**Smartlearning VS AI**

[Screenshot]

En rapport om design principper i spil og hvordan de kan implementeres.

Indholdsfortegnelse

[Krav til opgave: 5](#_Toc529367155)

[Opbygning af rapport: 7](#_Toc529367156)

[Indledning 7](#_Toc529367157)

[Problemstilling 7](#_Toc529367158)

[Formål 8](#_Toc529367159)

[Metode 8](#_Toc529367160)

[Forudsætningsafsnit 8](#_Toc529367161)

[Hvad er Unity? 8](#_Toc529367162)

[Spillet - Game design 9](#_Toc529367163)

[Hovedafsnit. 9](#_Toc529367164)

[Observerpattern. 9](#_Toc529367165)

[Command pattern 10](#_Toc529367166)

[State pattern. 10](#_Toc529367167)

[Flyweight pattern. 10](#_Toc529367168)

[Konklusion 11](#_Toc529367169)

[Ting der skal implementeres: 12](#_Toc529367170)

[Ideer til håndtering af combat elementer 13](#_Toc529367171)

[Krævede Assets: 18](#_Toc529367172)

[Scripts: 25](#_Toc529367173)

[Ekstra kodningskrydder 26](#_Toc529367174)

[Scripts: SmartlearningVsAI 29](#_Toc529367175)

[ClickHandler script 29](#_Toc529367176)

[Script interface: IDamageable 32](#_Toc529367177)

[Script interface: IDie 32](#_Toc529367178)

[Script: Respawn class 32](#_Toc529367179)

[Script: EnemyAI class 33](#_Toc529367180)

[Script: Tower 35](#_Toc529367181)

[Script: Minion 37](#_Toc529367182)

[Script: Stats 39](#_Toc529367183)

[Script: Spawn point 41](#_Toc529367184)

[Script: Nexus 42](#_Toc529367185)

[Script: HealthBar 43](#_Toc529367186)

[Script: ManaBar 44](#_Toc529367187)

[Script: Abstract damageCalculation 47](#_Toc529367188)

[Script: GameManager 48](#_Toc529367189)

[Script: AIWin 48](#_Toc529367190)

[Script: PlayerWin 48](#_Toc529367191)

[Script: PlayerControl 49](#_Toc529367192)

[Scripts: SoulForging scripts 51](#_Toc529367193)

[Enemy Script 51](#_Toc529367194)

[Player 52](#_Toc529367195)

[Projectile 54](#_Toc529367196)

[Script: Repo Project 56](#_Toc529367197)

[AIHealthSystem 56](#_Toc529367198)

[Character 59](#_Toc529367199)

[Damage 61](#_Toc529367200)

[Fireball 67](#_Toc529367201)

[Idamageable 70](#_Toc529367202)

[Player 70](#_Toc529367203)

[PlayerControl 73](#_Toc529367204)

# 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.

# Opbygning af rapport:

# Indledning

(Indledningen er opgavens vigtigste afsnit — det er her, at opgavens idé og design begrundes og Forklares)

## Problemstilling

Problemstilling

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)

## Formål

Formål

## Metode

Metode inkl. Afgrænsning (synsvinkel, valg, kvalificeringer,afgrænsninger) Her redegøres for opgavens ‘fremgangsmåde’, dvs. hvorfor man vælger at fokusere på nogle aspekter, og hvorfor man ser bort fra andre. Positive (til-)valg kvalificeres i så høj grad som muligt, mens der samtidig argumenteres for eventuelle afgrænsninger og dermed fravalg, fx med hensyn til empirisk materiale. Desuden kan der foretages begrebsdefinitioner og valg af teoretisk ståsted.

# Forudsætningsafsnit

Det er vigtigt at have kendskab til to ting, for at forstå denne rapport. Den første er at have en forståelse af hvad Unity er, og hvilket spil der bygges, hvor disse design mønstre bliver implementeret i.

## 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, det er koblet på og de andre komponenter der er sat på samme element.

I nedenstående billede kan man se en ”inspector”-fane, som består af de komponenter, der er sat på det enkelte element. Her ses de komponenter, der er sat på spilleren i spillet.

[INDSÆT BILLEDE MED INSPECTOR]

## 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 grafiske elementer, der ikke er lavet af undertegnet, 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 tårne 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.

# Hovedafsnit.

Få lavet en god måde at indsætte billeder med overskrift på billede og caption tekst under billedet, så det ser lidt professionelt ud.

## Observerpattern.

Hvad er det?

Hvordan virker det normalt

Hvordan virker det i spil og hvordan er det implementeret i mit spil.

Der er en update funktion i Unity, som køres igennem hvert frame i spillet. Her vil man kunne placere kode, der skal afvikles hele tiden for eksempelvis at undersøge om et kriterie er opfyldt. Det kunne eksempelvis være om musemarkøren er over en fjende, da markøren i det tilfælde skal ændre sig til et sværd.

Her kan observer design pattern bruges, til at publish en event, når markøren er over en fjende. Så kan script i musemarkøren simplet subscribe til denne information og kan nøjes med at afvikle script når event publiseres.

I spil er der ofte situationer, hvor det er hændelser der påvirker adfærd i andre enheder. Eksempelvis at det er ødelæggelsen af nexus der påvirker at spillet slutter, det er reduktionen af liv eller mana, der påvirker at en healthbar/manabar bliver reduceret. Observer pattern brugt i disse situationer i spillet og kunne bruges i mange andre features der kunne implementeres i spillet.

## Command pattern

Hvad er det

Hvordan virker det i spil og hvordan er det implementeret i mit spil.

(Abilities remapping)

## State pattern.

Hvad er det?

Hvordan virker det normalt

Hvordan virker det i spil og hvordan er det implementeret i mit spil.

(Enemy AI.)

## Flyweight pattern.

Hvad er det?

Hvordan virker det normalt

Hvordan virker det i spil – det sker implisit i Unity via prefabs – Forklar det med transform, mesh mv.

# Konklusion

Svar på problemformuleringen.

# Ting der skal implementeres:

- 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.

- Spawning af minions. (game manager)

- Lav game manager til spawning af minions, player og AI

- Playerwin/AiWin script på nexus til end game scene -> Visuelt eller scene?

- Lav Command pattern til user input.

- Lav State pattern til AI logik

- 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?) – ikke nødvendig når der ikke er teamfight.

- 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

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

- Sæt up basal combat.

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

* Skal der laves menuer, music mv. når basal combat er på plads.

## Ideer til håndtering af combat elementer

Minions.

IsBegingAttacked bliver kaldt af player sammen med TakeDamage når minions angribes. Det kan bruges til at få minion til angribe player – ellers angriber den bare den tætteste modstander. Når der ikke angribes nogle – kan den søge om der er nogle i en afstand – Hvis ikke så går den efter næste punkt på en prioriteret liste (stack?)

Logik

inions walk along a lane until they find an enemy unit. When they find an enemy unit, they attack it. An enemy champion can **aggro** the minion if they basic attack or use a targeted ability or spell on a nearby allied champion. When they find several enemy units they choose their target using a priority system that follows this order:

1. Enemy champions attacking an allied champion.
2. Enemy minions attacking an allied champion.
3. Enemy minions attacking an allied minion.
4. Enemy turrets attacking an allied minion.
5. Enemy champions attacking an allied minion.
6. The closest enemy minion.
7. The closest enemy champion.

Once a minion has chosen a target, it only switches to a new target if the new target has higher priority. If they see a new target with the same priority as their current target, they keep attacking their current target.

Minions reevaluate their target every few seconds. For instance, if an enemy champion attacks an allied champion and then stops attacking, minions will keep targeting the enemy champion for a short time after he stops attacking.

When the target of a minion leaves its sight, the minion switches to a new target or keeps advancing if there are no targets within sight.

AIModstander

Start med at lade den løbe mod tower og angrib -> State driven senere.

Player

Command design pattern på special abilities.

Healer

Lav Spawn point til et område hvor der er healing. Se RPG course – hvor der er healer (Bare et script der kalder TakeDamage med negativt værdi, når player er indenfor et område) – Kunne lave en sphere collider der kunne lave event på at player er i mål zonen / skal også være en der siger at man er udenfor?)

Bare tomt gameelement med en omkreds og script til healing.

HealthValues

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | 1 | 2 | 3 | 4 | 5 |
| Player (173%) | 570 | 986 | 1402 | 1818 | 2236 |
| AI | 570 | 986 | 1402 | 1818 | 2236 |
| Minions | 473 | 655 | 837 | 1018 | 1200 |
| Tower | 3800 |  |  |  |  |
| Nexus | 550 |  |  |  |  |

ManaValues

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | 1 | 2 | 3 | 4 | 5 |
| Player (149%) | 317 | 472 | 627 | 782 | 940 |
| AI | 317 | 472 | 627 | 782 | 940 |

Attack Damage Values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Level | 1 | 2 | 3 | 4 | 5 |
| Player (160%) | 58 (0.658 speed) | 92 | 147 | 235 | 376 |
| AI | 58 (0.658 speed) | 92 | 147 | 235 | 376 |
| Minions (150%) | 12 (1.25 speed) | 18 (1.25 speed) | 27 (1.25 speed) | 40 (1.25 speed) | 60 (1.25 speed) |
| Tower | 152 (0.83 speed) | +4 AD every 30 sec – max 180 | |  |  |
| Nexus | 0 |  |  |  |  |

Minions lose 45% max HP from tower shots.

Minions do 40% reduced damage against enemy champion

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

VFX – muligheder:

<https://assetstore.unity.com/packages/vfx/particles/spells/fantastic-cartoon-fx-100604>

<https://assetstore.unity.com/packages/vfx/particles/spells/ky-magic-effects-free-21927>

<https://assetstore.unity.com/packages/vfx/particles/spells/battle-skill-fx-62242> - nogle er gode til aoe

<https://assetstore.unity.com/packages/vfx/particles/spells/magical-ultimate-91406> - Meteor til aoe? – dyr pakke.

<https://assetstore.unity.com/packages/vfx/particles/spells/buff-loop-fx-34104> - gratis og har lyd og

Music – ambient

[https://assetstore.unity.com/packages/audio/ambient/fantasy/swords-sorcery-rpg-music-lite-57360 - 0,89](https://assetstore.unity.com/packages/audio/ambient/fantasy/swords-sorcery-rpg-music-lite-57360%20-%200,89) euro

<https://assetstore.unity.com/packages/audio/ambient/fantasy/warped-fantasy-music-pack-49914> - Ret god og graits.

<https://assetstore.unity.com/packages/audio/ambient/fantasy/dark-fantasy-music-pack-1-121547> - ganske god - gratis

<https://assetstore.unity.com/packages/audio/ambient/fantasy/dark-fantasy-music-pack-2-mini-pack-124182> - god og gratis.

Bars – graphics – måske:

<https://assetstore.unity.com/packages/2d/gui/icons/50-progress-bars-pack-4-dangerous-progress-94371>

1,77 euro – husk også en lvl graphic.

Minions

- Burde upgraders linært i mindre grad end player

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

- lidt ekstra guld efterhånden som tiden går



# 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:

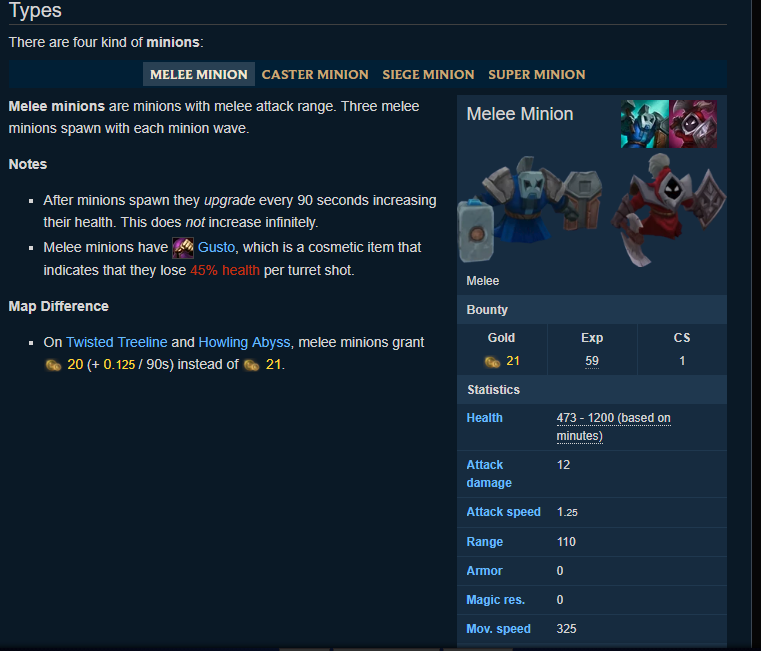
Dream World series (Death er god 17,87euro stk.) FBX animations – gratis minons (creap og dragon)

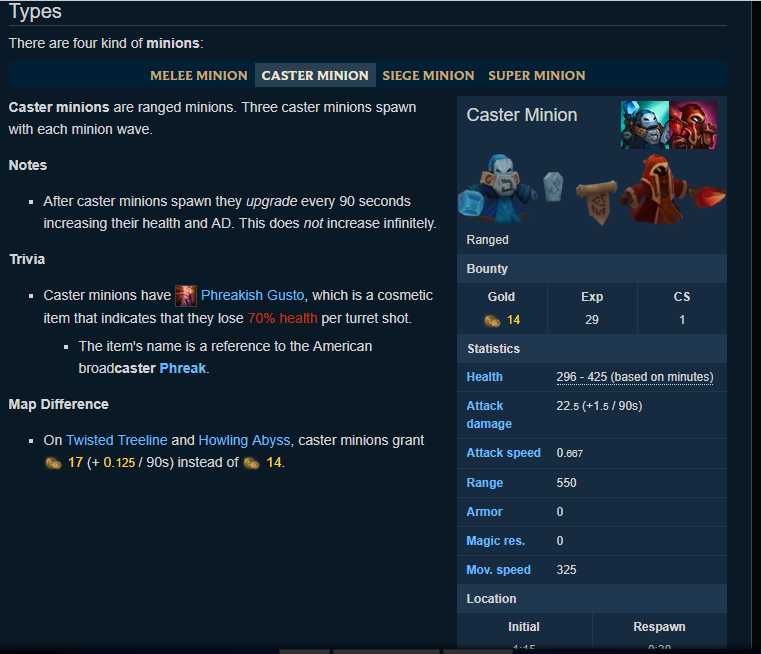
MOBA Game series (10,72-22,33 euro – enkelte er gratis (minions)) FBX animation

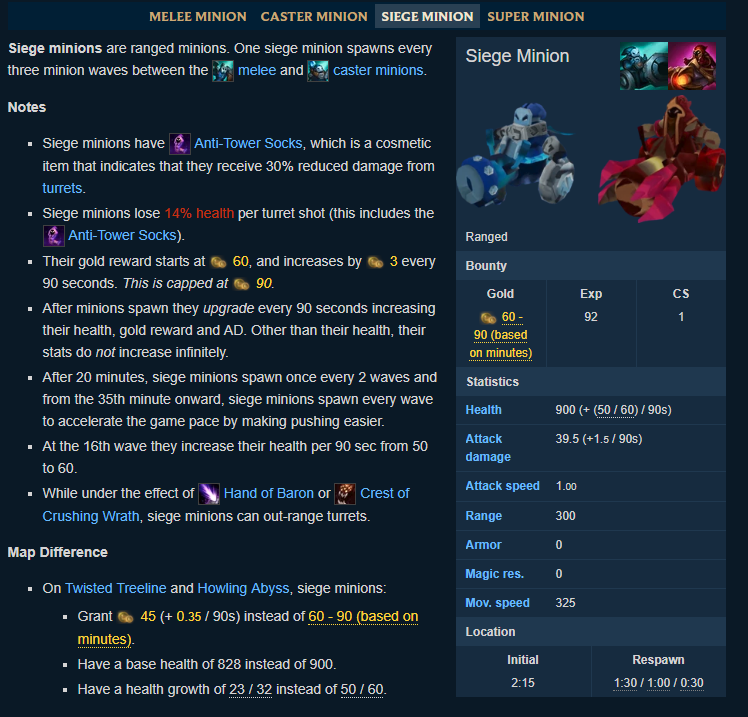
* Gratis minion <https://assetstore.unity.com/packages/3d/characters/humanoids/war-assassin-122300>
* Gratis minion <https://assetstore.unity.com/packages/3d/characters/humanoids/sword-knight-pack-122590>
* Rain Entertainment har en serie til 4-9 euro der måske kunne bruges – Nogle er gratis som (mionion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

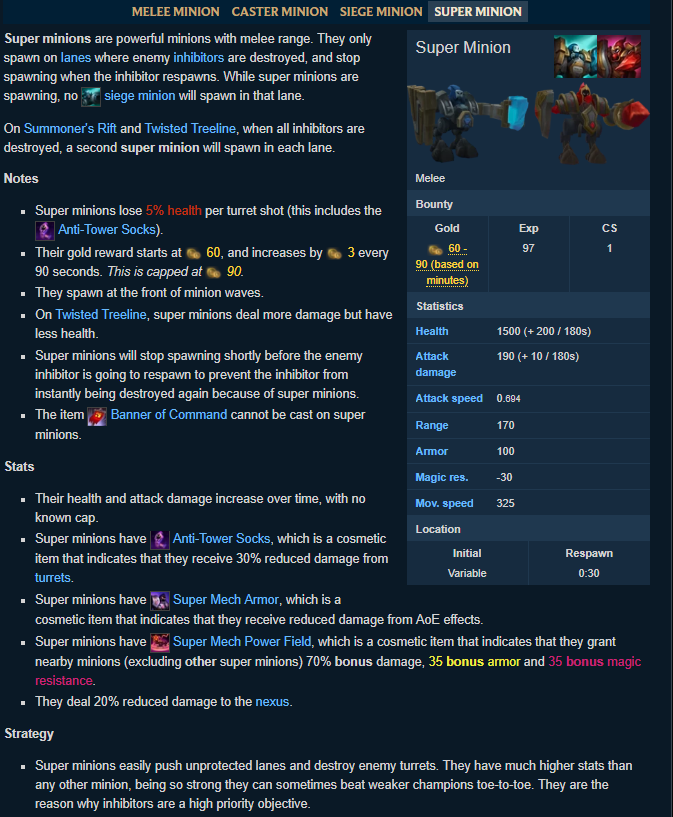
Mechanim (gratis):

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







+

-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.

# Ekstra kodningskrydder

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)

Gizmos:

Visualization of code in scene view – Can also be enabled to view in game view.:

// Draw a linie from “from” position to the “to” position.

// It set a black cirkel at “hvor”

Gizmos.color = Color.black;

Gizmos.DrawLine(from, to)

Gizmos.DrawSphere(hvor, størelse på cirkel)

//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);

# Scripts: SmartlearningVsAI

## ClickHandler script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ClickHandler : MonoBehaviour {

[SerializeField] int walkLayer = 9;

[SerializeField] int enemyLayer = 11;

[SerializeField] float minTimeBetweenHits = 1f;

[SerializeField] float damagePerHit = 35f;

Private float lastHitTime;

Private bool isRunning = false;

//GameObject currentTarget = null;

CameraRaycaster cameraRayCaster;

NavMeshAgent navMeshAgent;

Animator animator;

void Start()

{

cameraRayCaster = FindObjectOfType<CameraRaycaster>();

animator = FindComponentOfType<Animator>();

navMeshAgent = FindObjectOfType<NavMeshAgent>();

// Subscriber registers for info from notifyLeftMouseClickObservers and says

// that when notifyLeftMouseClickObservers is called the method “ClickToMove” should be called.

cameraRayCaster.notifyLeftMouseClickObservers += ClickHandler;

}

void Update()

{

// FLYT TIL METODEKALD

// if at the stopping distance or less the animator is changed to idle

// if the destination is farter away than the stopping distance the animation

// is set to running.

if (navMeshAgent.remainingDistance <= navMeshAgent.stoppingDistance)

{

isRunning = false;

}

else

{

isRunning = true;

}

// Setting the animation bool to the value of script bool.

Animator.SetBool(“isRunning”, isRunning)

}

// The Method which is called through the delegate

// on the raycastFromCamera when left mouse is clicked

void ClickHandler (RaycastHit raycastHit, int layerHit)

{

// No matter what is click the player runs to the walkable target or the enemy.

// Stopping at the stopping distance.

navMeshAgent.destination = raycastHit.point;

// If the priority layer was an enemy layer an additiona check id performed

if (layerHit == enemyLayer)

{

// Saves the enemy hit by the ray.

GameObject enemy = raycastHit.collider.gameObject;

// If the enemy clicked is Out of range I leave the method

if (enemy.transform.position - transform.position).magnitude > navMeshAgent.stoppingDistance)

{

Return;}

}

// This will only be run if the enemy is in range

// Get the component of the enemy that have a takeDamage method.

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

if (Time.time - lastHitTime > minTimeBetweenHits)

{

StartCoroutine(AutoAttack());

}

}

}

IEnumerator AutoAttack()

{

lastHitTime = Time.time;

animator.play(”Dark\_elf\_autoAttack”);

yield return new WaitForSeconds(0.5f);

enemyComponent.TakeDamage(damagePerHit);

yield return new WaitForSeconds(0.5f);

}

Void OnDrawGizmos()

{

// Draw a linie from “from” position to the “to” position.

// It set a black cirkel at navMeshAgent.destination

Gizmos.color = Color.black;

Gizmos.DrawLine(transform.position, navMeshAgent.destination??)

Gizmos.DrawSphere(navMeshAgent.destination, 0.2f)

}

}

## Script interface: IDamageable

public interface IDamageable

{

void TakeDamage(float damage);

}

## Script interface: IDie

public interface IDie

{

void Die();

}

## Script: Respawn class

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Respawn : MonoBehaviour, IDie {

[SerializedField] float waitTimeBeforeDeath = 1f; //can vary depending on animation.

[SerializedField] private float dieCount = 0f;

[SerializedField] private float WaitPrDeath = 5f;

[SerializedField] Private String die = ”Die”; // TODO

[SerializedField] Private String born = ”Born”; // TODO

[SerializedField]Transform spawnPoint;

Stats stats;

private Renderer renderer;

Animator animator;

void Awake()

{

Assert.IsNotNull(spawnPoint);

}

void Start()

{

animator = FindComponentOfType<Animator>();

renderer = GetComponent<Renderer>();

renderer.enabled = true;

stats = GetComponent<Stats>();

}

public void Die ()

{

StartCouroutine(CharDie())

dieCount++;

}

Private virtual IEnumerator CharDie()

{

animator.play(die);

yield return new WaitForSeconds(waitTimeBeforeDeath);

renderer.enabled = false;

transform.position = spawnPoint.position;

yield return new WaitForSeconds(dieCount\* WaitPrDeath);

stats.Respawn();

renderer.enabled = true;

animator.play(born);

}

}

## Script: EnemyAI class

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyAI : MonoBehaviour {

[SerializedField] float waitTimeBeforeDeath = 1f; //can vary depending on animation.

[SerializeField] float attackRadius = 3f;

[SerializeField] float chaseRadius = 5f;

[SerializeField] float damagePerHit = 35f;

[SerializeField] float minTimeBetweenHits = 1f;

Private float lastHitTime;

Private float autoAttackAnimationTime = 0.5f // TODO

Private String autoAttack = ”Fallen\_???\_autoAttack”; // TODO

GameObject player;

NavMeshAgent navMeshAgent;

Animator animator;

void Start()

{

player = GameObject.FindGameObjectWithTag("Player");

animator = FindComponentOfType<Animator>();

navMeshAgent = FindObjectOfType<NavMeshAgent>();

}

void Update()

{

// Move to method

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

if (distanceToPlayer <= attackRadius)

{

// This will only be run if the enemy is in range

// Get the component of the enemy that have a takeDamage method.

var playerComponent = player.GetComponent<Stats>();

if (Time.time - lastHitTime > minTimeBetweenHits)

{

StartCoroutine(AutoAttack());

}

}

if (distanceToPlayer > attackRadius)

{

// AI Logic?;

}

if (distanceToPlayer <= chaseRadius)

{

navMeshAgent.destination = player.transform;

}

else

{

// AI Logic to come;

}

}

// Couroutine – the yield is a await like keyword that yields for an amount of time

// before returning to the method and running the following linie.

// Here the animation clip is played and after X sec the gameobject (enemy) is destroyed.

IEnumerator AutoAttack()

{

lastHitTime = Time.time;

animator.play(autoAttack);

yield return new WaitForSeconds(autoAttackAnumationTime);

playerComponent.TakeDamage(damagePerHit);

}

}

## Script: Tower

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Tower{

[SerializeField] float attackRadius = 3f;

[SerializeField] float damagePerShot = 9f;

[SerializeField] float secondsBetweenShots = 1f;

bool isAttacking = false;

GameObject target;

Void Awake()

{

Assert.IsNotNull(target);

}

void Start()

{

}

void Update()

{

// TODO edit to some selection logic.

float distanceToTarget = Vector3.Distance(target.transform.position, transform.position);

if (distanceToTarget <= attackRadius && !isAttacking)

{

isAttacking = true;

// Attack

Debug.Log(“Tower is attacking”);

}

if (distanceToPlayer > attackRadius)

{

// Stop attacking

isAttacking = false;

}

}

Private override IEnumerator Die()

{

DestroyObject(gameObject);

}

void OnDrawGizmos()

{

//Draw attack sphere

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

Gizmos.DrawWireSphere(transform.position, attackRadius);

}

}

## Script: Minion

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Minion, IDie {

{

[RequireComponent(typeof (UnityEngine.AI.NavMeshAgent))]

public class AICharacterControl : MonoBehaviour

{

[SerializeField] float attackRadius = 3f;

[SerializeField] float waitTimeBeforeDeath = 1f;

[SerializeField] float chaseRadius = 5f;

[SerializedField] Transform target; // Ændre til stack med targets nexus/towers //Set til tower til start

NavMeshAgent navMeshAgent;

Animator animator;

bool isAttacking = false;

private void Start()

{

navMeshAgent = FindObjectOfType<NavMeshAgent>();

animator = FindComponentOfType<Animator>();

}

private void Update()

{

//SKAL ÆNDRES TIL logic omkring target selection.

//Start med at gå til target.

// Lav til virtual metode så den kan overskrives.

// Der skal også laves animation bool vedr. running/walking.

(target != null)

{ navMeshAgent.SetDestination(target.position);

}

float distanceToTarget = Vector3.Distance(target.transform.position, transform.position);

if (distanceToTarget <= attackRadius && !isAttacking)

{

isAttacking = true;

Debug.Log(“attacking”)

}

if (distanceToTarget > attackRadius)

{

isAttacking = false;

Debug.Log(“Stop attaking”)

}

if (distanceToTarget <= chaseRadius)

{

navMeshAgent.SetDestination(target.position);

}

else

{

navMeshAgent.SetDestination(transform.position);

}

}

Public void Die ()

{

}

Private virtual IEnumerator Die()

{

animator.play(”Die”);

yield return new WaitForSeconds(waitTimeBeforeDeath);

DestroyObject(gameObject);

}

}

## Script: Stats

using System;

using UnityEngine;

public class Stats : MonoBehaviour, Idamageable

{

[Header("Stats")]

[SerializeField] private float critChance = 0.05f;

[SerializeField] private int level = 1; //Max 5

[SerializeField] private float mana = 317f;

[SerializeField] private float health = 570f;

[SerializeField] private float baseDamage = 58f;

Private float currentHealth;

Private float currentMana;

// Hvor meget skal være eksponeret?

public float CritChance { get { return critChance; } }

// Creates an event that take a float and is being called when health or mana is changed.

Public event Action<float> OnHealthChanged;

Public event Action<float> OnManaChanged;

IDie ObjectToDie;

Private void Awake()

{

IDie ObjectToDie = gameObject.GetComponent<IDie>();

currentHealth = maxHealth;

currentMana = maxMana;

}

Public void TakeDamage(float damageAmount)

{

// Reduced the health with the damage taken.

currentHealth -=damageAmount;

float currentHealthPct = currentHealth/ maxHealth;

// Calls the objects this script is on and call the Die method on the script that implements IDie interface.

if (currentHealthPoints <= 0)

{

ObjectToDie.Die();

float currentHealthPct = 0;

}

// Calls the event saying that the health changed and gives a new percentage

// to whomever is registered for the info.

OnHealthChanged(currentHealthPct);

}

Public void ManaUsed(float manaUsed)

{

// Reduced the mana with the amount used.

currentMana +=manaUsed;

float currentManaPct = currentMana/ maxMana;

// Calls the event saying that the health changed and gives a new percentage

// to whomever is registered for the info.

OnManaChanged(currentManaPct);

}

Public void LevelUp()

{

If (level < 5)

{

Level ++;

baseDamage \*=1.60f

Health \*=1.73f

Mana \*= 1.49f

currentHealth = maxHealth;

currentMana = maxMana;

// Calls the events saying that the health and mana changed and gives a new percentage

OnHealthChanged(currentHealthPct);

OnManaChanged(currentManaPct);

}

}

Public void Respawn()

{

currentHealth = maxHealth;

currentMana = maxMana;

// Calls the events saying that the health and mana changed and gives a new percentage

OnHealthChanged(currentHealthPct);

OnManaChanged(currentManaPct);

}

}

Private void Update()

{

}

## Script: Spawn point

using System;

using UnityEngine;

public class Stats : MonoBehaviour, Idamageable

{

[SerializeField] private gameObject healTarget;

[SerializeField] private float healPrUpdate = -50;

[SerializeField] private float healRange = 5f;

Stats stats;

Awake()

{

Assert.IsNotNull(healTarget);

}

Start()

{

stats = GetComponent<Stats>();

}

Update()

{

float distanceToTarget = Vector3.Distance(healTarget.transform.position, transform.position);

if (distanceToTarget <= healRange)

{

stats.TakeDamage(healPrUpdate);

}

}

}

## Script: Nexus

using System;

using UnityEngine;

using UnityEngine.SceneManager;

public class Nexus : MonoBehaviour, IDie

{

[SerializeField] private string sceneLoad;

Awake()

{

Assert.IsNotNull(sceneLoad);

}

public void Die ()

{

DestroyObject(gameObject);

//SceneManager.LoadScene(sceneLoad);

// Måske bare tekst som I Legend of zafirah.

}

}

## Script: HealthBar

using System;

using System.Collections;

using UnityEngine;

using UnityEngine.UI;

public class HealthBar : MonoBehaviour

{

[SerializeField] private Image foregroundImage;

[SerializeField] Private float updateSpeedSeconds = 0.2f;

private void Awake()

{

Assert.IsNotNull(foregroundImage);

// Subscribe to the event in health script on this elements parent.

// Parent in Unity is the hierarchy – which basicly is the element

// the canvas is placed on

GetComponentInParent<Stats>().OnHealthChanged += HealthChangedHandler;

}

// The method that is called when the event says that health is changed.

Private void HealthChangedHandler(float currentHealthPct)

{

// Starts a Coroutine which makes it possible to have a smooth change

// in the bar as Lerp can be used.

// the fill amount could be instantly.

StartCoroutine(ChangeToPct(float currentHealthPct));

}

// This is the Coroutine that is being called from the handler.

Private IEnumerator ChangeToPct(float currentHealthPct)

{

float preChangePct = foregroundImage.fillAmount;

float elapsed = 0f;

while (elapsed < updateSpeedSeconds)

{

// Adds time paced to elapsed

Elapsed += Time.deltaTime;

// Mathf.Lerp interpolate between the old health and new health based

// on how much of the updatespeed as passed.

foregroundImage.fillAmount = Mathf.Lerp(preChangePct, currentHealthPct, elapsed / updateSpeedSeconds;

yield return null;

}

// Now the while is over as the updatespeed is passed and the fill amount is the new percentage

foregroundImage.fillAmount = currentHealthPct;

}

Private void LateUpdate()

{

// Makes sure that the health is being displayed with direction towards the camera.

Transform.LookAt(camera.main.transform);

Transform.rotate(0, 180, 0);

}

## Script: ManaBar

using System;

using System.Collections;

using UnityEngine;

using UnityEngine.UI;

public class ManaBar : MonoBehaviour

{

[SerializeField] private Image foregroundImage;

[SerializeField] Private float updateSpeedSeconds = 0.2f;

private void Awake()

{

Assert.IsNotNull(foregroundImage);

// Subscribe to the event in Stats script on this elements parent.

// Parent in Unity is the hierarchy – which basicly is the element

// the canvas is placed on

GetComponentInParent<Stats>().OnManaChanged += ManaChangedHandler;

}

// The method that is called when the event says that mana is changed.

Private void ManaChangedHandler(float currentManaPct)

{

// Starts a Coroutine which makes it possible to have a smooth change

// in the bar as Lerp can be used.

// the fill amount could be instantly.

StartCoroutine(ChangeToPct(float currentManaPct));

}

// This is the Coroutine that is being called from the handler.

Private IEnumerator ChangeToPct(float currentManaPct)

{

float preChangePct = foregroundImage.fillAmount;

float elapsed = 0f;

while (elapsed < updateSpeedSeconds)

{

// Adds time paced to elapsed

Elapsed += Time.deltaTime;

// Mathf.Lerp interpolate between the old mana and new mana based

// on how much of the updatespeed as passed.

foregroundImage.fillAmount = Mathf.Lerp(preChangePct, currentManaPct, elapsed / updateSpeedSeconds;

yield return null;

}

// Now the while is over as the updatespeed is passed and the fill amount is the new percentage

foregroundImage.fillAmount = currentManaPct;

}

Private void LateUpdate()

{

// Makes sure that the mana is being displayed with direction towards the camera.

Transform.LookAt(camera.main.transform);

Transform.rotate(0, 180, 0);

}

## Script: Abstract damageCalculation

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

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public abstract class DamageCalculation : MonoBehaviour

{

private float abstract CalculateDamage(float baseDamage, gameObject target, gameObject attacker)

{

// To add a random element I set baseDamage to 94% and the add a random value

// from 1 to 11 giving a range of 95-105%

Random random = new Random();

int randomNumber = random.Next(0, 0.12);

float damage = baseDamage \* 0,94 + baseDamage \* randomNumber/100

// calc if it was a crit

float multiplierOnCrit = MultiplierOnCrit();

// Add potential crit damage multiplier

float damageDone = damage \* multiplierOnCrit;

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

return damageDone;

}

private float MultiplierOnCrit()

{

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

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

{

return 1;

}

else

{

return 2;

}

}

}

## Script: GameManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameManager : MonoBehaviour

{

// Legends of zafirah -> Spawning of minions.

// adding to minions level after x time in game

// Subscribe to AIWinScript (on nexus) and PlayerWin scripts

## Script: PlayerControl

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

namespace Characters

{

public class PlayerControl : MonoBehaviour {

// Use this for initialization

void Start() {

}

// Update is called once per frame

void Update() {

if (Input.GetKeyDown(KeyCode.Alpha1))

{

// Do Special ability 1 and 2 by command design pattern.

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);

}

}

}

# Scripts: SoulForging scripts

## 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 {

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);

}

}

## 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;

}

}

}

}

## 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);

}

}

}

# Script: 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.

}

}

}

## 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);

}

}

}