**Krav til opgave:**

Afgangsprojektet er den afsluttende prøve på uddannelsen og skal dokumentere, at uddannelsens mål for læringsudbytte er opnået. Derfor skal afgangsprojektet afspejle uddannelsens faglige område. Emnet og problemstillingen i afgangsprojektet er selvvalgt og skal godkendes af institutionen. Emnet formuleres, så uddannelsens indhold som helhed samt eventuelle valgfag uden for uddannelsens faglige område inddrages. Fokusset i projektet er analyse, refleksion, vurdering og håndtering af problemer inden for det valgte faglige område gennem anvendelse af relevante teorier og metoder.

Viden: Den uddannede:

- har viden om erhvervets praksis og central anvendt teori og metode i relation til det valgte informationsteknologiske område

- har forståelse for praksis og central anvendt teori og metode samt forståelse for erhvervets anvendelseaf teori og metode i relation til det valgte informationsteknologiske område.

Færdigheder: Den uddannede kan:

- anvende fagområdets centrale metoder og redskaber samt anvende de færdigheder, der knytter sig til erhverv med relation til det valgte informationsteknologiske område

- vurdere praksisnære problemstillinger samt opstille og vælge løsningsmuligheder

- formidle praksisnære problemstillinger og løsningsmuligheder til samarbejdspartnere og brugere.

Kompetencer: Den studerende kan:

- håndtere udviklingsorienterede situationer

- deltage i fagligt og tværfagligt samarbejde med

Mundtlig prøve på baggrund af langt individuelt projekt

Et langt projekt er kendetegnet ved:

- at den studerende selv udarbejder en problemformulering eller hypotese

- at den studerende selv vælger det teoretiske og empiriske grundlag for projektet

- at det skriftlige materiale præsenterer og anvender teori og empiri, der demonstrerer den studerendes evne til teoretisk og empirisk belysning af problemformuleringen.

Projektets indhold: Et langt projekt skal indeholde følgende elementer:

- præsentation af problemstilling, der tydeliggør relevansen af problemformuleringen

- problemformulering, evt. i hypoteseform samt evt. relevant afgrænsning

- metodeovervejelser og metodevalg, dvs. valg af teoretisk og empirisk tilgangsvinkel

- analyse indeholdende evt. mulige handlerum

- konklusion og perspektivering.

Projekts omfang: Et langt projekt har et omfang på maksimum 15 normalsider. Det lange projekt kan suppleres med bilag i form af videooptagelser, interviewoptagelser, links med videre.

Bedømmelse: Bedømmelsesgrundlaget er en samlet helhedsvurdering af den skriftlige og den mundtlige præstation. Der gives en samlet karakter efter 7-trins skalaen.

Mundtlig prøve: Den mundtlige prøve har en varighed på i alt 30 minutter, der har følgende omtrentlige fordeling:

- 1/4 til et mundtligt oplæg

- 2/4 til eksaminationssamtale

- 1/4 til votering og tilbagemelding til den studerende.

EN GOD PROBLEMSTILLING?

Formuler en problemstilling, der består af 1-3 spørgsmål.

Vurder din problemstilling. Den er god, hvis den lever op til følgende kriterier:

• Lægger op til, at du arbejder med et reelt historisk problem.

• Spørgsmålene er klare og tydelige.

• Du kan få adgang til forskellige kilder, der kan bruges til at belyse problemstil­lingen.

• Det er realistisk, at du kan besvare pro­blemstillingen inden for de givne ram­mer (bl.a. den tid, der er afsat til arbej­det).

• Problemstillingen lægger op til undersø­gelser, der er relevante i faget historie.

• Problemstillingen indeholder spørgsmål på forskellige niveauer (beskrivende, forklarende, vurderende osv.)

Det er en dårlig problemstilling, hvis den:

• Lægger op til ren gengivelse af, hvad der står i en bog eller på en hjemmeside.

• Mangler fokus (ingen sammenhæng mel­lem spørgsmålene).

• Er for bred.

• Ikke lægger op til diskussion.

• Faktisk ikke kan besvares.

• Er upræcis.

Eksempel

Da timen er slut, når de frem til følgende:

• Hvilke grunde var der til, at de fleste euro­pæiske stater forbød tortur i begyndelsen af 1800-tallet?

• Hvorfor anvender nogle stater stadig tortur, når det har været forbudt i mange år?

Hvad er Unity?

Unity er en gameengine, som kan kombinere grafiske elementer, teknisk opsætning og scripts, der kan spille sammen.

En meget stor del af et spil er opsætning af de grafiske elementer (verdenen), opsætningen af den grafiske brugergrænseflade, opsætning af animationer mv. og den script baseret del er mindre – men den vigtigste i forhold til at få spillet et at virke.

Script i Unity kan godt sidestilles med backend C# scripts, der skal snakke sammen med et frontend element, hvor script kan få input fra hvad der sker med de grafiske elementer og påvirke andre grafiske elementer – en ting der dog er ret specielt for scripts i unity er at de bliver koblet på et grafisk element (camera, spilleren, et tårn) og vil som udgangspunkt kun ”kende” til det element, der er koblet på og de andre komponenter.

Denne opbygning bliver gennemgået når koden til de specifikke problemstillinger bliver gennemgået.

Problemstillinger

Hvordan bruges design mønstre i en game engine som Unity?

Hvordan adskiller implementeringen af design mønstre sig i game engines?

Hvordan virker og implementeres design mønstre, der er opfundet i forbindelse med udviklingen af spil.

* Afgrænsning til de specifikke der kan gennemgåes på de x antal sider.
  + Command (bruger input er oplagt er at bruge det på
  + Observer (cursor – tower win condition mv.)
  + State (kunne måske bruge på computer AI – low health or maybe cd used. – Finite state machine)
  + Spatial Partition Pattern potentielt
  + Singleton I unity - måske

Design mønstre i spil? (observer pattern, singleton pattern – andre?)

<http://gameprogrammingpatterns.com/design-patterns-revisited.html> (Command, Flyweight, Observer, Prototype, Singleton, state)

<https://www.quora.com/What-are-the-10-most-used-design-patterns> - Factory - Singleton, Builder, MVC, Adapter, Proxy, Observer, Filter.

<https://brewhouse.io/2015/10/14/game-programming-design-patterns.html> - Gaming pattern (Command er ikke gang of four design pattern)

<https://www.habrador.com/tutorials/programming-patterns/> - Specific til Unity (Command Pattern, Flyweight pattern, Observer Pttern, State Pattern, Subclass Sandbox Pattern, Spatial Partition Pattern)

Spillet - Game design

De design principper der gennemgås vil blive implementeret i et spil, der er lavet af undertegnet i Unity. Alt er enten lavet af undertegnet og de elementer der ikke er lavet er der licens til, så spillet er fuld lovligt at distribuere og udbrede efter lyst.

Spillet er baseret på League of Legends, men her er man singlespiller mod én modstander. Hver har en base, og hvis denne bliver ødelagt af modstanderen har man tabt.

Man starter i hver sin ende og mellem de to hovedbygninger er der tårn der beskytter sin base. Herudover bliver der løbende dannet bølger af minions der går mod modstanderens tårn/base og slår løs på det som de møder (efter en logik der er defineret i scripts).

Minions bliver stærkere og får mere liv som tiden i spillet går og bliver derved en større og større trussel overfor tårn og base – som ikke får mere liv – mens tårnet dog stiger en lille smule i angrebsstyrke.

Spillerens karakter stiger i level, når der fås en bestemt mængde experience. Experience fås ved at få ”last hit” på en minion, modstanderens dræbes eller tårnet bliver ødelagt.

Efterhånden som spilleren stiger i level fås ekstra abilities og mere styrke som sker i en mere eksponentiel grad i forhold til minions og bliver derfor langsomt stærkere end minions – men det kræver typisk at der fås experience fra last hit.

Ting der skal implementeres:

- Lav click to move færdig – så fjernet walkable fra particles.

Overvej om vi ikke skal have Raycast script på camera alligevel og udstille ”hit” så player movement kan bruge den – da ”andre” måske også skal bruge raycast – og det er noget der hører til på camera.

Brug den fra soulforging som er den forbedre version med event.

- Placer modstander, minions – sæt dem som enemies (layer Enemies) og sæt op animations og basalt script til at bevæge sig til waypoint. – Burde kunne tages meget fra player hvor destination er waypoint.

Sæt lige størrelse så de ser pæne ug

Sæt evt. shader mv. til mobile/diffuse så de passer med landskab

Kig lige på ”Sun” og indstil så alt kan ses.

- Sæt up cursoraffordance -> CursorAnimation -> observer pattern. (Se Repo) - begræns til hand og sværd.

Kig på Udemy course samt ovenstående.

- Find enemies script fra repo og se om det kan sættes op med chase radius mv. (Idamageable) – Byt evt. ud med state pattern script.

- Flyt Tower og nexus til ny empty 3d objekt – så det ikke hænger på enviroment.

- ,marker towers og nexus som enemies og prefab – så de er klar til healthbars mv.

- Lighting (i Lol er der synsfelt omkring minions og tårne – alt andet er skjult.

- fog of war - <https://www.youtube.com/watch?v=N__0TjPoS14>

- Evt. Bare ingen lys på nær lys som er på spiller – minions og bygninger.

- UI – Screenshot af LOL UI og prøv at lav noget tilsvarende i gimP?

-Minimap (2nd camera – bowl master?)

- område med spells (cd fra zafirah spillet) – health/mana (med værdier) – avg. Spellsdamage / attack damage info

- Sæt up animationer (spiller + modstander og minions) – Det der mangler – actions.

Tænk lidt over hvilke spell animationerne passer til. Ideer:

- AoE rundt om én, Stun, Slow, AoE damage til minions clear.

- Attack Square/cone in animation - se Devslope video.

- status – minions killed på hver side – guld farmed på hver side – kills

- Projektiler fra tower (raycast bullets) – måske spawnpoint i midten med et offset som er ud til kanten – så vil den kunne skyde rundt uden at ramme sig selv.

- Spellanimationer. (tænk lidt over hvad man skal kunne af spells – evt. kombi – debuff på enemy)

- Healthbars på tower, nexus, mobs

- Sæt up basal combat.

- minions spawning

- minions stats increase over time (<https://www.youtube.com/watch?v=62IFyHUdH9U>)



Krævede Assets:

# Hvilke chars passer til?

KRAV: ét attack animation skal bruges til mouse button attack (melee attack)

# Player: Toon Dark Elf Mage (4,47) (har et melee auto attack og aoe spell animation – run, die, born, knockback, skill1-3 (1 til auto), stun

AI: Toon Demon (gratis) ((har et melee auto attack og aoe spell animation – run, die, born, knockback, skill 1-3 (1 til auto , 1 til aoe?), stun

PlayerMinion: Toon Ice Elemental (4,47) (death, knockback, skil 1+2, stun, walk, spawn)

AIMinion: Toon Rck (Gratis) - ) (death, knockback NO, skil 1-7, stun, walk, born (Alternative er Toon Armor Warrior – Evt. Kan spiller ikke lave knockbacks?)

Find evt. andre gratis som ”jungle” mobs.

Andre muligheder:

# Dream World series (Death er god 17,87euro stk.) FBX animations – gratis minons (creap og dragon)

# MOBA Game series (10,72-22,33 euro – enkelte er gratis (minions)) FBX animation

* Gratis minion <https://assetstore.unity.com/packages/3d/characters/humanoids/war-assassin-122300>
* Gratis minion <https://assetstore.unity.com/packages/3d/characters/humanoids/sword-knight-pack-122590>
* Rain Entertainment har en serie til 4-9 euro der måske kunne bruges – Nogle er gratis som (mionion only) https://assetstore.unity.com/packages/3d/characters/creatures/rock-warrior-123300

Mechanim (gratis):

<https://assetstore.unity.com/packages/3d/animations/rpg-character-mecanim-animation-pack-free-65284>

# Ekstra

Healthbars – Hjælp:

<https://www.youtube.com/watch?v=CA2snUe7ARM>

(canvas på mob der skal have health – worldspace og scale til 0.01)

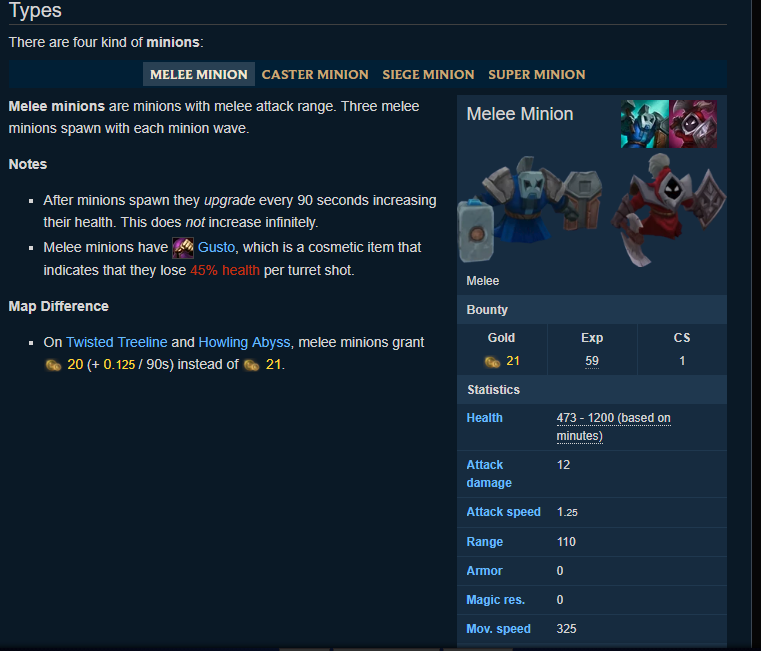
Event – subscriber

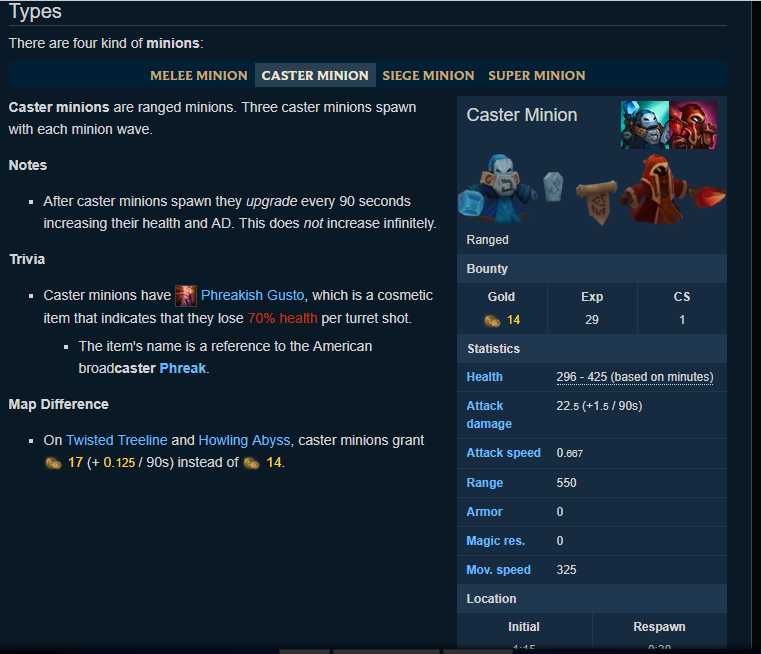
## Minions

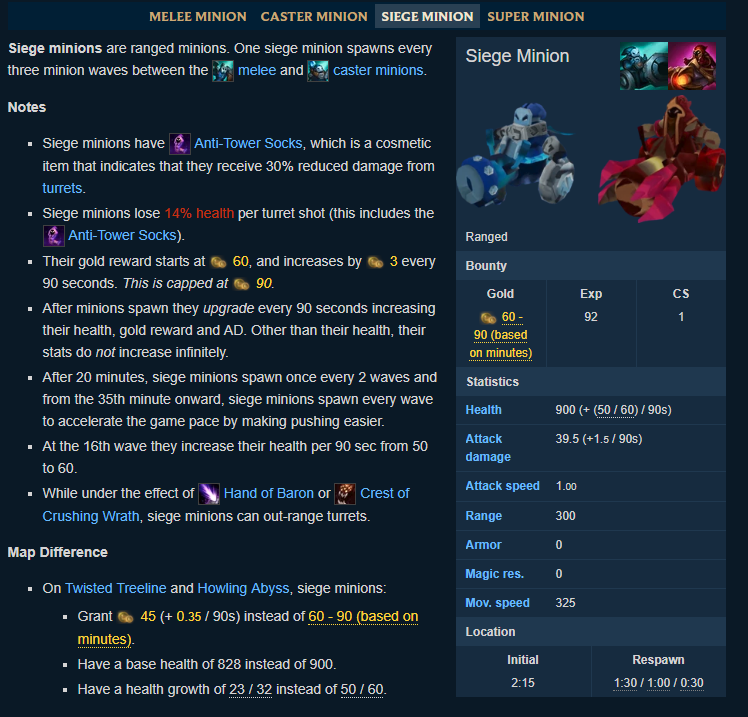
- Burde upgraders linært i mindre grad end player

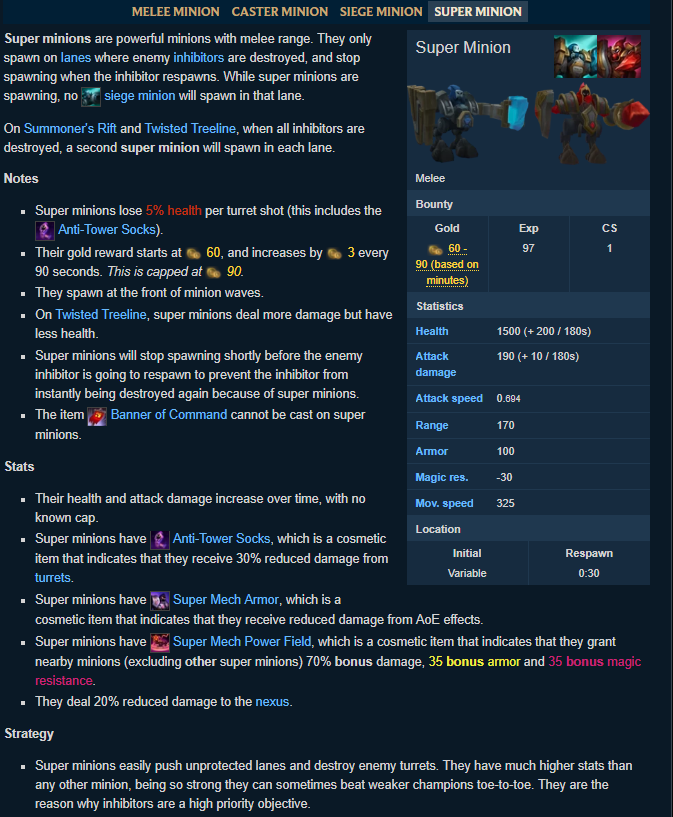
- Taber specifik procent fra tower hit (lol = 45%)

- lidt ekstra guld efterhånden som tiden går







+

-guld fra minions – måske bare xp

-lvl og ekstra attacks (få ekstra abilities - flere stats)

-UI (attacks/keybindings, minimap (camera)

-fog of war - måske bare mørke i gangene. (vision ved tårn - spot light)

- jungle minions? - random spawn

-Musik? -gfx når man laver angreb

-regn (random) - muligvis torden

-Multiplayer i unity?

-sæt sværhedsgrad - både hvad angår modstander - måske også AI niveau?

- trailer – animeret game menu

- fonts: Arial Narrow - Friz Quadrata TT - Morpheus - Skurri - Comic sans(?)

Programmerings.

Singleton – musikspiller.

Observer pattern – cursor -> hvad markere man

Interface på minions måske-

Mob/tower har en queue til enemies? – prioriter player over queue

Waypoints er collection af waypoint der itererer over.

Spawn locations er også en liste måske.

Minions og modstander og mob arver fra en char klasse med health info mv.

Crit, hit mv. fra soulforge spil?

Scripts:

1. mob script – require boxcollider mv. – health, damage – scaling
2. AI script – arver fra mob
3. Player script.
4. Minions script arver fra mob – waypoints – list/stack/queue med enemies?
5. Tower script
6. Nexus script – Simple health – and by maybe event for game loss/win
7. Camera follow script
8. Player movement script (using abilities)
9. Game manager med spawn script?
10. Music manager - singelton
11. Cursor affordance – event based.

Assert i awake.

* Husk at man kan skrive: “Assert.IsNotNull target;” så vil den komme og broke sig hvis man ikke har sat target I inspector (kræver at target er serialized field)

Unit test i Unity

* BowlMaster have Unit test – tænker at man kunne lave den på damage beregninger.

Smarte ting til spillet:

* Brug ”regions” i koden, så man kan expande det man ønsker.
  + #region tekst tekst
  + #endregion
* Lav evt. unit test i Unity – bare enkelte til damage calculation. (Bowling del)
* [Attributes] meta data på public/serializedfields ->
  + [Header(“Overskrift”)]
  + [Space],
  + [Range(min, max)],
  + [Multiline] //bruges på string så man kan se flere linier
  + [TextArea] // renders a flexible textbox with a scrollbar
  + [Tooltip(“text”)] //mousehover tooltip
  + [HideInInspector] //fjerner public fra inspector
* [Attributes der kan sættes på klassen]
  + [HelpURL(”https//www.dr.dk/”)] // bliver til eksternt link
  + [DisallowMultipleComponent] //kan ikke sætte flere af det samme script på et element.
  + [RequireComponent(typeof(Rigidbody))]
  + [ExecuteInEditMode] //kan eksempelvis bruges på camera, hvor det vil følge player også når man er i edit mode.
* [Attributes der sættes på metode kald]
  + [UnityEditor.MenuItem(“Tools/Give Sword/Do it”)] //laver en menu bar “Tools” med et menu punkt “Give Sword” og et under punkt “Do it” – kan eksempelvis bruges til at level en character op I level mens man spiller.

Brackets - RPG courses <https://www.youtube.com/watch?v=FhAdkLC-mSg&t=509s>

Lighting: <https://www.youtube.com/watch?v=VnG2gOKV9dw>

Pause menu: <https://www.youtube.com/watch?v=JivuXdrIHK0>

Options menu: <https://www.youtube.com/watch?v=YOaYQrN1oYQ>

Start Menu (devslopes= <https://www.youtube.com/watch?v=zc8ac_qUXQY>

Power ups: <https://www.youtube.com/watch?v=CLSiRf_OrBk>

Game Grind Unity RPG fra Game Grind <https://www.youtube.com/watch?v=gPc2aDwpggM>

Unity Event tips: <https://www.youtube.com/watch?v=mC_yUEnGQXM>

Unity Gizmos tips: <https://www.youtube.com/watch?v=jljaISrJRz8&t=336s>

Unity Health bar: <https://www.youtube.com/watch?v=GfuxWs6UAJQ>

Damage Popup Text in unity: <https://www.youtube.com/watch?v=fbUOG7f3jq8&t=483s>

Lyd på angreb med sværd: <https://www.youtube.com/watch?v=aZMJJwtnuiA>

Ability CD: <https://www.youtube.com/watch?v=NX8cX3osMFc>

Post Processing: <https://www.youtube.com/watch?v=haz_OUgfPPg>

Power ups : <https://www.youtube.com/watch?v=2Y9J04r6ZL8>

Player Pref: <https://www.youtube.com/watch?v=Cx4vhIeM-7k>

Creating bullet trailing <https://www.youtube.com/watch?v=UDb6KtT7I_E>

Health bar: <https://www.youtube.com/watch?v=yeMOuXiVAos>

Fire guns in unity (tower): <https://www.youtube.com/watch?v=1UekWA1osNw>

Click objects: <https://www.youtube.com/watch?v=EANtTI6BCxk>

Adding music: <https://www.youtube.com/watch?v=lq-5Ws-h0Kc>

Dev assets – Pay as much as you want? [https://bit.ly/2KA6ohq](https://www.youtube.com/redirect?q=https%3A%2F%2Fbit.ly%2F2KA6ohq&event=video_description&v=odLEkNXlWEA&redir_token=9tXVAMTOldlAmiuKFpCl8sDkw3J8MTUzOTk0NjAxNkAxNTM5ODU5NjE2)

**Scripts**

**------SoulForging scripts---------**

**Lav en mappe med Editor – Lyt til RPG og skriv kommentare.**

Indsæt følgende:

using UnityEditor;

// Custom Editor makes it possible to add options to the inspector which would not

// othervise be possible

[CustomEditor(typeof(RaycastFromCamera))]

public class RaycastFromCamera Editor : Editor

{

// This say that the UI Editor we are about to make is unfolded.

bool isLayerPrioritiesUnfolded = true; // store the UI state

// This method override the normale Gui with ours

public override void OnInspectorGUI()

{

// Serialize raycastFromCamera instance

serializedObject.Update();

// If the bool from earlier is changed this menu is expanded with header “Layer Priorities”

isLayerPrioritiesUnfolded = EditorGUILayout.Foldout(isLayerPrioritiesUnfolded, "Layer Priorities");

if (isLayerPrioritiesUnfolded)

{

// This simple indent a linie

EditorGUI.indentLevel++;

{

// To have more clear menu the info is pushed to methods.

BindArraySize();

BindArrayElements();

}

// Returns to the indent as before.

EditorGUI.indentLevel--;

}

// De-serialize back to raycastFromCamera (and create undo point)

serializedObject.ApplyModifiedProperties();

}

// This method creates an array with the size of the Layers we need

void BindArraySize()

{

int currentArraySize = serializedObject.FindProperty("layerPriorities.Array.size").intValue;

int requiredArraySize = EditorGUILayout.IntField("Size", currentArraySize);

if (requiredArraySize != currentArraySize)

{

serializedObject.FindProperty("layerPriorities.Array.size").intValue = requiredArraySize;

}

}

// This method binds the elements to the array from above.

void BindArrayElements()

{

int currentArraySize = serializedObject.FindProperty("layerPriorities.Array.size").intValue;

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

{

var prop = serializedObject.FindProperty(string.Format("layerPriorities.Array.data[{0}]", i));

prop.intValue = EditorGUILayout.LayerField(string.Format("Layer {0}:", i), prop.intValue);

}

}

}

**CameraRaycast – Omdøb til RaycastFromCamera**

using UnityEngine;

using UnityEngine.EventSystems;

using System.Linq;

using System.Collections.Generic;

public class RaycastFromCamera: MonoBehaviour

{

// INSPECTOR PROPERTIES RENDERED BY CUSTOM EDITOR SCRIPT

[SerializeField] int[] layerPriorities;

float maxRaycastDepth = 100f; // Hard coded value

int topPriorityLayerLastFrame = -1; // So get ? from start with Default layer terrain

// Setup delegates for broadcasting layer changes to other classes

public delegate void OnCursorLayerChange(int newLayer); // declare new delegate type

public event OnCursorLayerChange notifyLayerChangeObservers; // instantiate an observer set

public delegate void OnClickPriorityLayer(RaycastHit raycastHit, int layerHit); // declare new delegate type

public event OnClickPriorityLayer notifyMouseClickObservers; // instantiate an observer set

void Update()

{

// Check if pointer is over an interactable UI element

if (EventSystem.current.IsPointerOverGameObject ())

{

NotifyObserersIfLayerChanged (5);

return; // Stop looking for other objects

}

// Raycast to max depth, every frame as things can move under mouse

Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);

RaycastHit[] raycastHits = Physics.RaycastAll (ray, maxRaycastDepth);

RaycastHit? priorityHit = FindTopPriorityHit(raycastHits);

if (!priorityHit.HasValue) // if hit no priority object

{

NotifyObserersIfLayerChanged (0); // broadcast default layer

return;

}

// Notify delegates of layer change

var layerHit = priorityHit.Value.collider.gameObject.layer;

NotifyObserersIfLayerChanged(layerHit);

// Notify delegates of highest priority game object under mouse when clicked

if (Input.GetMouseButton (0))

{

notifyMouseClickObservers (priorityHit.Value, layerHit);

}

}

void NotifyObserersIfLayerChanged(int newLayer)

{

if (newLayer != topPriorityLayerLastFrame)

{

topPriorityLayerLastFrame = newLayer;

notifyLayerChangeObservers (newLayer);

}

}

RaycastHit? FindTopPriorityHit (RaycastHit[] raycastHits)

{

// Form list of layer numbers hit

List<int> layersOfHitColliders = new List<int> ();

foreach (RaycastHit hit in raycastHits)

{

layersOfHitColliders.Add (hit.collider.gameObject.layer);

}

// Step through layers in order of priority looking for a gameobject with that layer

foreach (int layer in layerPriorities)

{

foreach (RaycastHit hit in raycastHits)

{

if (hit.collider.gameObject.layer == layer)

{

return hit; // stop looking

}

}

}

return null; // because cannot use GameObject? nullable

}

}

**Cursor affordance = omdøb til CursorAnimation**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(RaycastFromCamera))]

public class CursorAffordance : MonoBehaviour {

[SerializeField] Texture2D walkCursor = null;

[SerializeField] Texture2D attackCursor = null;

[SerializeField] Vector2 cursorHotspot = new Vector2(0, 0);

[SerializeField] const int walkableLayerNumber = 8;

[SerializeField] const int enemyLayerNumber = 9;

RaycastFromCamera raycastFromCamera;

// Use this for initialization

void Start () {

raycastFromCamera = FindObjectOfType<RaycastFromCamera >();

raycastFromCamera.notifyLayerChangeObservers += OnLayerChanged; //register for changes on layer changes from raycasts.

}

void Update()

{

}

void OnLayerChanged(int newLayer)

{

switch (newLayer)

{

case walkableLayerNumber:

Cursor.SetCursor(walkCursor, cursorHotspot, CursorMode.Auto);

break;

case enemyLayerNumber:

Cursor.SetCursor(attackCursor, cursorHotspot, CursorMode.Auto);

break;

default:

Cursor.SetCursor(walkCursor, cursorHotspot, CursorMode.Auto);

break;

}

}

}

**Enemy Script**

**– Fjern alt med spawn projectile – tilgengæld kan det evt. sættes på tower – arver fra enemy?**

**–Fjern alt med chace mv. – Flyt evt. til minion – der arver fra enemy?**

**– Skal nok lige have importeret fireball prefab.**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityStandardAssets.Characters.ThirdPerson;

public class Enemy : MonoBehaviour, IDamageable {

[SerializeField] float maxHealthPoints = 100f;

[SerializeField] float attackRadius = 3f;

[SerializeField] float chaseRadius = 5f;

[SerializeField] float damagePerShot = 9f;

[SerializeField] float secondsBetweenShots = 1f;

[SerializeField] GameObject projectileToUse;

[SerializeField] GameObject projectileSocket;

[SerializeField] Vector3 aimOffset = new Vector3 (0, 1f, 0);

bool isAttacking = false;

private float currentHealthPoints;

AICharacterControl aICharacterControl = null;

GameObject player;

void Start()

{

currentHealthPoints = maxHealthPoints;

aICharacterControl = GetComponent<AICharacterControl>();

player = GameObject.FindGameObjectWithTag("Player");

}

void Update()

{

float distanceToPlayer = Vector3.Distance(player.transform.position, transform.position);

if (distanceToPlayer <= attackRadius && !isAttacking)

{

isAttacking = true;

InvokeRepeating("SpawnProjectile", 0f, secondsBetweenShots); //TODO switch to Coroutines

}

if (distanceToPlayer > attackRadius)

{

isAttacking = false;

CancelInvoke();

}

if (distanceToPlayer <= chaseRadius)

{

aICharacterControl.SetTarget(player.transform);

}

else

{

aICharacterControl.SetTarget(transform);

}

}

public float healthAsPercentage

{

get

{

return (currentHealthPoints/maxHealthPoints);

}

}

public void TakeDamage(float damage)

{

currentHealthPoints = Mathf.Clamp(currentHealthPoints - damage, 0f, maxHealthPoints);

if (currentHealthPoints <= 0)

{

DestroyObject(gameObject);

}

}

void SpawnProjectile()

{

GameObject newProjectile = Instantiate(projectileToUse, projectileSocket.transform.position, Quaternion.identity) as GameObject;

newProjectile.name = "fireball";

Projectile projectileComponent = newProjectile.GetComponent<Projectile>();

projectileComponent.SetDamage(damagePerShot);

Vector3 playerCenterPosition = player.transform.position + aimOffset;

Vector3 unitVectorToPlayer = (playerCenterPosition - projectileSocket.transform.position).normalized;

float projectileSpeed = projectileComponent.projectileSpeed;

newProjectile.GetComponent<Rigidbody>().velocity = unitVectorToPlayer \* projectileSpeed;

}

void OnDrawGizmos()

{

//Draw attack sphere

Gizmos.color = new Color(255f, 0f, 0f, 0.5f);

Gizmos.DrawWireSphere(transform.position, attackRadius);

//Draw move sphere

Gizmos.color = new Color(255f, 255f, 255f, 0.5f);

Gizmos.DrawWireSphere(transform.position, chaseRadius);

}

}

**Enemy Health Bar – Se om der ikke er en pænere løsning.**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class EnemyHealthBar : MonoBehaviour

{

RawImage healthBarRawImage = null;

Enemy enemy = null;

// Use this for initialization

void Start()

{

enemy = GetComponentInParent<Enemy>(); // Different to way player's health bar finds player

healthBarRawImage = GetComponent<RawImage>();

}

// Update is called once per frame

void Update()

{

float xValue = -(enemy.healthAsPercentage / 2f) - 0.5f;

healthBarRawImage.uvRect = new Rect(xValue, 0f, 0.5f, 1f);

}

}

**EnemyUI – Da man ikke kunne have nested prefabs. Det kan man nu. Brug det i stedet**

using UnityEngine;

public class EnemyUI : MonoBehaviour {

[Tooltip("The UI canvas prefab")]

[SerializeField] GameObject enemyCanvasPrefab = null;

Camera cameraToLookAt;

// Use this for initialization

void Start()

{

cameraToLookAt = Camera.main;

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

}

// Update is called once per frame

void LateUpdate()

{

transform.LookAt(cameraToLookAt.transform);

// transform.rotation = Quaternion.LookRotation(cameraToLookAt.transform.forward);

}

}

**IDamageable**

public interface IDamageable

{

void TakeDamage(float damage);

}

**Player**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Player : MonoBehaviour, IDamageable {

[SerializeField] float maxHealthPoints = 100f;

[SerializeField] int enemyLayer = 9;

[SerializeField] float damagePerHit = 12;

[SerializeField] float minTimeBetweenHits = 1f;

[SerializeField] float maxAttackDistance = 2f;

private float currentHealthPoints;

GameObject currentTarget = null;

CameraRaycaster cameraRayCaster;

float lastHitTime = 0f;

public float healthAsPercentage {get{return (currentHealthPoints/maxHealthPoints);} }

void Start()

{

currentHealthPoints = maxHealthPoints;

cameraRayCaster = FindObjectOfType<CameraRaycaster>();

cameraRayCaster.notifyMouseClickObservers += OnMouseClick;

}

void Update()

{

if (Input.GetKeyDown(KeyCode.Alpha2))

{

Debug.Log("Cast fireballl");

}

}

public void TakeDamage(float damage)

{

currentHealthPoints = Mathf.Clamp(currentHealthPoints - damage, 0f, maxHealthPoints);

if (currentHealthPoints <= 0)

{

DestroyObject(gameObject);

}

}

void OnMouseClick(RaycastHit raycastHit, int layerHit)

{

if (layerHit == enemyLayer)

{

GameObject enemy = raycastHit.collider.gameObject;

currentTarget = enemy;

//Check eemy is in range

if ((enemy.transform.position - transform.position).magnitude > maxAttackDistance)

{

return;

}

var enemyComponent = enemy.GetComponent<Enemy>();

if (Time.time - lastHitTime > minTimeBetweenHits)

{

enemyComponent.TakeDamage(damagePerHit);

lastHitTime = Time.time;

}

}

}

}

**PlayerHealthBar**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

[RequireComponent(typeof(RawImage))]

public class PlayerHealthBar : MonoBehaviour

{

RawImage healthBarRawImage;

Player player;

// Use this for initialization

void Start()

{

player = FindObjectOfType<Player>();

healthBarRawImage = GetComponent<RawImage>();

}

// Update is called once per frame

void Update()

{

float xValue = -(player.healthAsPercentage / 2f) - 0.5f;

healthBarRawImage.uvRect = new Rect(xValue, 0f, 0.5f, 1f);

}

}

**Projectile**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Projectile : MonoBehaviour {

public float projectileSpeed = 10f;

private float damageCaused = 10f;

public void SetDamage(float damage)

{

damageCaused = damage;

}

void OnTriggerEnter(Collider collider)

{

Component damageableComponent = collider.gameObject.GetComponent(typeof(IDamageable));

if (damageableComponent)

{

(damageableComponent as IDamageable).TakeDamage(damageCaused);

}

}

}

**-----------Repo Project.------**

**AIHealthSystem**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

namespace Characters

{

public class AIHealthSystem : MonoBehaviour

{

[SerializeField] Image healthBar;

[SerializeField] AudioClip[] damageSounds;

[SerializeField] AudioClip[] deathSounds;

//[SerializeField] float deathVanishSeconds = 2.0f;

[SerializeField] float resisdencePrLevel = 30f;

[SerializeField] float maxResistance = 360f;

[SerializeField] float hitPenaltyBoss = 0.16f;

[SerializeField] float hitpenalty2levels = 0.05f;

[SerializeField] float hitpenalty1levels = 0.03f;

private float health;

[SerializeField] float doubleDamagePenetration = 100f;

[SerializeField] float crushingBlowChance = 10f;

[SerializeField] float crushingBlowMultiplier = 1.5f;

[SerializeField] float glancingBlowChance = 25f;

[SerializeField] float glancingBlowMultiplier = 0.5f;

GameObject parent;

const string DEATH\_TRIGGER = "Death";

float currentHealthPoints;

//Animator animator;

//AudioSource audioSource;

Character character;

public float healthAsPercentage { get { return currentHealthPoints / health; } }

public float Health { get { return currentHealthPoints; } }

public float ResisdencePrLevel { get { return resisdencePrLevel; } }

public float MaxResistance { get { return maxResistance; } }

public float HitPenaltyBoss { get { return hitPenaltyBoss; } }

public float Hitpenalty2levels { get { return hitpenalty2levels; } }

public float HitPenalty1levels { get { return hitpenalty1levels; } }

public float DoubleDamagePenetration { get { return doubleDamagePenetration; } }

public float CrushingBlowChance { get { return crushingBlowChance; } }

public float GlancingBlowChance { get { return glancingBlowChance; } }

public float CrushingBlowMultiplier { get { return crushingBlowMultiplier; } }

public float GlancingBlowMultiplier { get { return glancingBlowMultiplier; } }

void Start()

{

//animator = GetComponent<Animator>();

//audioSource = GetComponent<AudioSource>();

character = GetComponent<Character>();

health = 220;

currentHealthPoints = health;

}

void Update()

{

UpdateHealthBar();

}

void UpdateHealthBar()

{

if (healthBar) // Enemies may not have health bars to update

{

healthBar.fillAmount = healthAsPercentage;

}

}

public void ReduceHealth(float damage)

{

Debug.Log("ReduceHealth was called with: " + damage);

bool characterDies = (currentHealthPoints - damage <= 0);

currentHealthPoints = Mathf.Clamp(currentHealthPoints - damage, 0f, health);

//var clip = damageSounds[UnityEngine.Random.Range(0, damageSounds.Length)];

//audioSource.PlayOneShot(clip);

if (characterDies)

{

//StartCoroutine(KillCharacter());

Debug.Log("Har ikke implementeret død");

}

}

public void Heal(float points)

{

currentHealthPoints = Mathf.Clamp(currentHealthPoints + points, 0f, health);

}

/\*IEnumerator KillCharacter()

{

characterMovement.Kill();

animator.SetTrigger(DEATH\_TRIGGER);

audioSource.clip = deathSounds[UnityEngine.Random.Range(0, deathSounds.Length)];

audioSource.Play(); // overrind any existing sounds

yield return new WaitForSecondsRealtime(audioSource.clip.length);

var playerComponent = GetComponent<PlayerControl>();

if (playerComponent && playerComponent.isActiveAndEnabled) // relying on lazy evaluation

{

SceneManager.LoadScene(0);

}

else // assume is enemy fr now, reconsider on other NPCs

{

DestroyObject(gameObject, deathVanishSeconds);

}

}\*/

}

}

**Character**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

namespace Characters

{

[SelectionBase]

public class Character : MonoBehaviour

{

//[Header("Capsule Collider")]

//[SerializeField] Vector3 colliderCenter = new Vector3(0, 0, 0);

//[SerializeField] float colliderRadius = 0.5f;

//[SerializeField] float colliderHeight = 2.0f;

//[Header("Nav Mesh Agent")]

//[SerializeField] float navMeshAgentSteeringSpeed = 1.0f;

//[SerializeField] float navMeshAgentStoppingDistance = 1.3f;

[Header("Stats")]

[SerializeField] private float hitChance = 0.96f;

[SerializeField] private float critChance = 0.15f;

[SerializeField] private float critMultiplier = 2.0f;

[SerializeField] private int level = 1;

[SerializeField] private float resistanceToSpells = 90.0f;

[SerializeField] private float penetration = 5.0f;

[SerializeField] private float blockChance = 0.05f; //dodge/Parry?

[SerializeField] private float inteligence = 25f;

[SerializeField] private float agility = 5f;

[SerializeField] private float strength = 10f;

[SerializeField] private float stamina = 22f;

public float HitChance { get { return hitChance; } }

public float CritChance { get { return critChance; } }

public float CritMultiplier { get { return critMultiplier; } }

public int Level { get { return level; } }

public float ResistanceToSpells { get { return resistanceToSpells; } }

public float Penetration { get { return penetration; } }

public float BlockChance { get { return blockChance; } }

public float Inteligence { get { return inteligence; } }

public float Agility { get { return agility; } }

public float Strength { get { return strength; } }

public float Stamina { get { return stamina; } }

//NavMeshAgent navMeshAgent;

Animator animator;

Rigidbody ridigBody;

void Awake()

{

AddRequiredComponents();

}

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

private void AddRequiredComponents()

{

//var capsuleCollider = gameObject.AddComponent<CapsuleCollider>();

//capsuleCollider.center = colliderCenter;

//capsuleCollider.radius = colliderRadius;

//capsuleCollider.height = colliderHeight;

ridigBody = gameObject.AddComponent<Rigidbody>();

ridigBody.constraints = RigidbodyConstraints.FreezeRotation;

ridigBody.mass = 80;

//navMeshAgent = gameObject.AddComponent<NavMeshAgent>();

//navMeshAgent.speed = navMeshAgentSteeringSpeed;

//navMeshAgent.stoppingDistance = navMeshAgentStoppingDistance;

//navMeshAgent.autoBraking = false;

//navMeshAgent.updateRotation = false;

//navMeshAgent.updatePosition = true;

}

}

}

**Damage**

**(burde nok laves som en DamageCalculation der implementere IDamageCalculation.**

using Characters;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Damage : MonoBehaviour, IDamageable

{

AIHealthSystem aIHealthSystem;

Character character;

Player player;

float hitChance;

// Use this for initialization

void Start () {

//this.GetComponent<AudioSource>().PlayOneShot(audioLaunch);

//info from this components gameObject

character = GetComponent<Character>();

//info from attacker.

player = GameObject.FindObjectOfType<Player>();

//HealthSystem

aIHealthSystem = GetComponent<AIHealthSystem>();

}

// Update is called once per frame

void Update () {

}

public void TakeDamage(float damage, bool isMagical)

{

//Debug.Log("Damage in Damage - TakeDamage start at: " + damage);

float damageDone = CalculateDamage(damage, isMagical);

aIHealthSystem.ReduceHealth(damageDone);

}

private float CalculateDamage(float damage, bool isMagical)

{

float hitMultiplier = HitMultiplier();

float multiplierOnCrit = MultiplierOnCrit();

float damageDiffFromPenetrationAndResistance = DamageDiffFromPenetrationAndResistance(isMagical);

float damageDiffFromMeleeCombat = DamageDiffFromMeleeCombat(isMagical);

float damageDone = damage \* hitMultiplier \* multiplierOnCrit \* damageDiffFromPenetrationAndResistance \* damageDiffFromMeleeCombat;

//Debug.Log("hitMultiplier: " + hitMultiplier);

//Debug.Log("multiplierOnCrit: " + multiplierOnCrit);

//Debug.Log("damageDiffFromPenetration: " + damageDiffFromPenetrationAndResistance);

//Debug.Log("damageDiffFromMeleeCombat: " + damageDiffFromMeleeCombat);

return damageDone;

}

private int levelDifference()

{

return (character.Level - player.Level);

}

private float HitMultiplier()

{

bool isBosslevel = (character.Level - 2 > player.Level);

bool is2HigherLevel = (character.Level - 2 == player.Level);

bool is1HigherLevel = (character.Level - 1 == player.Level);

if (isBosslevel)

{

hitChance = player.HitChance - aIHealthSystem.HitPenaltyBoss;

}

else if (is2HigherLevel)

{

hitChance = player.HitChance - aIHealthSystem.Hitpenalty2levels;

}

else if (is1HigherLevel)

{

hitChance = player.HitChance - aIHealthSystem.HitPenalty1levels;

} else

{

hitChance = player.HitChance;

}

float randomNumber = Random.Range(0, 101);

if (randomNumber > (hitChance\*100))

{

return 0;

}

else

{

return 1;

}

}

private float MultiplierOnCrit()

{

float randomNumber = Random.Range(0, 101);

if (randomNumber > (player.CritChance\*100))

{

return 1;

}

else

{

return 2;

}

}

private float DamageDiffFromPenetrationAndResistance(bool isMagical)

{

if (isMagical)

{

float resistance = character.ResistanceToSpells + levelDifference() \* aIHealthSystem.ResisdencePrLevel;

return (1 - (resistance / aIHealthSystem.MaxResistance) + (character.Penetration/aIHealthSystem.DoubleDamagePenetration));

//eksempel 1 - 90/360 + 5/100 = 1-0,25+0,05 = 0,80

//eksempel 2 - 0/360 + 30/100 = 1-0+0,3 = 1,3

//target 58 - inflicter 60 / resist 90 / penn 20

//eksempel - (90-2\*30)/360 + 20/100 = 1-0,0833+0,2 = 1,2833

} else

{

return 1;

}

}

private float DamageDiffFromMeleeCombat(bool isMagical)

{

if (isMagical == false)

{

float randomNumber = Random.Range(0, 100);

float blockChance = character.BlockChance;

if ((character.Level - 2) > player.Level)

{

//crushingBlow possible

if (randomNumber < (aIHealthSystem.CrushingBlowChance))

{

return aIHealthSystem.CrushingBlowMultiplier;

}

else

{

return 1; //could be crushing hit - but is not

}

} else if ((character.Level + 2) < player.Level)

{

//GlancingBlow possible

if (randomNumber < (aIHealthSystem.GlancingBlowChance))

{

return aIHealthSystem.GlancingBlowMultiplier;

}

else

{

return 1; //could be Glanicing - but is not

}

} else

{

return 1; //Can't be either glancing or crushing

}

} else

{

return 1; //Is not melee damage.

}

}

}

EnemyUI

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyUI : MonoBehaviour

{

[Tooltip("The UI canvas prefab")]

[SerializeField] GameObject enemyCanvasPrefab = null;

Camera cameraToLookAt;

// Use this for initialization

void Start()

{

cameraToLookAt = Camera.main;

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

}

// Update is called once per frame

void LateUpdate()

{

transform.LookAt(cameraToLookAt.transform);

transform.rotation = Quaternion.LookRotation(cameraToLookAt.transform.forward);

}

}

**Fireball**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Characters;

public class Fireball : MonoBehaviour {

public float speed = 5.0f;

public bool isMagical = true;

public float minDamage = 35.0f;

public float maxDamage = 50.0F;

public ParticleSystem particle = null;

public GameObject spotLight;

//public AudioClip audioHit;

//public AudioClip audioLaunch;

private bool canMove = true;

GameObject target = null;

MouseManager mouseManager;

void Awake()

{

}

// Use this for initialization

void Start () {

//this.GetComponent<AudioSource>().PlayOneShot(audioLaunch);

//mouseManager

mouseManager = GameObject.FindObjectOfType<MouseManager>();

//info from target.

if (mouseManager.selectedObject != null)

{

target = mouseManager.selectedObject;

}

}

// Update is called once per frame

void Update () {

MoveObject();

}

void MoveObject()

{

if (canMove)

{

//transform.LookAt(target.transform);

transform.Translate(0.0f, 0.0f, speed \* Time.deltaTime);

}

}

void OnTriggerEnter (Collider collider)

{

Component damagableComponent = collider.gameObject.GetComponent(typeof(IDamageable));

if (damagableComponent)

{

float damageDone = DamageFromMinMax(minDamage, maxDamage);

Debug.Log("damage from fireball: " + damageDone);

(damagableComponent as IDamageable).TakeDamage(damageDone, true);

}

//this.GetComponent<AudioSource>().PlayOneShot(audiotHit);

GetComponent<Renderer>().enabled = false;

GetComponent<Collider>().enabled = false;

GetComponentInChildren<ParticleSystem>().Stop(true);

spotLight.SetActive(false);

canMove = false;

Destroy(this.gameObject,3.0f);

}

private float DamageFromMinMax(float minDamage, float maxDamage)

{

return Random.Range(minDamage, maxDamage);

}

}

**Idamageable**

public interface IDamageable{

void TakeDamage(float damageDone, bool isMagical);

}

**Player**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

namespace Characters

{

[SelectionBase]

public class Player : MonoBehaviour

{

//[Header("Capsule Collider")]

//[SerializeField] Vector3 colliderCenter = new Vector3(0, 0, 0);

//[SerializeField] float colliderRadius = 0.5f;

//[SerializeField] float colliderHeight = 2.0f;

//[Header("Nav Mesh Agent")]

//[SerializeField] float navMeshAgentSteeringSpeed = 1.0f;

//[SerializeField] float navMeshAgentStoppingDistance = 1.3f;

[Header("Stats")]

[SerializeField] private float hitChance = 0.96f;

[SerializeField] private float critChance = 0.15f;

[SerializeField] private float critMultiplier = 2.0f;

[SerializeField] private int level = 1;

[SerializeField] private float resistanceToSpells = 90.0f;

[SerializeField] private float penetration = 5.0f;

[SerializeField] private float blockChance = 0.05f; //dodge/Parry?

[SerializeField] private float inteligence = 25f;

[SerializeField] private float agility = 5f;

[SerializeField] private float strength = 10f;

[SerializeField] private float stamina = 22f;

public float HitChance { get { return hitChance; } }

public float CritChance { get { return critChance; } }

public float CritMultiplier { get { return critMultiplier; } }

public int Level { get { return level; } }

public float ResistanceToSpells { get { return resistanceToSpells; } }

public float Penetration { get { return penetration; } }

public float BlockChance { get { return blockChance; } }

public float Inteligence { get { return inteligence; } }

public float Agility { get { return agility; } }

public float Strength { get { return strength; } }

public float Stamina { get { return stamina; } }

//NavMeshAgent navMeshAgent;

Animator animator;

Rigidbody ridigBody;

void Awake()

{

AddRequiredComponents();

}

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

private void AddRequiredComponents()

{

//var capsuleCollider = gameObject.AddComponent<CapsuleCollider>();

//capsuleCollider.center = colliderCenter;

//capsuleCollider.radius = colliderRadius;

//capsuleCollider.height = colliderHeight;

ridigBody = gameObject.AddComponent<Rigidbody>();

ridigBody.constraints = RigidbodyConstraints.FreezeRotation;

ridigBody.mass = 80;

//navMeshAgent = gameObject.AddComponent<NavMeshAgent>();

//navMeshAgent.speed = navMeshAgentSteeringSpeed;

//navMeshAgent.stoppingDistance = navMeshAgentStoppingDistance;

//navMeshAgent.autoBraking = false;

//navMeshAgent.updateRotation = false;

//navMeshAgent.updatePosition = true;

}

}

}

**PlayerControl**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace Characters

{

public class PlayerControl : MonoBehaviour {

public Transform parent;

public Transform castSocket;

public GameObject fireball;

public GameObject fireballCast;

// Use this for initialization

void Start() {

}

// Update is called once per frame

void Update() {

if (Input.GetKeyDown(KeyCode.Alpha1))

{

StartCoroutine(CastFireball());

}

}

void LaunchFireball()

{

GameObject obj = Instantiate(fireball, castSocket.position, castSocket.rotation, parent) as GameObject;

obj.name = "fireball";

}

IEnumerator CastFireball()

{

StartCoroutine(CastingFireball());

yield return new WaitForSeconds(1.5f);

LaunchFireball();

}

IEnumerator CastingFireball()

{

GameObject obj = Instantiate(fireballCast, castSocket.position, castSocket.rotation) as GameObject;

yield return new WaitForSeconds(1.5f); //todo affected by haste

DestroyObject(obj);

}

}

}

---OLD----

Click to move

private void ClickToMove()

{

// Creating a Ray instance which is a ray from camera(marked as main in tag) to the mouseposition

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

// Creating a helper ray in scene view so i can see it - but can't be seen in the game.

Debug.DrawRay(ray.origin, ray.direction \* distanceToBackground, Color.blue);

// Create an paramter for the Out paramter in the Physics Raycast later - Out paramters is a code smell

// but the documentation state this is how to do it.

RaycastHit hit;

// Now if left moust button is pressed down the If sentence is true

if (Input.GetMouseButtonDown(0))

{

// This if statement calls Physics.Raycast which take the ray from camera to mouse point

// and if that ray hits a target before it has traveled to the "distanceToBackground" it return true and return the "hit" to the out paramter.

if (Physics.Raycast(ray, out hit, distanceToBackground, walkable))

{

// here the ray hit something and the navMeshAgents destination is set to that point - meaning it will start travelling to the point

navMeshAgent.SetDestination(hit.point);

Debug.DrawRay(transform.position, hit.point, Color.green);

}

}

// Here a method is called to handle the animaton of the character

AnimateIfRunning();

}

private void AnimateIfRunning()

{

// If the character is travelling the bool is set to true.

if (navMeshAgent.remainingDistance <= navMeshAgent.stoppingDistance)

{

isRunning = false;

}

else

{

isRunning = true;

}

//Setting the Bool in the animation controller equal to the value of the bool in this script.

animator.SetBool("isRunning", isRunning);

}