextCheck = Time.time + checkRate;
        DestinationReachedCheck();
    }
}

void DestinationReachedCheck()
{
    if (aiMaster.onRoute)
    {
        if (navMeshAgent.remainingDistance < navMeshAgent.stoppingDistance)
        {
            aiMaster.onRoute = false;
            aiMaster.CallEventTargetReached();
            navMeshAgent.isStopped = true;

            if (aiMaster.heardSound)
            {
                aiMaster.heardSound = false;
                aiMaster.CallEventExitSearchState();
            }
        }
    }
}
```

Kun vihollinen on saavuttanut kohteensa AI_DestinationReached scripti kertoo tästä pääscriptille asettamalla onRoute boolin takaisin falseksi, jotta Roam scripti voi asettaa viholliselle uuden kohteen.

Pelaajan havaitseminen

Pelaajan havaitseminen tapahtuu AI_Detection scriptin kautta. Teköälyllä on rajattu alue jolta se voi havaita pelaajan. Teköälyn näkökenttä on kartion muotoinen alue, jonka koko vaihtelee vihollisen tilasta riippuen. Pelaaja voi piiloutua viholliselta näköesteitä kuten seiniä ja erilaisia objektejä (esim tynnyreitä) hyödyntäen. Loin tekoälylle myös detection score systeemin, jonka ansiosta vihollisella menee hetki havaita pelaaja, vaikka pelaaja olisikin sen näkökentässä, riippuen siitä kuinka kaukana pelaaja on vihollisesta.

```
43      void Update()
44      {
45          DetectionFieldOfView();
46          LowerDetectionScore();
47
48          if (detectionScore < 0)
49          {
50              detectionScore = 0;
51          }
52
53          if (detectionScore > 10)
54          {
55              detectionScore = 10;
56          }
57
58          //change the fov angle depending on the state the ai is in
59          if (aiMaster.neutralState)
60          {
61              fovAngle = 150f;
62          }
63
64          if (aiMaster.searchState)
65          {
66              fovAngle = 225f;
67          }
68
69          if (aiMaster.huntState || aiMaster.stealthState)
70          {
71              fovAngle = 360f;
72          }
73      }
```

```
75      void DetectionFieldOfView()
76      {
77          if (Time.time > nextCheck)
78          {
79              nextCheck = Time.time + checkRate;
80
81              //check if the player is within the monster's range
82              Collider[] colliders = Physics.OverlapSphere(aiTransform.position, detectRadius, playerLayer);
83
84              if (colliders.Length > 0)
85              {
86                  foreach (Collider target in colliders)
87                  {
88                      if (target.CompareTag("PlayerController"))
89                      {
90                          Vector3 direction = target.transform.position - aiTransform.position;
91                          float angle = Vector3.Angle(direction, aiTransform.forward);
92
93                          //check if the player is within the monster's field of view
94                          if (angle < fovAngle * 0.5f)
95                          {
96                              if (LineOfSight(target.transform))
97                              {
98                                  break;
99                              }
100                         }
101                     }
102                 }
103             }
104             else
105             {
106                 aiMaster.CallEventTargetLost();
107             }
108         }
109     }
```

Pienen väliajan välein luodaan OverlapSphereä hyödyntäen pallon muotoinen detection alue, jonka sisältä pelaajaa yritetään etsiä.

```
bool LineOfSight(Transform target)
{
    //check if the player can be seen
    if (Physics.Linecast(start: head.position, end: target.position, out hit, sightLayer))
    {
        if (hit.transform == target)
        {
            if (aiMaster.neutralState)
            {
                aiMaster.CallEventGotoInvestigationState();
            }

            if (aiMaster.takenDamage)
            {
                //if the player has damaged the ai, it will be at maximum alert
                detectionScore = 10;
            }

            if (detectionScore >= 10)
            {
                aiMaster.CallEventSetNavTarget(target);

                if (!aiMaster.huntState && aiMaster.seenByPlayer)
                {
                    if (aiMaster.stealthState)
                    {
                        aiMaster.CallEventExitStealthState();
                    }
                    //switch from detecting player state to hunt state
                    aiMaster.CallEventGotoHuntState();
                }

                if (!aiMaster.stealthState && !aiMaster.seenByPlayer)
                {
                    //switch to stealth state if the player hasn't seen the monster
                    aiMaster.CallEventGotoStealthState();
                }

                if (!aiMaster.stealthState && !aiMaster.aggroAnimationPlayed && !aiMaster.takenDamage)
                {
                    //roar angrily at the player
                    aiMaster.CallEventAggro();
                }
                return true;
            }
            else
            {
                //if detection score is under 10 points, increase detection score based on distance from player
                distanceFromPlayer = Vector3.Distance(a: head.position, b: target.position);
                detectionScore += 300 / distanceFromPlayer;
                aiMaster.CallEventSetNavTarget(target);
                return true;
            }
        }
        else
        {
            aiMaster.CallEventExitInvestigationState();
            aiMaster.CallEventTargetLost();
            return false;
        }
    }
    else
    {
        aiMaster.CallEventTargetLost();
        return false;
    }
}
```

Kun pelaaja on löydetty tekoälyn detection alueelta. Katsotaan linecastilla voidaanko pelaaja nähdä, vai onko pelaaja näköesteen takana. Kun pelaaja on havaittu, kutsutaan pääscriptiltä eventtejä jotka siirtävät vihollisen jahtaamaan pelaajaa, sekä tilanteesta riippuen siirtymään joko stealth tai hunt stateen.

```
void LowerDetectionScore()
{
    //when the monster has returned to default roam state, its detection score will lower
    if (Time.time > detectionScoreTimerNextCheck)
    {
        detectionScoreTimerNextCheck = Time.time + detectionScoreTimerCheckRate;
        if (detectionScore > 0 && !aiMaster.huntState && !aiMaster.searchState
            && !aiMaster.investigationState && !aiMaster.stealthState)
        {
            detectionScore -= 1;
        }
    }
}
```

Detection scorea lasketaan kun vihollinen on palautunut oletustilaan.

Samanlainen detection mekaniikka on myös toistettu pelaajasta viholliseen. Tämän avulla vihollinen saa tietää onko pelaaja vielä havainnut vihollista. Jos ei, niin vihollinen huomatessaan pelaajan siirtyy stealth stateen.

Kuulemismekaniikka

Tein pelin viholliselle myös mekaniikan jolla se kuulee pelaajan toimintojen sekä ympäristön esineiden aiheuttamia ääniä ja reagoi niihin. Tämä onnistui tekemällä viholliselle trigger colliderin joka seuraa törmäyksiä vain ja ainoastaan noise layerillä oleviin collidereihin.

```
public class Noise : MonoBehaviour
{
    //The range of the noise
    public GameObject noiseColliderPrefab;

    public void SoundRangeColliderOn(float radius, float timer)
    {
        //score = noiseScore;
        GameObject soundRange = Instantiate(noiseColliderPrefab, transform.position, Quaternion.identity);
        soundRange.GetComponentInChildren<SphereCollider>().radius = radius;
        StartCoroutine(DestroySoundRangeCollider(timer, soundRange));
    }

    IEnumerator DestroySoundRangeCollider(float timer, GameObject soundRange)
    {
        yield return new WaitForSeconds(timer);
        Destroy(soundRange);
    }
}
```

Jokainen ääniefekti pelissä synnyttää pallon muotoisen noise colliderin, joka tuhotaan pienellä viiveellä. Tämän colliderin kokoa voidaan muuttaa riippuen kuinka paljon esineestä tai toiminnasta syntyy ääntä.

```
    void OnTriggerEnter(Collider col)
    {
        //if the AI collides with a noise trigger collider, it will set its destination to the source of the noise
        if (col.tag == "Noise")
        {
            float noiseVolume = CalculateNoiseVolume(col.gameObject.transform.parent.transform);
            if (aiMaster.target == null && navMeshAgent != null && !aiMaster.huntState && noiseVolume > 0)
            {
                if (!aiMaster.searchState)
                {
                    aiMaster.CallEventGotoSearchState();
                }

                navMeshAgent.SetDestination(col.transform.position);

                if (navMeshAgent.remainingDistance > navMeshAgent.stoppingDistance)
                {
                    navMeshAgent.speed = 5;
                    aiMaster.CallEventIsRunning();
                    navMeshAgent.isStopped = false;
                    aiMaster.onRoute = true;
                    aiMaster.heardSound = true;
                    aiMaster.roaming = false;
                }
            }
        }
    }

    float CalculateNoiseVolume(Transform noise)
    {
        //calculates the volume of the noise heard, depending on the amount and type of obstacles between noise source and the listener
        RaycastHit[] blockingObjects;
        float noiseVolumeToCalculate = 100;
        float distanceToTarget = Vector3.Distance(a: noise.position, b: transform.position);

        blockingObjects = Physics.RaycastAll
            (origin: transform.position, direction: noise.position - transform.position, distanceToTarget);
        if (blockingObjects.Length > 0)
        {
            for (int i = 0; i < blockingObjects.Length; i++)
            {
                if (blockingObjects[i].transform.gameObject.tag == "NoiseBlock")
                {
                    noiseVolumeToCalculate -= 35;
                }
                if (blockingObjects[i].transform.gameObject.tag == "NoiseBlock2x")
                {
                    noiseVolumeToCalculate -= 50;
                }
            }
        }
        return noiseVolumeToCalculate;
    }
```

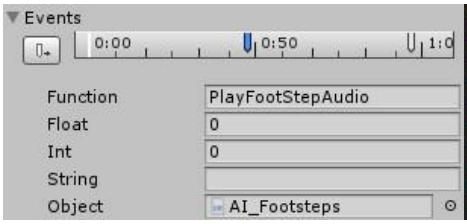
Kun vihollisen trigger collider osuu noise collideriin, tehdään vihollisesta RaycastAll äänen alkulähteen sijaintiin ja tallennetaan kaikki väliin osuvat objektit taulukkoon. Tämän jälkeen taulukosta etsitään objekteja joihin ollaan laitettu noiseblock tageja. Editorin puolella näitä tageja ollaan laitettu seiniin, lattioihin ja kattoihin. Tämän

mekaniikan avulla äänen voimakkuus “heikkenee” jos äänenlähteen ja vihollisen välillä on paljon ääntä blokkaavia objekteja.

Koodissa on myös otettu huomioon, että pelissämme seinien, lattioiden ja kattojen molemmat puolet lasketaan eri objekteiksi. Laitoin esimerkiksi pelimaailmassa lattia ja kattopalasiin kaksinkertaisen ääniblockin, jolla sain varmistettua että yläkerrassa oleva vihollinen ei kuule alakerrassa metelöivää pelaajaa. Avoimissa monikerroksisissa huoneissa tämä tagi jätettiin laittamatta, jotta ääni kulkisi realistisesti.

Animaatio eventtien hyödyntäminen

Sain animaatio eventtejä hyödyntämällä tietyt vihollisen toiminnot synkattua animaatioihin.




```
void PlayFootStepAudio()
{
    //this function is called from the run and walk animations
    //plays a random footstep audio from an array
    int n = Random.Range(1, footSteps.Length);
    audio.clip = footSteps[n];
    audio.PlayOneShot(audio.clip, volumeScale: 1);
    footSteps[n] = footSteps[0];
    footSteps[0] = audio.clip;
}
```

Käytin animaatio eventtejä soittamaan vihollisen askeläänet aina kun animaatiossa vihollisen jalka osuu maahan. Tällöin askeläänet tulevat juuri oikeaan aikaan sekä vihollisen kävellessä ja juostessa.

```
public void OnAIAttack()
{
    //This function is called from the attack animation
    if (attackTarget != null)
    {
        //before dealing damage to player, check if player is in front of the AI and in attack range
        if (Vector3.Distance(a: aiTransform.position, b: attackTarget.position) <= attackRange
            && attackTarget.GetComponentInParent<PlayerController>() != null)
        {
            Vector3 toOther = attackTarget.position - aiTransform.position;

            if (Vector3.Dot(lhs: toOther, rhs: aiTransform.forward) > 0.5f)
            {
                attackTarget.GetComponentInParent<PlayerController>().RestoreHP(-attackDamage);
                if (DatabaseController.instance != null)
                {
                    DatabaseController.instance.damageTaken += attackDamage;
                }
            }
        }
    }
}
```

Samaa olen hyödyntänyt myös vihollisen hyökkäykseen, jonka koodi kutsutaan keskeltä animaatiota. Tämän ansiosta pelaaja voi animaation nähdessään ennakoida ja mahdollisesti väistää vihollisen hyökkäyksen astumalla sivulle.

Haasteet

Tekoälyä rakentaessani ilmeni lukuisia erilaisia ongelmia. Kerron tässä muutamista merkittävimmistä haasteista sekä ratkaisuistani niihin.

Tekoälyn reitinhakeminen oli välillä hieman ongelmallinen. Tekoäly saattoi vaeltaessaan mennä sekaisin ja jäädä juoksemaan tai kävelemään paikalleen, koska ei löytänyt reittiä kohteeseensa.

Tätä ongelmaa varten tein muutoksia, joilla reitinhaku aikakatkaistaan jos vihollinen jää jumiin. Tässä tapauksessa tekoäly ottaa uuden kohteen. Pienensin myös aluetta, jolta tekoäly etsii vaelluskohteensa. Nämä ratkaisut tuntuivat auttavan ja tekoäly ei jäänyt enään jumiin peliä testatessa.

Toinen ongelma tekoälyn reitinhaussa oli pelaajan jahtaaminen navmesh alueen reunoilta. Jos esimerkiksi pelaaja oli seinää vasten eli juuri navmesh alueen ulkopuolella ja tekoäly katsoi että pelaajan kohdalla on ylemmässä kerroksessa navmesh-aluetta, juoksi tekoäly sitten ylempään kerrokseen.

```
NavMeshHit hit;
if (NavMesh.SamplePosition(aiMaster.target.position, out hit, maxDistance: 2f, areaMask: NavMesh.AllAreas))
{
    navMeshAgent.SetDestination(hit.position);
    navMeshAgent.isStopped = false;
}
else
{
    navMeshAgent.SetDestination(aiMaster.target.position);
    navMeshAgent.isStopped = false;
}
```

Sain tämän korjattua lisäämällä koodin ehdon, jossa hyödynnettään navmeshin sampleposition funktiota etsimään kohteen ympäriltä määrätyltä alueelta paikka johon voidaan päästä. Tämä korjaus auttoi ja vihollista oli korjauksen jälkeen paljon vaikeampaa jekuttaa hyödyntäen pelimaailman geometriaa.

Minulla oli myös hieman vaikeuksia saada vihollisen animaationvaihdokset kunnollisiksi ja kulutin niihin melko paljon aikaa. Esimerkiksi vihollisella oli paha tapa satunnaisesti liukua eteenpäin idle animaatiossa, sen sijaan että se kävelisi. Koodia parantamalla ja animaation exit aikoja muuttamalla state machinesta sekä suurella määrällä testausta, sain animaation vaihdokset lopulta kuitenkin melko hyvään kuntoon.

Oppiminen

Kulutin tekoälyn luomiseen todella paljon aikaa ja kirjoitin sitä kehittäessäni paljon koodia. Tämän työstäminen on ollut ohjelmoinnin oppimiseni kannalta erittäin hyödyllistä ja se on parantanut kykyäni luoda laajempia ja selkeämpiä kokonaisuuksia. Myös unityn osaaminen on mielestäni parantunut tätä työstäessäni, sillä iso osa vihollista luodessa tehtiin myös editorin puolella mm. Animator state machinen ja navmeshin kanssa.

Opin myös hyödyntämään ja arvostamaan entistä enemmän debug.login käyttöä, sillä se teki virheiden löytämisestä näin ison kokonaisuuden sisältä huomattavasti vaivattomanpaa.

Event subscription systeemin käyttö oli minulle melko uusi asia, joten olen oppinut myös sen hyödyntämisestä ja eduista paljon, varsinkin tekoälyn kaltaisiin laajoihin kokonaisuuksiin.

Myös tekoälyn suunnittelu ja kehitys itsessään ja sitä varten huomioitavat asiat ovat tuleet jonkin verran tutuiksi.

Loin alkupohjan tekoälylle pitkälti tutoriaalien avulla. Mutta aloin pian muokkaamaan tekoälystä omanlaistani projektimme tarkoitukseen paremmin sopivaa tekoälyä hyödyntäen sekä tutoriaaleista sekä muualta aikaisemmin saamaani osaamista.

Esineiden ääni ja melee mekaniikat

Yleiskuvaus

Tein toiminnallisuuden, jonka avulla kaikista pelimme esineistä kuuluu ääni niiden collidereiden törmätessä toisiin collidereihin. Sama scripti mahdollistaa myös damagen tekemisen pelin viholliselle, jos niitä heittää tai lyö esineillä. Tämä oli projektin kannalta erittäin merkittävä ominaisuus, sillä pelissämme on useita esineitä joita pelaaja voi käsitellä.

Toteutus

Äänimekaniikat

Äänen voimakkuus lasketaan koodissa kyseisen esineen liikenopeudesta. Mitä suurempi esineen liikenopeus on ennen törmäystä, sitä kovempi ääni siitä tulee ja sitä suuremman noise colliderin se spawnaa.

Törmäysnäänet laitoin soitettavaksi randomilla taulukosta, eli useamman eri äänen laitto samaan esineeseen on koodin ansiosta mahdollista. Samoin on mahdollista käyttää erillistä taulukkoa soittamaan ääniä kun esine osuu

viholliseen tarvittaessa. Eli vaikka esine kolahtaa metallipintaan lyödessä, niin vihollista lyödessä kuuluu siihen sopivampi ääni.

Meleemekaniikat

Käytin esineissä damagen laskemiseen liikenopeutta ennen törmaystä aivan kuten äänimekaniikoissakin. Damageen vaikuttaa liikenopeuden lisäksi erillinen damagekerroin joka voidaan säätää jokaista esinettä kohden eriarvoiseksi. Tämän avulla pystyin mahdollistamaan sen että esimerkiksi kirves tekee osuessaan paljon enemmän damagea kuin vaikka kahvikuppi.

Tämän lisäksi tein toisen muuttujan joka asettaa rajan siihen kuinka paljon liikenopeutta esineellä täytyy olla, jotta se voi edes tehdä yhtään damagea viholliseen. Säädin tämän rajan alhaiseksi erityisesti aseiksi tarkoitetuilla esineillä kuten kirveellä ja sorkkaraudalla esimerkiksi, ja korkeaksi heikommilla esineillä kuten kahvikupit, perunat ym.

```
35  void FixedUpdate()
36  {
37      if (rb.velocity.x != 0 || rb.velocity.y != 0 || rb.velocity.z != 0)
38      {
39          velocityTotal = Mathf.Abs(f:rb.velocity.x + rb.velocity.y + rb.velocity.z);
40      }
41  }
42  void OnCollisionEnter(Collision col)
43  {
44      if (Time.time > nextcheck)
45      {
46          nextcheck = Time.time + checkrate;
47
48          //calculate the volume of the collision noise by velocity
49          float collideVolume = 0.1f * velocityTotal;
50
51          //limit the volume to prevent it from being absurdly loud
52          if (collideVolume > 1f)
53          {
54              collideVolume = 1f;
55          }
56
57          //only alert enemies with the noise if the velocity is high enough
58          if (velocityTotal > 2)
59          {
60              noise.SoundRangeColliderOn(radius:velocityTotal * 5, timer:0.1f);
61          }
62
63          //play a different sound if the item hits an enemy
64          //also do melee damage to the enemy based on velocity and damage multiplier
65          if (col.gameObject.tag == "Enemy")
66          {
67              if (velocityTotal >= damageThresholdVelocity)
68              {
69                  if (willSpawnBloodParticles)
70                  {
71                      GameObject go = (GameObject)Instantiate(bloodParticle, col.contacts[0].point, Quaternion.identity);
72                  }
73
74                  int damageToDeal = Convert.ToInt32(value:velocityTotal * damageMultiplier);
75                  if (enemyCollideNoise.Length > 0)
76                  {
77                      PlayCollideNoise(collideVolume, enemyCollideNoise);
78                  }
79                  else
80                  {
81                      PlayCollideNoise(collideVolume, collideNoise);
82                  }
```

```
83
84                  if (col.gameObject.GetComponent<AI_TakeDamage>() != null)
85                  {
86                      col.gameObject.GetComponent<AI_TakeDamage>().DoDamage(damageToDeal);
87                  }
88                  if (interactable != null && DatabaseController.instance != null && interactable.attachedToHand != null)
89                  {
90                      DatabaseController.instance.meleeHitsOnEnemy++;
91                      DatabaseController.instance.meleeHits++;
92                  }
93              }
94          }
95
96          else
97          {
98              //prevent noise before the player has started the game
99              //this is to prevent the horrible amount of noise resulting from everything in the level dropping into their places at start
100             if (GameController.instance.gameStarted)
101             {
102                 PlayCollideNoise(collideVolume, collideNoise);
103             }
104
105             if (interactable != null && DatabaseController.instance != null && interactable.attachedToHand != null)
106             {
107                 DatabaseController.instance.meleeHits++;
108             }
109         }
110     }
111 }
112
113 void PlayCollideNoise(float volume, AudioClip[] collide)
114 {
115     //play a sound at random from the collision sound array
116     audio.pitch = UnityEngine.Random.Range(0.8f, 1.2f);
117     if (collide.Length > 1)
118     {
119         int n = UnityEngine.Random.Range(1, collide.Length);
120         audio.clip = collide[n];
121         audio.PlayOneShot(audio.clip, volume);
122         collide[n] = collide[0];
123         collide[0] = audio.clip;
124     }
125     else
126     {
127         audio.clip = collide[0];
128         audio.PlayOneShot(audio.clip, volume);
129     }
130 }
131 }
```

Haasteet

Yksi ongelma joka selvisi melee mekaniikoiden testauksen alkuvaiheilla oli se että pienenkin damage kertoimen esineet tappoivat vihollisen turhan nopeasti. Tämän vuoksi laitoin aikarajan siihen kuinka usein esineen osumakoodi ajetaan.

Toinen iso ongelma oli vihollisten törmääminen pelimaailmassa lojuviin esineisiin, niiden vaeltaessa ympäri pelimaailmaa (varsinkin tynnyreihin). Tämä sai viholliset sekä ottamaan damagea ja tuottamaan ääntä, johon ne itse tietysti listener komponentin takia reagoivat, vaikka äänenlähde ei ollutkaan pelaaja tai pelaajan toiminta.

Ongelma on vielä jossain määrin olemassa pelissä, mutta olen lieventänyt sitä huomattavasti asettamalla koodiin rajat jotka esineen täytyy täyttää tehdäkseen damagea ja ääntä (johon viholliset reagoivat).

Yksi pienempi ongelma oli pelin aloittaessa kuuluva meteli kun kaikki esineet tippuivat paikalleen pelimaailmassa. Koska varsinainen peli kuitenkin alkaa pelaajan painaessa uuden pelin aloitusta päävalikossa, niin ratkaisu oli helppo. Eli äänet pidetään hiljaisena ennenkuin varsinainen pelaaminen alkaa.

Oppiminen

Tätä komponenttia luodessani pääsin soveltamaan paljon projektin aikana ja aikaisemmin saamaa osaamistani. Tämä on ensimmäisiä isompia scriptejä joita sain kirjoitettua ilman yhdenkään tutoriaalin apua. Opin myös tätä kehitellessäni ja testatessani tavoista kuinka olisi järkevää toteuttaa lähitaistelumekaniikat VR-pelissä.

5. Muita tekemiäni kokonaisuuksia

Tähän kappaleeseen olen listannut pienempiä asioita joita olen kehittänyt tai ollut mukana kehittämässä projektia varten. Nämä ovat minun yksin luomiani kokonaisuuksia:

- Noise mekaniikat pelaajan liikkumiseen
- Äänimekaniikat pelaajan liikkumiseen
- Ampumismekaniikat pistooliin
- Ääni ja noise mekaniikat pistooliin
- Taskulampun päälle ja poislaitto mekaniikat
- Oven avautumismekaniikat
- Scriptattu kohtaus pelin alussa, jossa vihollinen näytetään pelaajalle ensimmäistä kertaa, sen vaeltaessa viereisen huoneen läpi
- Triggerit joilla vaihdetaan musiikkikappaleita pelaajan edetessä pelissä
- Secret Area triggerit, joilla kerrotaan tietokannalle kun pelaaja on löytänyt salaisen alueen
- Vihollisten “spawnaaminen” pelialueelle pelaajan edetessä pelissä
- Pelaajan vahingonotto mekanismi
- Pelaajan kuolemis mekanismi

Näiden lisäksi teimme toisen ohjelmoijan kanssa parikoodauksella seuraavat kokonaisuudet:

- Pelaajan liikkuminen: kävely, juoksu, hyppiminen ja kyykkiminen
- Oven avaaminen oven vieressä sijaitsevalta numeropadilta
- Numeropadin mekaniikat

Aivan kuten aikaisemmin mainitsemissani isommissa kokonaisuuksissa. Myös nämä pienemmät asiat ovat olleet arvokkaita oppimiseni kannalta. Jokaisen näiden luominen on laajentanut tietämystäni unityn komponenteista ja niiden hyödyntämisestä. Myös taitoni ja itsevarmuuteni kirjoittaa koodia on parantunut kaikkia näitä kokonaisuuksia luodessa.

Olen ottanut alas hieman poimintoja näiden komponenttien koodeista niiltä osin mitkä koin itse oppimiarvoltaan tai projektin kannalta merkittävimmiksi:

Ovi Scripti

```
void Awake()
{
    audio = GetComponent<AudioSource>();
    DOTween.Init();

    if (doorIsOpenOnStart)
    {
        doorOpenCoord = transform.position.y;
        doorClosedCoordY = transform.position.y - doorOpenHeight;
        doorIsOpen = true;
    }

    if (!doorIsOpenOnStart)
    {
        doorOpenCoord = transform.position.y + doorOpenHeight;
        doorClosedCoordY = transform.position.y;
        doorIsOpen = false;
    }
}

public void DoorOpen()
{
    doorIsOpen = true;
    DoorCloseTween.Kill();
    DoorOpenTween.Append(transform.DOMoveY(endValue:doorOpenCoord, duration:doorMoveTime, snapping:false));
    audio.PlayOneShot(doorOpenAudio);

    //have the door automatically close after a while
    //bulkheads are special and will always remain open
    if (!doorIsBulkhead)
    {
        Invoke(methodName:"DoorClose", time:10);
    }
}

public void DoorClose()
{
    CancelInvoke();
    doorIsOpen = false;
    DoorOpenTween.Kill();
    DoorCloseTween.Append(transform.DOMoveY(endValue:doorClosedCoordY, duration:doorMoveTime, snapping:false));
    audio.PlayOneShot(doorCloseAudio);
}
```

```
void OnTriggerStay(Collider col)
{
    //prevent the door from closing if something is blocking it
    if (!doorIsBulkhead)
    {
        if ((col.tag == "Object" || col.tag == "PlayerController" || col.tag == "Player" ||
             col.tag == "Enemy" || col.tag == "PlayerHands") && !doorIsOpen && !(transform.position.y <= doorClosedCoordY + 0.1))
        {
            doorIsBlocked = true;
            DoorOpen();
            Debug.Log(message:"ovi tukossa");
        }
    }
}

void OnTriggerExit(Collider col)
{
    if (!doorIsBulkhead)
    {
        if ((col.tag == "Object" || col.tag == "PlayerController" || col.tag == "Player" ||
             col.tag == "Enemy" || col.tag == "PlayerHands") && doorIsBlocked)
        {
            doorIsBlocked = false;
            Debug.Log(message:"nyt on kunnollista");
        }
    }
}
}
```

Yllä olevissa kuvissa on ovi scripti, jolla kaikki pelin ovet avataan. Oven koko ja avautumisnopeus on kaikki säädettävissä ja ovi toimii missä tahansa korkeudessa pelimaailmassa. Ovi myös havaitsee sulkeutuessaan jos pelaaja tai jokin objekti on jäämässä väliin ja Scripti on helposti kutsuttavissa muista scripteistä, kuten numeropadin ja triggereventtien scripteistä. Hyödynsin tässä scriptissä myös DOTween nimistä Unity lisäosaa, joten opin sen käytöstä hieman.

AI Intro Scripti

```
void Update()
{
    if (aiIntroStarted)
    {
        if (Time.time > nextCheck)
        {
            nextCheck = Time.time + checkRate;
            DestinationReachedCheck();
        }
    }
}

public void StartAIIntro()
{
    //the ai dummy will start walking and will also let out an angry roar
    SetNewDestination();
    if (aiAudio != null)
    {
        aiAudio.TutorialAudioAggro();
    }
    aiIntroStarted = true;
}

void DestinationReachedCheck()
{
    if (aiMaster.onRoute)
    {
        if (navMeshAgent.remainingDistance < 1)
        {
            //once the ai dummy reaches the third waypoint, play another sound
            if (i == 2)
            {
                aiAudio.TutorialAudioHuntState();
            }
            aiMaster.onRoute = false;
            aiMaster.CallEventTargetReached();
            navMeshAgent.isStopped = true;
            i++;

            //when the ai dummy reaches its goal, destroy it
            if (i > 3)
            {
                DestroyIntroPropAI();
            }
            else
            {
                SetNewDestination();
            }
        }
    }
}
```

```
void SetNewDestination()
{
    if (i < waypoints.Length)
    {
        //sets the ai dummy's destination to the next waypoint in the array
        navMeshAgent.isStopped = false;
        navMeshAgent.SetDestination(waypoints[i].position);
        navMeshAgent.speed = 1;
        aiMaster.onRoute = true;
        aiMaster.CallEventIsWalking();
    }
}

public void DestroyIntroPropAI()
{
    Destroy(aiTransform.gameObject);
}
}
```

Yllä olevissa kuvissa on scripti jonka avulla pelin vihollisen ensimmäinen ilmestyminen pelaajalle on toteutettu. Käytin tätä kohtausta varten erillistä dummy versiota AI prefabista, jolta oli iso osa komponenteista poistettu. Dummy ei reagoi pelaajaan mitenkään eikä vaella vapaasti ympäristössä. Se myös tuhotaan ennenkuin pelaaja pääsee sen luokse tai kun se on päässyt maaliinsa.

Tutorial End Scripti

```
void Start()
{
    //disable all AI objects associated with this component
    for (int i = 0; i < aiObjects.Length; i++)
    {
        aiObjects[i].SetActive(value:false);
    }
}

void OnTriggerEnter(Collider col)
{
    //once the player touches the trigger, enable monsters around the level
    if (col.tag == "PlayerController")
    {
        for (int i = 0; i < aiObjects.Length; i++)
        {
            aiObjects[i].SetActive(value:true);
        }

        if (musicPlayer.GetComponent<MusicPlayer>() != null)
        {
            musicPlayer.GetComponent<MusicPlayer>().StartLobbyAmbient();
        }

        //start rotating the skybox
        if (spaceRotation != null)
        {
            spaceRotation.rotationStarted = true;
        }
        DisableThis();
    }
}

void DisableThis()
{
    //disable the trigger to prevent the player from triggering it multiple times
    Destroy(otherTrigger);
    Destroy(obj:this);
}
```

Ylläoleva scripti ajetaan kun pelaaja osuu trigger collideriin poistuessaan pelin aloitusalueelta. Se enabloi viholliset pelimaailmassa, käynnistää taustamusiikin sekä alkaa pyörittää avaruus skyboxia. Se myös tuhotaan tämän jälkeen. Koska pelaajalla on kaksi mahdollista reittiä poistua aloitusalueelta, myös triggereitä täytyy olla kaksi. Toinen triggeri tuhotaan myös samanaikaisesti.

Tekemiäni osia Pistooli Scriptistä

```
if (pistol.attachedToHand != null)
{
    SteamVR_Input_Sources source = pistol.attachedToHand.handType;
    if (shoot[source].stateDown && Time.time > nextFire)
    {
        nextFire = Time.time + fireRate;
        Shoot();
    }
```

Koodi tunnistaa kummassa kädessä pistooli on pelaajalla, joten vasemmassa kädessä olevalla pistollilla ampuminen tapahtuu vasemman ohjaimen liipaisimesta sekä toisinpäin. Loput pistoolin koodista alla onkin suurelta osin jo aikaisemmin oppimani soveltamista, mutta pelin kannalta silti erittäin tärkeää toiminnallisuutta.

Laitoin pistoolille raycastin joka ammutaan pistoolin piipusta eteenpäin ja osuessaan viholliseen se kutsuu AI_TakeDamage scriptiä ja tekee vahinkoa. Tämän lisäksi pistoolilla ampuminen puskee taaksepäin erilaisia objekteja joille on asetettu rigidbody. Pistoolilla ampuminen synnyttää myös noise colliderin joka hälyttää sen sisään osuvan vihollisen siihen sijaintiin missä pistoolilla ammuttiin.

Alla on loput tekemistäni osista pistoolin koodista:

```
RaycastHit hit;

//offset range for the shot
Vector3 offsetVect = new Vector3(x:Random.Range(-offset, offset), y:Random.Range(-offset, offset), z:Random.Range(-offset, offset));

if (Physics.Raycast(origin:pistolEnd.transform.position, direction:pistolEnd.transform.forward + offsetVect, out hit, range))
{
```

```
            //Do damage to enemies on hit
            if (hit.transform.gameObject.tag == "Enemy")
            {
                GameObject go = (GameObject)Instantiate(enemyHitPrefab, hit.point, Quaternion.LookRotation(hit.normal));
                if (hit.transform.gameObject.GetComponent<AI_TakeDamage>() != null)
                {
                    hit.transform.gameObject.GetComponent<AI_TakeDamage>().DoDamage(damage);

                    if (hit.transform.root.gameObject.GetComponent<AI_Master>() != null)
                    {
                        if (!hit.transform.root.gameObject.GetComponent<AI_Master>().isDead)
                        {
                            //Record the bullet hit on enemy to the database controller, bullet hits on dead enemies are not recorded
                            if (DatabaseController.instance != null)
                            {
                                DatabaseController.instance.bulletsHitOnEnemy++;
                            }
                        }
                    }
                }
            }

            //instantiate a particle effect on hit
            if (hitPrefab != null)
            {
                if (hit.transform.gameObject.tag != "Enemy")
                {
                    GameObject go = (GameObject)Instantiate(hitPrefab, hit.point, Quaternion.LookRotation(hit.normal));
                }
            }

            if (hit.rigidbody != null)
            {
                //push rigidbodies hit by the gun's raycast

                hit.rigidbody.AddForce(-hit.normal * hitForce, ForceMode.Impulse);
            }
```

```
            //play muzzle flash particle effect and enable point light
            muzzleFlashLong.Play();
            muzzleFlashShort.Play();
            if (muzzleLight != null)
            {
                MuzzleLightSourceOn();
            }

            //play audio for the shot
            audio.pitch = Random.Range(0.9f, 1.1f);
            audio.PlayOneShot(shootAudio);

            //Alert nearby enemies with noise
            noise.SoundRangeColliderOn(radius: 20, timer: 0.1f);

            //Record the shot to the database controller
            if (DatabaseController.instance != null)
            {
                DatabaseController.instance.shotBullets++;
            }

        }

        void MuzzleLightSourceOn()
        {
            muzzleLight.SetActive(value: true);

            //turn off point light after a time
            Invoke(methodName: "MuzzleLightSourceOff", muzzleLightTimer);
        }
        void MuzzleLightSourceOff()
        {
            muzzleLight.SetActive(value: false);
        }
```

6. VR-kehityksestä oppimaani

Projektin aikana olen oppinut asioista, joita tulee ottaa huomioon VR-pelikehityksessä. Kuten liikkumisen toteutus ja pelaajan vuorovaikutus pelimaailman kanssa.

Opin että pelaajaobjektia ei kannattaisi liikuttaa ilman pelaajan inputtia, sillä se saattaa tuntua epämukavalta pelaajalle. Opin myös että VR-kehityksessä tulee ottaa huomioon pelaajan eri objektit kuten VR kamera, kädet ja pelaajan oma bodycollider ja miettiä kuinka niiden tulisi vaikuttaa peliympäristössä.

Merkittävä osa VR toiminnallisuudesta projektiimme tuli valmiina Unityn SteamVR pluginin kautta. Meidän ratkaistavaksemme jäi pääasiassa näiden valmiiden komponenttien hyödyntäminen meidän omiin tarkoituksiimme.

Haasteet

Ehkä suurin haaste oli liikkumisen saaminen oikeanlaiseksi. Koska käytimme pelaajalle valmista SteamVR pluginin prefabia, jonka pohjalta aloimme tekemään omanlaista pelaajaobjektiamme, meille oli hieman epäselvää miten kaikki osat siinä toimivat.

Kun teimme pelaajalle ohjaimen tatilla kääntymisen, käänsi se pelaajan sijaan koko VR-pelialuetta. Saimme lopulta kääntymisen toimimaan niinkuin halusimme kääntämällä pelaajan sijaan VR-kameraa joka oli pelaajaprefabin lapsiobjekti.

Toinen ongelma olivat trigger colliderit. Pelaajaprefabissa tuli mukana character controller componentti joka toimi samaan aikaan bodycolliderina pelaajalle. Trigger colliderit eivät kuitenkaan tunnistaneet character controlleria joten jouduimme luomaan pelaajalle uuden bodycolliderin. Sen jälkeen laitoimme tämän uuden colliderin päivittämään sijaintiaan VR kameran sijaintiin sekä muuttamaan kokoaan kameran korkeudesta riippuen pelialueen pohjaan nähden.

7. Project Owner

Projektin aikana oli sivutyönäni ryhmämme project ownerina toimiminen. Tosin suurin osa ajastani kului pelitekniikan kanssa.

Vastasin projektin aikana product backlogista ja sprint backlogista sekä käyttäjäkertomusten ja niiden kriteerien kirjoittamisesta. Olin vastuussa siitä mitkä userstoryt valitaan sprintin suunnittelussa mukaan sprinttiin. Näiden lisäksi vastasin pelin visiosta ja kävin koko projektin ajan yhteistyötä tiimin artistien ja äänisuunnitelijoiden kanssa suunnitellessamme pelin sisältöä.

Opin projektin aikana project ownerin tehtävistä ja vastuista sekä kehitin osaamistani sekä ymmärrystäni scrum-prosessista. Olen oppinut käyttäjäkertomuksista ja niiden tekemisestä. Samalla myös zenhubin hyödyntäminen projektin organisoinnissa on tullut tutuksi. Tunnen mielestäni paremmin omat heikkouteni ja vahvuuteni projektinomistajana toimimisessa.

8. Olisiko ollut parannettavaa?

Sivutyöni jäi projektin aikana hieman taka-alalle, keskittyessäni enemmän pelitekniikkaan. Olisin mielestäni voinut hoitaa project ownerin hommia enemmän ja paremmin, mikä olisi helpottanut varsinkin projektimme sprinttien suunnittelussa.

Myös scrum-prosessista ja sen hyödyntämisestä projektityöskentelyssä oli varsinkin alkuvaiheissa hieman sekaannusta ryhmässämme, mikä saattoi hieman haitata työnteon ja projektin suunnittelua. Silti olen erittäin tyytyväinen tiimimme lopputulokseen ja siihen kuinka pysyimme näistä vaikeuksista pitämään kiinni alkuperäisestä visiostamme.

Ohjelmoinnissa olisimme voineet hyödyntää event systeemiä myös tekoälyn ulkopuolella. Esimerkiksi ison osan pelaajan toiminnoista olimme laittaneet samaan scriptiin, mikä teki koodista hieman sekavaa ja muutoksista hankalampaa. Siinä vaiheessa kun olin oppinut event systeemistä, olisi pelaajan scriptien muuttaminen kyseistä systeemiä käyttämään tosin ollut jo ehkä liian myöhäistä ja turhaa.

9. Pohdintaa

Pidän opintojaksoa hyvin onnistuneena. Olen saanut projektin aikana paljon osaamista ohjelmoinnista sekä pelinteosta unityllä. Lisäksi sain paljon kokemusta projektityöskentelystä ryhmässä, sillä tämä oli ensimmäinen iso ryhmässä tuotettu ohjelmistoprojekti opintojeni aikana. Sain kokemusta siitä minkälaista on tuottaa peliä suuremmalla tiimillä, kun kaikki tätä aikaisemmat peliprojektini ovat olleet sooloprojekteja. Pidin myös pelituotantoa tiimissä erittäin mielekkäänä työnä. Myös scrum-prosessista sekä projektinomistajan roolista tuli kevään aikana hyvin tietämystä.