



Pigmania

Gruppe 16

av

Anders Myrmel, Giar Rehani, Lars-Erik Moi,  
Markus Roland, Oskar Markussen, Shaheen Thayalan

i

IKT201  
Internettjenester

Veiledet av Christian Auby  
Fakultet for teknologi og realfag  
Universitetet i Agder

Grimstad, Desember 2021

### **Sammendrag**

Dette semesteret hadde vi emnet internettjenester. Prosjektet som denne rapporten dekker er den avsluttende oppgaven vi hadde i dette emnet. Prosjektoppgaven var å lage en webapplikasjon over en periode på ca 8 uker. Oppgaven stilte få krav, vi fikk bestemme det meste innholdet i applikasjonen. Oppgaven skulle dekke MVC pattern og bruke Agile-rammeverket Scrum. Universitetslektor Christian Auby er product owner og veilder for prosjektet. Vi bestemte i oppstarts fasen å lage Pigmania. Dette er et spill bygget rundt griser som løper om kapp. Spillet skal gi brukeren mulighet til å vedde på disse løpene og en viktig komponent er grafikken. Ved å bruke Scrum i Jira sørget vi for at hvert gruppe-medlem var oppdatert om prosjektet og visste hvilket arbeid de måtte gjøre. Og ved å bruke Git kordinerte vi all koden.

## Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	<b>Vi erklærer videre at denne besvarelsen:</b> <ul style="list-style-type: none"><li>• Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands.</li><li>• Ikke refererer til andres arbeid uten at det er oppgitt.</li><li>• Ikke refererer til eget tidligere arbeid uten at det er oppgitt.</li><li>• Har alle referansene oppgitt i litteraturlisten.</li><li>• Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.</li></ul>	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høyskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

## Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).

Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

## Forord

Denne rapporten dekker utviklingsprosessen av eksamensoppgaven vår i faget IKT201-G Internettjenester. Oppgaven gikk ut på å utvikle en større webapplikasjon med bruk av C# som programmeringsspråk for applikasjonens backend. Vi har jobbet med prosjektet i et team bestående av seks medlemmer, og har dermed fått en førstehåndssopplevelse på hvordan det er å jobbe som utviklere i et SCRUM-team. Sammen har vi utviklet et nettleserbasert gambling spill kalt Pigmania.

Vi vil gjerne utnytte denne muligheten til å takke vår lærer og veileder Christian Auby, universitetslektor her ved Universitet i Agder. Han har lært oss mye i løpet av vår tid på universitetet, og vært til stor hjelp gjennom prosjektperioden.

Grimstad

22. Desember 2021

Anders Myrmel, Giar Rehani, Lars-Erik Moi, Markus Roland, Oskar Markussen, Shaheen Thayalan

# Innhold

<b>1</b>	<b>Innledning</b>	<b>1</b>
1.1	Bakgrunn . . . . .	1
1.2	Forutsetninger . . . . .	1
1.3	Begrensninger . . . . .	2
1.4	Produktvisjon . . . . .	2
1.4.1	Features . . . . .	2
1.4.2	Målgruppe . . . . .	3
1.4.3	Tidsramme og budsjett . . . . .	3
1.4.4	Lignende produkter . . . . .	3
<b>2</b>	<b>Prosess</b>	<b>4</b>
2.1	Planlegging . . . . .	4
2.1.1	Sprint . . . . .	4
2.1.2	Veiledning . . . . .	4
2.1.3	Arkitektur . . . . .	4
2.1.4	User stories . . . . .	5
2.2	Arbeidsfordeling . . . . .	6
2.3	Versjonskontroll . . . . .	6
<b>3</b>	<b>Teknisk bakgrunn / teoretisk bakgrunn</b>	<b>8</b>
3.1	ASP.NET Core, C sharp, backend . . . . .	8
3.2	Design pattern . . . . .	8
3.3	JIRA . . . . .	9
3.4	Bitbucket (Git) . . . . .	9
3.5	HTML, CSS, Javascript og front-end . . . . .	9
3.6	Bootstrap . . . . .	10
3.7	Vue.js . . . . .	10
3.8	Rider . . . . .	10
3.9	C sharp . . . . .	10
3.10	Personvern . . . . .	10
<b>4</b>	<b>Produkt</b>	<b>12</b>
4.1	Hovedmeny . . . . .	12
4.2	Truffle Clicker . . . . .	12
4.3	Leaderboard . . . . .	13
4.4	Racing . . . . .	14
4.5	Grise konfigurasjon . . . . .	15

<b>5</b>	<b>Implementasjon</b>	<b>16</b>
5.1	Overordnet arkitektur . . . . .	16
5.1.1	Back-end struktur . . . . .	16
5.1.2	Front-end struktur . . . . .	17
5.2	Hjemmesiden . . . . .	17
5.3	Racing algoritmen . . . . .	17
5.4	Racing grafikk . . . . .	19
5.5	Play side . . . . .	20
5.6	MoneyTransaction side . . . . .	21
5.7	PigOperations . . . . .	21
5.8	Truffle Clicker . . . . .	22
5.9	Leaderboard . . . . .	24
<b>6</b>	<b>Testing og validering</b>	<b>25</b>
6.1	Testing av User stories . . . . .	25
<b>7</b>	<b>Diskusjon</b>	<b>28</b>
7.1	Hva kunne blitt gjort annerledes? . . . . .	28
7.2	Resultater i forhold til krav . . . . .	28
7.3	Prosess . . . . .	29
7.4	Produkt . . . . .	29
7.5	Implementasjon . . . . .	30
7.6	Utfordringer . . . . .	31
7.6.1	Kombinere grafikk og race algoritme . . . . .	31
7.6.2	Implementere Pig model . . . . .	32
7.6.3	Vue.js Singel-Page-application . . . . .	32
7.6.4	Vue.js som et live løp av racing algoritmen . . . . .	33
7.6.5	Betting . . . . .	33
7.7	Videre arbeid . . . . .	33
7.7.1	Forbedre grafikk . . . . .	33
7.7.2	Egen gris . . . . .	34
7.7.3	Case opening . . . . .	34
7.7.4	Rarity basert racing . . . . .	34
7.7.5	Multiplayer . . . . .	34
7.7.6	Coin table (backend) . . . . .	35
7.7.7	Truffle clicker . . . . .	35
7.7.8	Pig breeding . . . . .	35
<b>8</b>	<b>Konklusjon</b>	<b>36</b>
<b>9</b>	<b>Gruppekontrakt</b>	<b>37</b>

## Figurer

1	Animert løp . . . . .	14
2	Dette er hvordan siden ser ut . . . . .	15
3	Oppsett med mapper i prosjektet . . . . .	16
4	Fart blir oppdatert av normal random generation . . . . .	18
5	Scenen blir tegnet på læreret . . . . .	19
6	Pseudokode for oppdatering av grisenes posisjon . . . . .	19
7	En variable bestemmer antall griser . . . . .	20
8	Siden til operasjonene som du kan gjøre på en gris innlogga som Admin	21
9	Siden til Edit . . . . .	22
10	Dette er hvordan vi implementerte admin autentiseringen for pig configuring i Rider . . . . .	22
11	Truffle clicker siden. Her ser vi oppsettet vi har brukt. Mynten genererer Truffle Coins, trøfflen er oppgraderingsknappen . . . . .	23
12	Formelen til oppgraderingsprisen (linje 4-5). For å unngå desimaltall, konverterer vi prisen fra å være en double til å bli en integer. . . . .	23
13	Eksempel på funksjon i kontroller. Figuren viser funksjonen til oppgraderingsknappen . . . . .	24
14	Flytte informasjon mellom script i backend . . . . .	31
15	HTTP gir 404, ingen grise modeller dukker opp . . . . .	32

## Tabeller

1	User-story Registrere Bruker . . . . .	25
2	User-story Innlogging av bruker . . . . .	25
3	User-story Innlogging av bruker . . . . .	25
4	User-story Vedde på en Gris . . . . .	26
5	User-story Skaffe penger uten å gamble på et løp . . . . .	26
6	User-story Bruker kan se ledertavle . . . . .	26
7	User-story Bruker kan se løpet . . . . .	26
8	User-story Produkt vs Krav . . . . .	29

## Listinger

# 1 Innledning

Innledningen introduserer prosjektet fra det ståstedet vi startet med. Først skal vi forklare hvilke for kunnskaper gruppen hadde. Kunnskaper vi har fra studie og tidligere arbeid. Videre skal vi se hvilke forutsetninger vi hadde for prosjektet. Altså hva vil trolig å gjøre prosjektet vanskelig og hva skal gjøre prosjektet overkommelig. Til slutt skal vi se våres Produktvisjon, planene vi lagde for produktet i oppstarts fasen. Produktvisjonen forklarer hva produkter skal være og forutsetninger vi hadde. I tillegg viser den hvor inspirasjonen våres kom fra.

## 1.1 Bakgrunn

Teamet vårt består av seks dataingeniør studenter. Dette semesteret har vi emnet Internettjenester, hvor den avsluttende oppgaven er et større prosjekt. Prosjektet er å utvikle en webapplikasjon, ellers stiller oppgaven få krav. Vi valgte å lage et spill. Utover det vi har lært i faget hadde gruppen lite erfaring med webutvikling. Vi hadde heller ikke jobbet med spillutvikling eller web design tidligere. I tillegg var verktøy for team koordinering også nytt for oss.

Derimot hadde vi erfaring med å jobbe med database, frontend, backend og kontrollere i eksamens prosjektet i ikt103 forrige semester 2021. Dette gjorde oss mer egnet til å ta på oss et slikt delt opp prosjekt som hadde flere deler enn bare det grafiske eller koding. En i gruppen vår hadde også valgt grafikk som et ekstra valgfag dette året. Dette utgjorde en stor del av den gode utførelse av den 3D grafiske delen av prosjektet.

Grunnen til at tema-et for webapplikasjon ble et grise spill, kommer med det at nesten alle i gruppen har en lindenskap/kjennskap til spill-verdenen og dens appellerende deler. Gambling aspektet av spillet kommer fra at enkelte individer har en god erfaring med slike applikasjoner i fortiden.

## 1.2 Forutsetninger

Vi bestemte oss relativt tidlig i semesteret for at vi skulle lage en mobilapplikasjon som først skulle være basert på React. Fire av de originale medlemmene var på Bekk sin React-Workshop for å få litt erfaring i React før vi begynte på prosjektet. Da prosjektet begynte å nærme seg trakk 2 av medlemmene våre seg fra gruppen slik at vi var 4 stk, men nærmere oppstart så fikk vi 2 nye medlemmer. Vi bestemte oss senere for at vi ikke skulle ha mobilapplikasjon eller bruke React.

Våre forutsetninger for prosessen til dette prosjektet var at samarbeid kanskje ville være vanskelig i starten. Vi tenkte dette fordi at i tidligere prosjekter har det vært vanskelig å sammenslå de ulike delene av arbeidet. Vi var usikre på Git først



og det var en av de vanskelige samarbeidstilpasningene vi tenkte kommer til å pågå. Utifra egne erfaringer hadde vi tanken om at noen av de ulike aspektene i prosjektet ville bli et hinder. Slik som planlegningen av hvordan systemet skulle være.

### 1.3 Begrensninger

Noen av de tekniske begrensningene som kom med prosjektet var tiden det ville ta å lære om de nye og mer kompliserte delene av systemet vårt. Det var flere ganger under prosjektet hvor problemene våre lå med lite kunnskap eller erfaring av de ulike emnene som Vue, “session storage”, “forms” og “modules”. Dette kommer tilbake til gruppens begrensede erfaring med Webapplikasjoner og programmeringsspråkene brukt på nettsider.

Noen av de omstendighetsbegrensningene som vi hadde var at vi hadde alle et ulikt syn på hvordan produktet skulle være/oppføre seg. Det gjorde ting vanskelig under utviklingsfasen fordi vi ikke ble enige om hva vi trengte å gjøre. Dette var også vanskelig fordi vi hadde andre fag som måtte bli gjort. Vi fikk aldri et fast bestemt og samme syn på et ferdigprodukt. Dette utviklet prosjektet vårt til å leve på drømmer om hva det kunne bli, og dette viser til viktigheten ved planlegning. Vi skulle også spesifisert konsekvensene av manglende arbeidstimer i gruppekontrakten.

### 1.4 Produktvisjon

Pigmania skal tilby et engasjerende og uformelt spill rettet mot unge voksne. Produktet har som hensikt å tilfredsstille kundens behov for en avhengighetsskapende spillopplevelse uten den potensielle konsekvensen av økonomisk ustabilitet. For å oppnå dette skal Pigmania være et enkelt spill å komme i gang med, samtidig som det skal engasjere brukeren med funksjoner som karaktersamling og gambling. Spillet skal skille seg fra konkurrentene ved å være gratis å spille. Brukeren skal ha mulighet til å registrere seg og begynne å nyte spillet umiddelbart.

#### 1.4.1 Features

Produktet vårt skal være et spill der man vedder på hesteløp, men hestene er griser. Derfor er kode navnet til prosjektet Pigmania. Navnet er basert på bilspillet “Trackmania”, som er et bilspill der man konkurrerer om å kjøre raskest på hvert level. Grisene vil ha unike aspekter og konkurrere over flere løp. På denne måten vil spillere ha informasjon om løp uten direkte innflytelse. Dette er hovedfokuset til prosjektet. Utover det har vi flere funksjoner vi ønsker å implementere dersom vi får tid.

Vi planlegger å først gjøre spillet operativt kun lokalt. For deretter å muligens skape funksjoner for at flere spillere kan samhandle med hverandre. Dette vil i første omgang gå ut på at spillerne vedder på de samme løpene. Her vil vi vurdere å implementere “Decimal odds” eller et lignende system.

Løpene kan holdes regelmessig på gitte tidspunkt. Spillere vil kunne vedde på og se informasjon om kommende løp. Dette vil gjøre løpene mer unike. I tillegg vil det åpne muligheter for å gjøre grisene mer unike ved at de endrer seg over tid.

I starten av planleggingen bygget konseptet på at brukere eier egne griser. Disse grisene kan tilpasses med trening og gjenstander. Etterhvert som prosjektet endrer seg er det usikkert hvordan disse funksjonene passer inn. Vi vil se på dette når vi nærmer oss mål med viktigere funksjoner.

#### **1.4.2 Målgruppe**

Målgruppen til Pigmania er ungdom og unge voksne. PC spill er mest populært blant denne demografien. Vi tror det viktigste salgspunktet til spillet er gambling aspektet. Pengespill er mest populært blant unge menn. Spillet skal være lett å komme igang med og derfor appellere til “casual” brukere.

#### **1.4.3 Tidsramme og budsjett**

Frist for prosjektet er 22. desember. Prosjektet må være operativt til denne datoen. For å oppnå dette har vi bestemt at hvert gruppemedlem skal bidra med 108 timer. Ingen i gruppen har god erfaring med å estimere tidsbrukt, derfor kan det vise seg at vi har estimert for lite tid. Om dette blir resultatet forventes det at alle gruppe medlemmer tar ansvar for å gjøre ferdig prosjektet.

#### **1.4.4 Lignende produkter**

Konsept ideen var inspirert av gambling- og racingaspektet ved “Zed.run”. “Zed.run” er et veddeløpsspill hvor eiere av virtuelle hester deltar i løp.

I tillegg finnes det utallige arkader og casinoer med sine egne hesteløp simulatorer, samt digitale tjenester hvor brukere kan gamble virtuell valuta på simulerte utfall. Populære eksempler på dette er “SaltyBet” og “Twitch plays Pokemon”. Et fellestrekk blant tittlene nevnt er at utfallet ikke er helt tilfeldig. Hvilke karakterer som er med i en match påvirker oddsene. Derfor vil erfarne spillere ha en liten fordel som kan uttrykke seg over mange matcher. Tilknyttet dette har disse spillene mekanikker som gjør de virtuelle karakterene unike. Dette gjør de på tre måter. 1. Ved å utnytte karakterer som målgruppen allerede er kjent med. 2. Karakterene har konsistene mekanikker og egenskaper. 3. Spillere kan bygge sine egne karakterer.

## 2 Prosess

Vi innså tidlig at en ryddig og oversiktlig prosess var avgjørende for prosjektet. Ukentlige møter fra dag en, samt programvare som Jira og Git bidro til å opprettholde en klar struktur i prosjektarbeidet. På denne måten ble det enklere å delegere nytt arbeid, samt orientere gruppe medlemmene om fullførte oppgaver. Kombinert med god kommunikasjon førte dette til et effektivt arbeidsmiljø og smidig prosjektstyring.

### 2.1 Planlegging

Planlegging har foregått på flere forskjellige måter. Vi har allerede sett på planleggingen vi gjorde i oppstartsfasen og vil derfor ikke ta det opp her. Etter oppstartsfasen har vi gjennomført jevnlige møter og veiledninger. På møtene har vi gjennomgått sprinter. Veiledningene våres ble holdt av Christian Auby. Utenom møtene har vi brukt Git for å samle koden våres.

#### 2.1.1 Sprint

Gruppen har møte angående sprint hver Mandag 13:00. Her går vi først gjennom “sprint retrospective”, “sprint review” og til slutt planlegger vi neste ukes sprint. I “sprint retrospective” diskuteres hva som fungerte bra, og hva som kan forbedres. Under “sprint review” ser vi på progresjonen vi har hatt i løpet av siste ukes sprint. Når dette er fullført avsluttes den ferdige sprinten og en ny sprint opprettes. “User stories” som ikke er fullført flyttes til den nye sprinten. Deretter begynner planleggingen av neste ukes sprint. Vi diskuterer sammen om hva vi ønsker å få gjort framover i prosjektet og oppretter “user stories”. Vi tildeler så story points til disse, gjerne ved å spille “Scrum Poker”. Til slutt bestemmer vi hvilke nye “user stories” som skal være med på sprinten.

#### 2.1.2 Veiledning

Gruppen har fått tildelt veiledningstime med “Scrum master” Universitetslektor Christian Auby på Tirsdager i par tallsuker. Før disse møtene samles gruppen for å planlegge hva som kan være lurt å ta opp. Ting som kommer opp på disse møtene har vært

#### 2.1.3 Arkitektur

I prosjektet vårt har vi brukt MVC mønsteret til å forme en applikasjon som er delt i tre logiske komponenter:

- Model

- View
- Controller

Våre modeller er klasser og objekter av brukere og griser i databasen. Modellene holder data som blir brukt av kontroller og sendt til Viewene.

Viewene viser brukeren alle nettstedene våre og innhold som er laget gjennom HTML, CSS og JavaScript.

Tilslutt har vi kontrollerene våre som styrer hvilke modeller eller views som en bruker kan ha tilgang til. Kontrollere er også hjernen bak hele nettsiden. De blir brukt til å flytte data fra modeller til viewet. Dersom viewet skal gjøre noe mot serveren, vil den først gå gjennom kontrolleren som deretter bestemmer hvor data skal bli lagret eller eventuelt hvilke data som skal bli hentet og vist tilbake til viewet.

#### 2.1.4 User stories

Gjennom prosjektet har vi hatt flere ideer om hva som skulle ha blitt implementert og hva som brukeren burde ha mulighet til å gjøre. Noen av disse ideene var urealistiske, mens andre var oppnåelige og en kjerne del av prosjektet. De viktigste mulighetene en bruker skal ha har vi skrevet ned som User stories. Her er noen av disse:

Som en bruker,  
Har jeg lyst til å registrere meg,  
for å kunne logge meg inn.

Som en bruker,  
Har jeg lyst til å logge meg inn,  
for å kunne identifisere meg selv.

Som en bruker,  
Har jeg lyst til å kunne velge hvor mye jeg skal gamble,  
for å kunne bestemme mengden på mine egne gevinster og tap.

Som en bruker,  
Har jeg lyst til å kunne velge hvem jeg skal gamble på,  
for å styre mitt eget løp.

Som en bruker,

Har jeg lyst til å skaffe penger andre steder en et løp,  
for å kunne skaffe penger når jeg er tom.

Som en bruker,  
Har jeg lyst til å kunne se hvilke rangering jeg har,  
for å vite hvor god jeg er i forhold til andre.

Som en bruker,  
Har jeg lyst til å ha muligheten til å se et løp når jeg gambler,  
for å føle mer spenning en å få resultat med en gang.

## 2.2 Arbeidsfordeling

Gruppen foredler arbeidet ut ifra hva folk ønsker å gjøre eller kan gjøre. Vi har scrum-møter hver mandag for å sette sammen en sprint og issues som vi kan jobbe med i løpet av uken. Vi snakker om hvem som er mer kompatibel for hver del slik at vi er effektive på de ulike delene. Siden vi har bra kommunikasjon mellom gruppemedlemene er det lett å vite hva som kan bli bygd på og hva som må bli slått sammen. Mye av arbeidet er fordelt rettferdig og dersom man ikke har noe å gjøre så spør man en i gruppen slik at man får noe å gjøre. Hvis ingen vet så er det bare å jobbe på rapporten.

## 2.3 Versjonskontroll

I løpet av prosjektet utnyttet vi versjonskontrollsystemet Git. Versjonskontrollsystemet la til rette for en jevn og kontinuerlig flyt av endringer i programvarekoden. Vi opprettet et repository for prosjektet på UIA sin bitbucket server der vi hadde oversikt over alle filene og mappene. Dette gjorde det enklere å spore og administrere endringer i programvarekoden.

I prosjektet hadde vi en hovedgren som skulle være funksjonell til en hver tid, i tillegg til egne grener dedikert til de forskjellige oppgavene i prosjektet. Etterhvert som vi la til funksjonalitet på grenene "commitet" og "pushet" vi for å lagre endringene til repository. Dette gjorde det enkelt for de andre medlemmene å "pulle" endringene og ta opp arbeidet der i fra.

Istedenfor å kontinuerlig "merge" med hovedgrenen etter hver implementasjon lagde vi ofte undergrener på undergrenene for så å "merge" disse sammen. Vi valgte å gjøre det på denne måten for å beskytte hovedgrenen. Dette gjorde det enklere for flere å arbeide på urelaterte oppgaver samtidig, og dersom en endring viste seg å

være inkompatibel med hovedgrenen fikk vi muligheten til å rydde opp i dette uten å blokkere arbeidet til resten av gruppen.

### 3 Teknisk bakgrunn / teoretisk bakgrunn

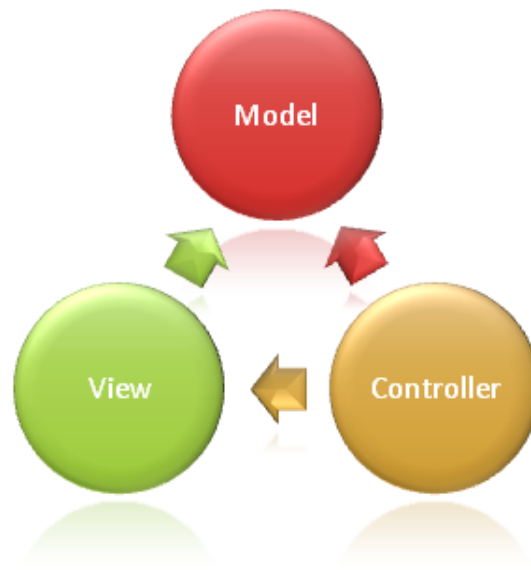
Prosjektoppgaven vår gikk ut på å utvikle en dynamisk nettside kombinert med server-sideprogrammering for å tilby funksjonalitet som database tilkobling og generering av resultat til nettlelere. Webapplikasjons-rammeverket ASP.NET la grunnmuren for server-siden, mens teknologiene HTML, CSS, Javascript, og spesifikt Vue.js utgjorde kjernen av klient-siden. I dette kapitlet skal vi beskrive verktøyene vi har brukt.

#### 3.1 ASP.NET Core, C sharp, backend

ASP.NET er et populært web-utvikling rammeverk som ofte blir brukt for å bygge web-applikasjoner. Vi har tatt for rammeverket .NET Core. Ved bruk av dette rammeverket og C sharp har vi bygget opp vår back-end i prosjektet. All koden vi har på back-end vil bli kjørt av serveren før innholdet blir sendt til klienten, altså nettleseren til bruker.

#### 3.2 Design pattern

Gjennom ASP.NET og utviklingsverktøyet Rider laget av JetBrains bruker vi et design pattern som heter MVC. MVC står for model, view og controller. Med et slikt design mønster kan vi gjøre slik at bruker-requester blir sendt til en controller som jobber sammen med en model som utfører brukerhandlinger eller å sende/motta resultater av queries. Kontrolleren velger viewet som skal vises til bruker med data fra model. Her er en modell på hvordan MVC pattern fungerer.



[1]

### 3.3 JIRA

JIRA er et verktøy som gjør at det er enkelt og effektivt å jobbe sammen som en gruppe. I JIRA legger man til “issues” eller oppgaver som kan ha en fastsatt prioritet. Disse oppgavene fordeles innad i gruppen og man velger selv hvilke oppgaver man skal holde på med. JIRA hjelper å holde styr på hvilke oppgaver som blir gjort og lar oss også å logge timer på oppgavene, slik at gruppa kan se hvor mye gruppemedlemmer jobber i prosjektet.

### 3.4 Bitbucket (Git)

UiA har sin egen Bitbucket-server. Bitbucket er en platform som gjør det enkelt for flere utviklere å finne en tilsvarende kode samtidig. Kort sagt fungerer dette ved hjelp av grener eller “branches” som lar utviklere jobbe med en separat kopi av koden, slik at andres endringer ikke forstyrrer hovedkoden (main), mens du får sjansen til å slå sammen endringene når du er ferdig med et kodebidrag.

### 3.5 HTML, CSS, Javascript og front-end

HTML, eller Hyper Text Markup Language er en terminologi innen web-utvikling og er språket som brukes på alle nettsteder for å vise innhold og beskrive strukturen til en internettside. Sammen med CSS for design og enkle animasjoner/overganger og javascript for interaktive nettsideelementer, bygger disse tre opp den komplette front-end av nettstedet vårt konstruert som offer for disse tre språkene. Front-end



vil si all koden som blir servert til brukeren i nettleseren. Back-end i motsetning er all kode som blir kjørt på serveren.

### 3.6 Bootstrap

Bootstrap er et CSS- og javascript-rammeverk for å lage responsive og mobilvennlige nettsider. Bootstrap består av layoutmaler for knapper, HTML-skjemaer, navigasjon, samtalebokser og mye mer. For det meste brukes Bootstrap ved å knytte CSS-klasser til html-tagger for å videre style elementene, men det er også muligheter for å opprette funksjoner men dette krever javascript for å fungere.

### 3.7 Vue.js

Vue er et progressivt javascript-rammeverk for å bygge brukergrensesnitt. Vue er ikke som andre uniforme rammeverk, Vue er ment fra bunnen og opp for å være trinnvis adopterbar. Vue er relativt enkelt å bruke og integrere i eksistene prosjekter og på den motsatte siden er Vue i tillegg i stand til å drive kraftige enkeltsideapplikasjoner når de er brukt i kombinasjon med moderne verktøy og støttebiblioteker.

### 3.8 Rider

Rider er en IDE (Integrated Development Environment) som er laget av JetBrains. Rider gir oss viktige programmeringsverktøy som koderedigering, kode generering, debugging, osv. I tillegg gir Rider oss maler for forskjellige prosjekttyper.

### 3.9 C sharp

C sharp er et objektorientert programmeringsspråk som er utviklet av Microsoft. C sharp baserer seg på C++ og Java, og ble utviklet for å ha en balanse mellom styrke og rask utvikling. Når man utvikler med .NET Core rammeverket er det effektivt å bruke et programmeringsspråk som er godt støttet av Microsoft og egnet mot .NET rammeverket.

### 3.10 Personvern

EU har personvernsregelverk som heter General Data Protection Regulation også kalt for GDPR. Dette regelverket ble satt i gang i 2018 og er en viktig del av personvern. Gjennom ASP.NET og Rider har vi brukt en mal for brukeroppsett som følger alle kravene som er i regelverket. Serveren tar vare på:

- Navn
- E-post

- Telefonnummer
- Passordhash
- truffleCoins
- truffleClickPower

Brukeren har også tilgang til å slette sin egen bruker og laste ned data om brukeren dersom en ønsker det.

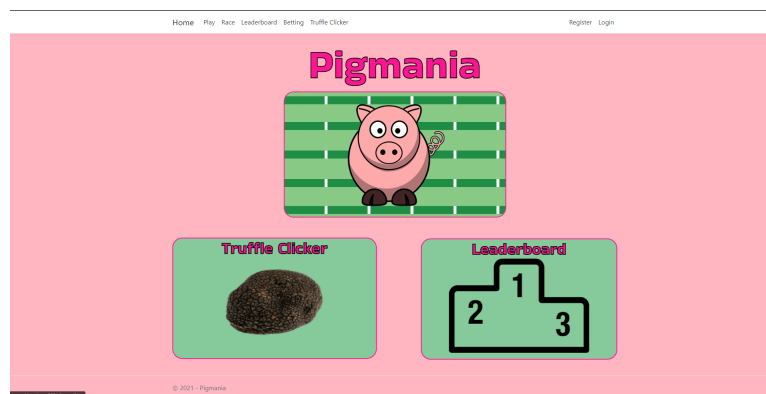
## 4 Produkt

“Pigmania” er et gambling spill hvor man gamlber på grise race. Spillet henter mye inspirasjon fra “Zed.run” og “Cockie Clicker”. Spillere kan tjene Truffle Coins trygt, men sakte ved å spille “Truffle Clicker”. Pengene de tjener her kan gambles for å tjene stort i Pigmania. Pigmania er lynraske race hvor virtuelle griser løper om kapp.

I dette kapitlet skal vi gå gjennom de viktigste funksjonene til “Pigmania”. Funksjonaliteten er delt på 5 hoved sider: Loginn/registerer, Play, TruffleClicker, Leaderboard og konfigurasjon.

### 4.1 Hovedmeny

I hovedmenyen viser vi startsidene til spillet hvor alle kan se oppsettet og hva spillet går ut på. Her så kan både brukere med konto og uten konto se ideen til spillet og hvordan det er.



På toppen av hjemmesiden har vi forskjellige sider som viser hva nettsiden vår går ut på. For å kunne være med på spille må man registrere en bruker for å få tilgang til visse funksjonaliteter. Alle kan se leaderboard, men bare brukere kan spille, vedde eller TruffleClicke.

### 4.2 Truffle Clicker

På denne siden kan registrerte brukere bruke truffle clickeren. Truffle Clicker er et minispill som vi har lagt til der brukeren kan trykke på en knapp som gir brukeren en truffle coin. Truffle clickeren har også en oppgraderingsknapp som brukeren kan kjøpe for å øke mengden truffle coins man får på hvert knappetrykk. Denne siden er ment som en alternativ måte å tjene penger på om man ønsker en liten pause fra gamblingen eller mangler noen få truffle coins for å kunne vedde på et nytt løp.

### 4.3 Leaderboard

Pigmania har en side dedikert til et leaderboard. Hensikten er å rangere alle spillere i en tabell, der hver enkelt rad viser spillerens rank, brukernavn, antall trufflecoins de besitter, og win rate. Spillerene rangeres i synkende order basert på antall trufflecoins de innehar. Et leaderboard gir brukeren et mål annet enn å vinne coins, ved at de kan konkurrere med andre spillere om rangeringer.

Pigmania Play LeaderBoard Truffle Clicker Register Login

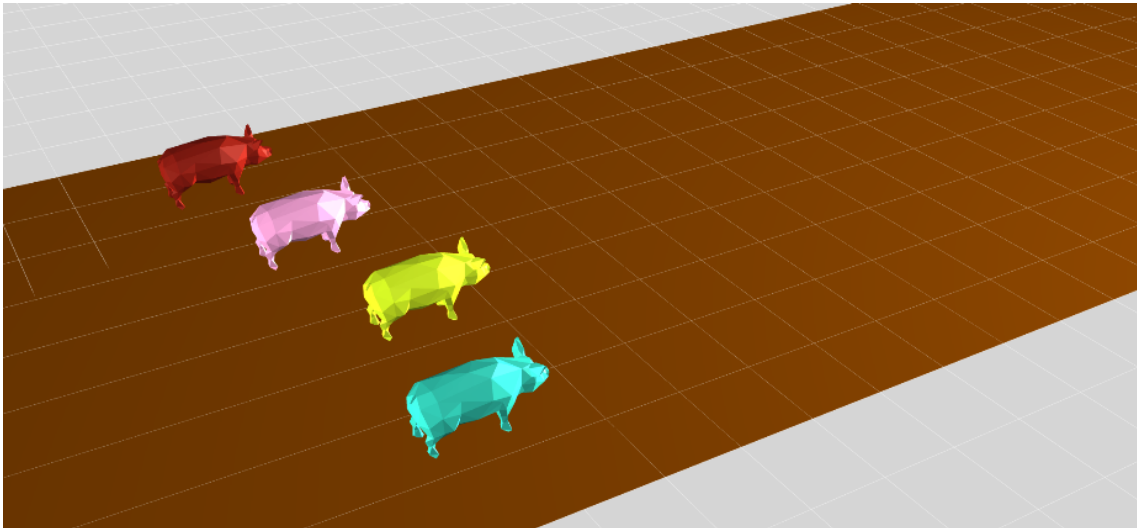
Leaderboard			
Check out the top players			
Rank	Username	Coins	Win rate
1	Shaheen	400	TBD
2	Anders	350	TBD
3	Giar	300	TBD
4	Lars	250	TBD
5	Markus	200	TBD
6	Oskar	150	TBD
7	PigMaster	100	TBD

#### 4.4 Racing

Fra Play siden velger brukeren hvor mye de ønsker å vedde. Det er mulighet for å vedde opp til 10 000 Truffle Coins. Det er kun en vinnergris i et res og premien er fire ganger veddemålet.

Etter å ha valgt hvor mye de skal vedde må spilleren velge grisen de vil vedde på. En ny skjerm viser navnet på fire tilfeldig valgte griser. At bare navnet vises gjør at det er lite informasjon å basere seg på. Den eneste måten spilleren kan øke oddsene sine på er å studere løp. Brukeren kan tilnære seg styrken til griser ved å se hvordan de opptrer over flere løp.

Når brukeren har valgt en gris starter løpet. Løpet vises grafisk i en 3D verden. Her følger kamera 4 veldig raske og svært fargerike griser. Fra denne skjermen kan brukeren når som helt åpne resultat siden. På denne siden vises plasseringen til grisene.



**Figur 1:** Animert løp

## 4.5 Grise konfigurasjon

Hvis du er administrator på nettsiden så kan du legge til griser som kan delta i løpet, du styrer selv hvor rask grisen skal være. Admin kan også endre eksisterende griser og slette griser. I tillegg ser de en liste over alle grisene.

### Index of Pigs

[Create New](#)

Name	Speed	
Oskar	25	<a href="#">Edit</a>   <a href="#">Delete</a>
Lars	14	<a href="#">Edit</a>   <a href="#">Delete</a>
Markus	13	<a href="#">Edit</a>   <a href="#">Delete</a>
Giar	14	<a href="#">Edit</a>   <a href="#">Delete</a>
Shaheen	15	<a href="#">Edit</a>   <a href="#">Delete</a>
Anders	10	<a href="#">Edit</a>   <a href="#">Delete</a>

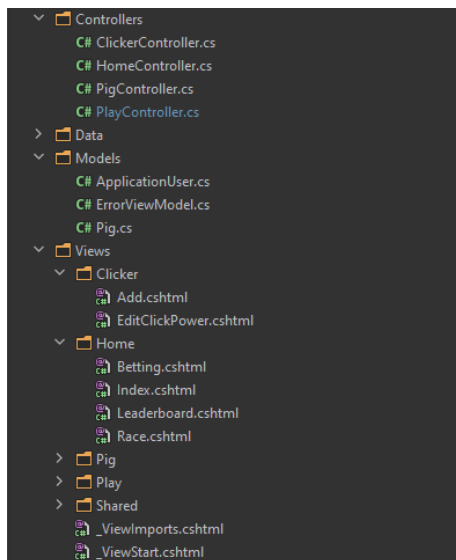
**Figur 2:** Dette er hvordan siden ser ut

## 5 Implementasjon

Pigmania består av en database, kontrollere, views og algoritmer. Databasen inneholder informasjon om brukere, virtuell valuta og griser. Views er hjemmeside, spill side, leaderboard, “TruffleCliket” og “PigOperations”. Vi har laget algoritmer for kalkulere race, oppdatere saldo og presentere race grafisk. I dette kapitlet beskriver vi hvordan disse delene er laget.

### 5.1 Overordnet arkitektur

Vi har strukturert hele applikasjonen vår utifra MVC modellen. Applikasjonen har en back-end og en front-end, hvor back-end håndterer data gjennom kontrollere og modellene, mens front-end er kun brukergrensesnittene, eller Viewene.



Figur 3: Oppsett med mapper i prosjektet

#### 5.1.1 Back-end struktur

Vi har 4 hovedkontrollere som kontrollerer alt i systemet vårt.

- ClickerController
- HomeController
- PigController
- PlayController

Clicker kontrolleren vår håndterer hvilke brukere som har tilgang til clicker viewene våre. Den har også hovedfunksjonene som å generere en trufflecoin til en bruker, og ta vekk trufflecoins når en bruker kjøper en oppgradering.

Home kontrolleren håndterer hvilke data som skal bli sendt til viewene på hjemmesiden.

Pig Kontroller håndterer redigering, sletting og lagring av griser. Kontrolleren krever at brukeren er en autorisert admin.

Play Kontrolleren håndterer gevinster eller tap i løp. Den passer på om bruker er logget inn før den viser viewene som er sendt fra denne kontrolleren.

### 5.1.2 Front-end struktur

Vi har strukturert all front-end koden vår inn i en egen View mappe. Vi har undermapper som tilhører hver kontroller, som hjelper å holde styr på hvilke kontrollere som har tilgang til hvilke views. Viewene har tilgang til å sende data imellom kontrolleren for å endre på data i modellene.

Innholdet i viewene er laget med CSHTML, CSS og JavaScript. Vi bruker JavaScript og CSS til å kunne kommunisere fra et view til kontroller til modell.

## 5.2 Hjemmesiden

Hjemmesiden til Pigmania er veldig enkel og viser frem til funksjonalitetene vi har til prosjektet vårt. Hjemmesiden består av tittelen til prosjektet og tre knapper som brukeren kan trykke på. De tre knappene er:

- Play
- Truffle Clicker
- Leaderboard

Vi har valgt å ha en lyserosa bakgrunn og lilla tekst for å fremheve grise-temaet til nettsiden. Knappene består av tekst og bilde som skal vise til funksjonaliteten til hver enkelt knapp. For å gjøre det litt penere, har vi implementert en hover effect i CSS for knappene. Dersom du holder over en av knappene, vil det skje en animasjon som øker størrelsen på knappen.

## 5.3 Racing algoritmen

Utviklingen av racing algoritmen foregikk i 4 ulike deler:

- Implementering av algoritmen
- Integrering av database og algoritmen



- Integrering av grafikk og algoritmen
- Integrering av betting og algoritmen

Algoritmen tar inn informasjon om de konkurrerende grisene og innstillinger for løpet. Informasjonen om grisene er i form av en dictionary som holder grisenes navn og hastighet. Innstillingene er et objekt som velger lengden og tempoet til løpet i tillegg til hvor mange griser som skal være med.

Før integrering av database og grafikk ble informasjonen om grisne hentet i JavaScript filen. Men nå blir all data sent inn som parametere. Algoritme som går igjennom løpet returnerer en dictionary. Denne inneholder grisenes navn, koordinater og avsluttende plassering. Informasjon blir hentet ut fra dictionary objektet i 3 deler, først hvilke dictionary, så kommer det to variabler som skal vise til hvor informasjonen enn vil ha ligger. Et eksempel på hvordan data ble hentet ut er `dict[name][2]`. Her hentes navnet til gris nummer to. Dette ble brukt til og beskrive grisene som løp på banen utenfor racing algoritmen.

Algoritmen er basert på at grisene har en posisjon, distanse og fart. Farten blir for hvert steg lagt til distansen. For hvert steg blir en variabel lagt til hver fart. Denne variabelene er valgt av en random number generator. Denne velger et tilfeldig tall i et intervall og valget er vektet mot midten av intervallet. En gris blir beskrevet som en klasse. Denne klassen har fem attributter. Disse er navn, fart, posisjon, avsluttende plassering og finish. Navn og fart får den fra parameterene sendt inn, posisjon starter alltid på null, plassering blir gitt etter at grisen har fullført løpet. Finish er en boolean som brukes i algoritmen for å vite når en gris er over målstreken.

```
for (let i = 0; i < settings.n_pigs; i++) { // Kodens kjøring en gang for hver gris.
  let rand = normal_random(0, 1, 0.5) // Velger et tilfeldig tall mellom 0 og 1, vektet mot 0.5.
  if (pigs[i].speed + rand > 1) // Grisenes farten går aldri under 1.
    pigs[i].speed = pigs[i].speed + rand;
}
```

**Figur 4:** Fart blir oppdatert av normal random generation

Det ble lagt til en model for griser med 2 attributter; navn og fart. Det blir hentet ut fire tilfeldige griser fra databasen ved hjelp av en funksjon. Vue i frontend håndterer kallet på disse funksjonene, algoritmen og kjøringen av databasen.

Kombinering av grafikk og algoritmen var en vanskelig prosess. Det ble gjort i backend og blir dypere forklart i 7.6.1 Kombinere grafikk og race algoritme.

Kombinering av betting algoritmen ble gjort med en funksjon kalt på av Vue. Denne kjører etter at løpet er opprettet. For at bettingen skulle kunne bli lagret i databasen, måtte kontrolleren returnere både modellen for griser og for brukere. Dette kunne bare bli gjort vis vi brukte ViewModel, men det fikk vi desverre noen problemer med, derfor valgte vi å lage en ny view og gjøre transaksjonen med databasen derifra. Det ble først prøvd å brukes html elementer til å lagre og vise senere, men dette gikk ikke når det skulle sendes til et annet view i betting prosessen. Derfor ble det valgt at vi brukte “SessionStorage” til å lagre variabler i. Denne informasjonen ble brukt i “MoneyTransaction Page” hvor den ble sendt til kontrolleren med en form, deretter blir disse lagt inn i databasen av kontrolleren.

## 5.4 Racing grafikk

Griseløpet er presentert i en 3D verden. Denne verdenen består av griser på en løpebane. Grafikken er vist og animert ved hjelp av “Three.js”. Dette Javascript biblioteket lar oss skrive grafikk som kjører lokalt hos klienten. “Three.js” bruker “WebGL” slik at programmet kjører i nettleseren uten at det må lastes inn noe annet en Javascript kode.

Mer spesifikt forklart opprettes en scene/koordinatsystem. Her plasserer vi diverse objekter, blant annet grise modellen. Denne er laget i Blender og blir importert med “Three.js” bibliotekt “GLTFLoader”. Utover dette blir også flater, lys og kamera plassert i scenen. Alt dette skal vises på et lærer som ligger på spill siden. På lærer vises et bilde av hva kamera ser. Dette bildet blir oppdatert 1000 ganger i sekundet eller så ofte nettleser og GPU tillater det. For hver oppdatering har kamera og grisene flyttet seg litt fram. Grisenes posisjon blir hentet inn fra en tabell skapt av racing algoritmen. Tabellen repeteres gjennom slik at griser og kamera flyttes, deretter vises et nytt bilde på lærer.

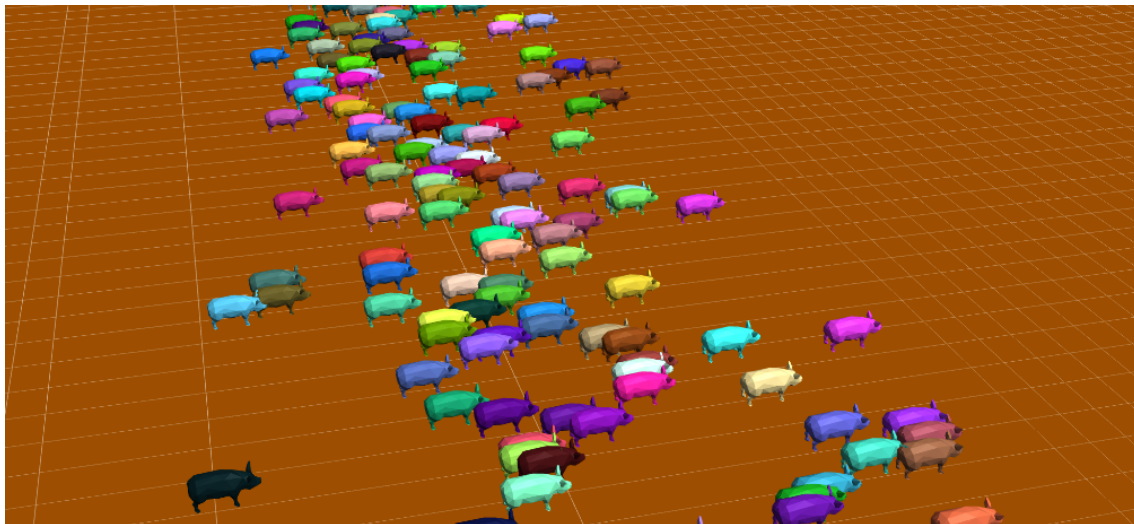
```
function updateFrame() {  
  renderer.render(scene, camera); // renderer tegner informasjonen på lærer.  
  requestAnimationFrame(updateFrame); // Vi oppdaterer konstant lærer med det nyeste bildet.  
}
```

Figur 5: Scenen blir tegnet på lærer

```
function AnimateObjects(tabell) { // En tabell med posisjoner er hentet fra racing algoritmen.  
  frame++; // En global variable sier hvileken frame vi har kommet til.  
  for (let pig in pigs)  
    pig.position.z = tabell[frame];  
}
```

Figur 6: Pseudokode for oppdatering av grisenes posisjon

Denne delen av prosjektet ble veldig såret av mangel på tid. Vi skriver mer om dette i "7.7 Videre arbeid".



**Figur 7:** En variable bestemmer antall griser

## 5.5 Play side

“Play” er siden hvor brukeren går for å vedde på griser. Spilleren velger hvor mye de vil vedde og hvilken gris de vedder på. Det blir så vist en simulasjon av reser og til slutt en tabell med resultater.

Vue bestemmer om hva som er synlig på siden ved bruk av “v-show”. Først med *var div* elementer som viser hvor mye spilleren kan vedde. Button elementer sender valget til en “session storage” og en Vue variabel. En Vue funksjon blir kalt på som kaller på en Javascript funksjon. Denne funksjonen velger fire tilfeldige griser fra databasen. Informasjonen om disse blir sendt til både visning i html elementer og lagret midlertidig i “session storage”. html elementene viser hvilke griser som kan veddes på og knapper som sender valget videre. Informasjonen i “session storage” blir sendt til racing algoritmen. Algoritmen returnerer koordinater, disse blir brukt i racing grafikken se 5.5 Racing grafikk. Når grafikken vises gjør Vue et kall som gjemmer alt annet enn en button og grafikken. Når denne knappen blir trykt gjør Vue en funksjon som kaller på en Javascript funksjon. Denne funksjonen finner ut om brukeren har vunnet og hvor mye penger brukeren har vunnet eller tapt. Dette blir lagret i en “session storage” og brukeren blir sent til en annen side med navn “Moneytransaction”.

## 5.6 MoneyTransaction side

MoneyTransaction side er en side som brukeren blir sent til etter et løp er ferdig. Når vinduet har lastet inn vil en funksjon som heter "onloadbli kalt på. Denne funksjonen tar informasjonen om hvor mye brukeren har vunnet eller tapt, og automatisk kaller på en Form. Denne Formen sender informasjonen som "posttil "PlayController", som gjøre endringen i databasen. Etter dette vil funksjonen sette opp en synlig tabell og to knapper. Tabellen viser hvilke plasser de ulike grisene avsluttet. Knappene viser til en gå hjem funksjon og en spill igjen funksjon.

## 5.7 PigOperations

På denne siden har vi funksjoner som lager nye griser, rediger griser og slette griser. Disse funksjonene og siden er kun tilgjengelig for Admin.

[Home](#)

[Race](#)

[Play](#)

[LeaderBoard](#)

[Betting](#)

[Truffle Clicker](#)

[Pig Configuring](#)

Hello Pigmania@uia.no!

Logout

# Index of Pigs

[Create New](#)

Name	Speed	
Oskar	20	<a href="#">Edit</a>   <a href="#">Delete</a>
Lars	14	<a href="#">Edit</a>   <a href="#">Delete</a>
Markus	13	<a href="#">Edit</a>   <a href="#">Delete</a>
Giar	14	<a href="#">Edit</a>   <a href="#">Delete</a>
Shaheen	15	<a href="#">Edit</a>   <a href="#">Delete</a>
Anders	10	<a href="#">Edit</a>   <a href="#">Delete</a>

**Figur 8:** Siden til operasjonene som du kan gjøre på en gris innlogga som Admin

For å legge til griser starter koden med et skjema som har to felter for navn og hastighet på grisen. Valideringen er satt til "ModelOnly", som betyr at det kun vil validere mot denne modellen for å sjekke at det kommer riktig verdier inn. Den første seksjonen inneholder et *inndatafelt* for navn og et inndatafelt for fart. Den andre seksjonen inneholder en *indataknapp* for submit. Når denne klikkes sendes data til serveren via AJAX (Asynchronous JavaScript And XML).

Når bruker skal redigere en gris så vil samme oppsettet oppstå som å lage en ny gris, men da kan du velge en eksisterende gris fra listen.(se figuren under)

Pigmania Play LeaderBoard Truffle Clicker **Pig Configuring** Hello Pigmania@uia.no! Logout

## Edit Pig

Name

Speed

[Save](#)

[Back to List](#)

**Figur 9:** Siden til Edit

*Pig Configuring* er markert gult for å vise hvor i Navigasjons baren denne siden befinner seg. (Bare Admin kan se dette)

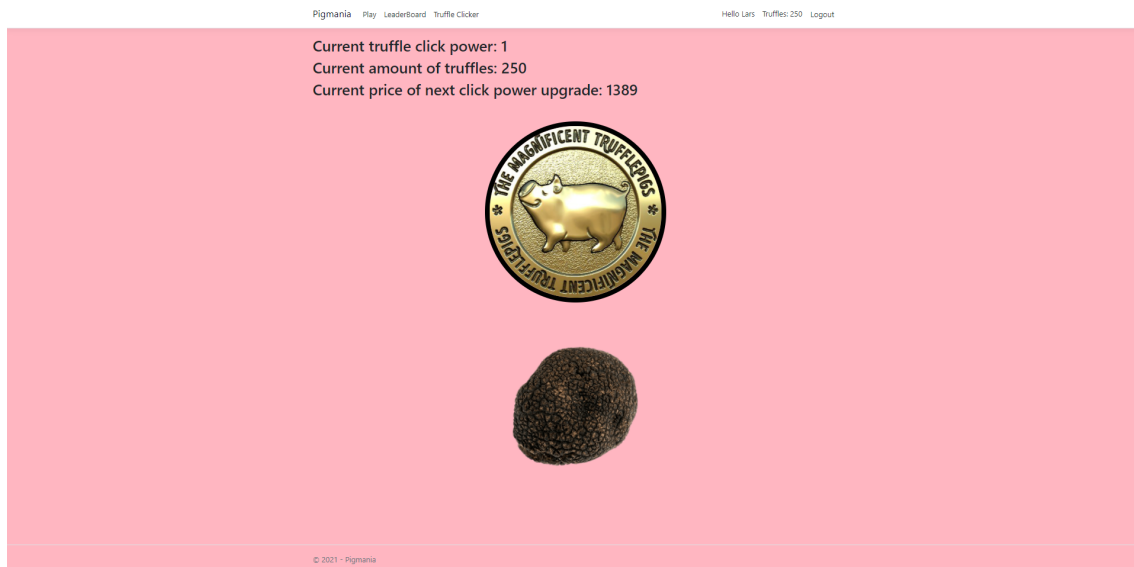
```
@if (User.Identity.IsAuthenticated)
{
    if (User.IsInRole("Admin"))
    {
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Pig" asp-action="Index">Pig Configuring</a>
        </li>
    }
}
```

**Figur 10:** Dette er hvordan vi implementerte admin autentiseringen for pig konfigurering i Rider

Denne pseudo koden viser hvordan vi bruker Identity framework til å autorisere at brukeren er Admin.

## 5.8 Truffle Clicker

Truffle clickeren består av to knapper og tre tekstfelt. Disse tre tekstfeltene sier hvor mange Truffle Coins brukeren har, antall oppgraderinger brukeren har og hvor mye neste oppgradering koster. Knappene er ikoner som har ulike funksjoner. Den ene knappen ser ut som en mynt og er hovedknappen for spillet. Denne knappen genererer Truffle Coins hver gang den trykkes. Den andre knappen er en oppgraderingsknapp som øker clickpower. Dersom bruker kjøper en click power oppgradering, vil antall trufflecoins man får per klikk gå opp med antall clickpower oppgraderinger. Antall oppgraderinger og Truffle Coins lagres i bruker modellen som en attributt.



**Figur 11:** Truffle clicker siden. Her ser vi oppsettet vi har brukt. Mynten genererer Truffle Coins, trøfflen er oppgraderingsknappen

For at det ikke skal bli enkelt å kjøpe oppgraderinger, har vi en formel som øker prisen til oppgraderingen for hver oppgradering brukeren har kjøpt. Vi henter antall truffleCoins og truffleClickPower fra brukeren og setter det inn i formelen for å regne ut prisen til neste oppgradering.

```
ApplicationUser applicationUser = (ApplicationUser) await UserManager.GetUserAsync(User);
var truffleCoins:int = applicationUser.TruffleCoins;
var truffleClickPower:int = applicationUser.TruffleClickPower;
double priceTemp = 2500 * truffleClickPower / 1.8;
int price = Convert.ToInt32(priceTemp);
<script>
document.getElementById("truffleClickPower").innerHTML = @truffleClickPower
document.getElementById("truffleCoins").innerHTML = @truffleCoins
document.getElementById("clickPowerCost").innerHTML = @price
</script>
```

**Figur 12:** Formelen til oppgraderingsprisen (linje 4 - 5). For å unngå desimaltall, konverterer vi prisen fra å være en double til å bli en integer.

Vi har valgt å utføre mesteparten av funksjoner og håndtering av databasen i back-end. Grunnen for å ha operasjoner i back-end er for å gjøre fusk vanskeligere. Å forhindre fusk er en viktig del av et spill som inneholder konkurranseorienterte funksjoner, som for eksempel leaderboard. Begge knappene kaller på en clicker kontroller

som utfører databasefunksjoner.

```
public async Task<IActionResult> EditClickPower([Bind(params include: "TruffleClickPower")] ApplicationUser user)
{
    if (ModelState.IsValid)
    {
        user = _um.GetUserAsync(user).Result;
        user.TruffleCoins = user.TruffleCoins;
        double priceTemp = 2500 * user.TruffleClickPower / 1.8;
        int price = Convert.ToInt32(priceTemp);
        if (user.TruffleCoins >= price)
        {
            user.TruffleCoins = user.TruffleCoins - price;
            user.TruffleClickPower = user.TruffleClickPower;
            user.TruffleClickPower = user.TruffleClickPower + 1;
            _context.Users.Update(user);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Add));
        }
    }
}
```

**Figur 13:** Eksempel på funksjon i kontroller. Figuren viser funksjonen til oppgraderingsknappen

## 5.9 Leaderboard

På leaderboard siden viser vi en rangering over alle brukere. Spillerene blir rangert ut i fra antall trufflecoins de innehaber. For å vise spillerene i leaderboardet henter vi først ut bruker modellen fra databasen. Deretter itererer vi over hver bruker i modellen og legger til brukernavnet, samt antall trufflecoins i en rad på leaderboardet. I stedet for å ha rangeringen tilknyttet modellen har vi valgt å implementere den via css og ha den assosiert med raden. Den siste kolumnen på leaderboardet er forbeholdt win raten til spilleren. Grunnet tidsbegrensninger fikk vi ikke tid til å implementere denne funksjonaliteten.

## 6 Testing og validering

Vi gjorde manuelle tester underveis som vi kodet. Det fleste testene var veldig uformelle tester for å se at ting fungerer før de pushes til branch, som for eksempel når vi la til Playername i AspNetUser så testa vi at vi kunne bruke playername i leaderboardet, vi lagde nye brukere med Playername og vi prøvde å endre på Playername før vi pusha det videre.

I tillegg gjorde vi mere strukturerte tester som resulterte i user stories. Vi hadde ikke eksterne brukere som kunne hjelpe oss, istedenfor prøvde vi å sette oss inn i hvordan en bruker tenker.

### 6.1 Testing av User stories

Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å registrere meg, for å kunne logge meg inn	Brukeren kan registrere seg på nettsiden	Brukeren kan registrere seg på nettsiden	Ja

**Tabell 1:** User-story Registrere Bruker

Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å logge meg inn, for å identifisere meg selv	Brukeren kan logge seg inn på nettsiden	Brukeren kan logge seg inn på nettsiden	Ja

**Tabell 2:** User-story Innlogging av bruker

Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å kunne velge hvor mye jeg skal gamble for å kunne bestemme mine egene gevinster og tap	Brukeren kan velge hvor mye han skal gamble på et race	Brukeren kan velge hvor mye han skal gamble på et race	Ja

**Tabell 3:** User-story Innlogging av bruker



Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å kunne velge en gris jeg kan vedde på	Brukeren kan velge grisen han vil vedde på	Brukeren kan velge grisen han vil vedde på	Ja

**Tabell 4:** User-story Vedde på en Gris

Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å skaffe meg penger andre steder enn et løp	Brukeren kan velge å skaffe penger andre steder enn et løp	Brukeren kan velge å skaffe penger andre steder enn et løp	Ja

**Tabell 5:** User-story Skaffe penger uten å gamble på et løp

Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å kunne se hvilken rangering jeg er i forhold til andre	Brukeren kan velge å se hvilken rangering brukeren har i forhold til andre	Brukeren kan velge å se hvilken rangering brukeren har i forhold til andre	Ja

**Tabell 6:** User-story Bruker kan se ledertavle

Description	Expected Outcome	Actual Outcome	Passed
Som en bruker, har jeg lyst til å ha muligheten til å se et løp når jeg gambler	Brukeren kan velge å se på løpet som brukeren satsa på	Brukeren kan velge å se på løpet som brukeren satsa på	Ja

**Tabell 7:** User-story Bruker kan se løpet

Testing av user-stories ble gjort ved hjelp av den fysiske og tekniske valideringen vi hadde. F.eks da vi testet om koden vår fungerte, kjørte vi et fysisk program som viste oss koden kjørte vellykket. Ved bruk av black-box og white-box metoden var det en mulighet for å verifisere og validere koden som ble skrevet. For eksempel ved bruk av black-box metoden vil en annen i gruppen teste den fysiske versjonen av koden, og i dette tilfellet tar vi logg inn funksjonen. Da en bruker logger inn må testerene kunne

se brukenavnet til brukeren som er logget inn. Men før dette ble validert, måtte han som skrev koden verifisere om koden som ble skrevet fungerte på den rette måten.

Dette kan bli gjort ved bruk av white-box metoden da det kommer opp på databasen at den følgende bruker er logget inn. Når en komponent av koden blir satt for unit test av en annen i gruppen, så blir den enten validert eller ikke. Dersom koden ikke blir validert av testeren så må den rettes på før ny test. Om koden blir validert, kan den merge med andre branch. Da koden blir satt sammen, gjør begge programmererne en integrerings test på den sammenslåtte koden. Om intergreingen feiles, må koden fikses før den blir satt for testing igjen, men dersom den fungerer så blir den kombinerte komponenten sammenslått med resten av komponentene for å gjøre et system test. Vi hadde ikke en spesifikk test person, så testing som ble gjort var at alle i gruppen testet alle funksjonalitetene i produktet. Dersom det var noe å fikse på, ble det sagt ifra til personen som gjorde den spesifikke funksjonaliteter, men om alle var fornøyde så var produktet klar for bruk.

## 7 Diskusjon

I starten av prosjektet hadde vi høye forventninger til hva vi skulle lage. Vi har underveis i prosjektet måtte synke disse. I dette kapitlet skal vi se på problemer og hvordan vi kunne unngått disse. Videre sammenlignes resultatet vårt med krav. Prossessen våres skal tas opp, samt produktet vi har lagget, implementasjonen av den og detaljer om de større utfordringene vi møtte. Til slutt skal vi se hva vi hadde lagt til produktet om vi hadde mulighet.

### 7.1 Hva kunne blitt gjort annerledes?

Hvis vi hadde startet med prosjektet på nytt ville vi gjort en bedre planleggingsfase. Vi skulle blitt mer enige på hvordan slutt produktet skulle se ut. I tillegg har vi nå et mer realistisk syn på hvor mye tid, arbeid og kunnskap det er å lage et slikt produkt. Vi var altfor optimistiske om hvor mye vi kunne gjøre med våre forkunnskaper og den tiden vi ble gitt.

Vi var for raske til å begynne å kode. Vi lagde user stories mot slutten av prosjektet, noe vi helst skulle hatt før vi begynte å kode. På denne måten ville vi hatt et mye mer konkret mål å jobbe mot. Vi hadde ikke tenkt på at produkt visjonen vi lagde i begynnelsen av prosjektet, ble kravene våres.

I tillegg ble ikke arbeidet fordelt utover prosjektet på en god måte. I gruppekontrakten som alle skrev under på så tenkte vi at vi skulle jobbe 108 timer per gruppemedlem. Når vi ser på timeloggen så ser vi at i gjennomsnitt hadde vi 31 timer fordelt på alle medlemene de første 5 ukene. Når eksamensperioden kom så hadde ingen av oss tid til å fortsette på dette prosjektet. Siden dette prosjektet hadde senest leveringsfrist ble den også prioritert lavest. Da hadde vi rett og slett ikke tid til å implementere og gjøre det vi egentlig ville. Alt i alt så tar vi dette med oss videre og er mer fokusert på det fremover.

### 7.2 Resultater i forhold til krav

Hvis vi sammenligner vårt produkt med hva vi hadde tenkt til i produkt vision se tabell 7. Så er resultatet vårt ganske anderledes en vi hadde tenkt. Det kommer mye av at vi ikke har hatt kravene i bakhodet mens vi holdt møter slik at vi kunne drøfte hvordan vi skulle få dette til. Vi var for sentrert på å fullføre den vertical slicen til å i det hele tatt tenke på den nødvendige planleggingen som går i et produkt som dette. Dette er delvis grunnen til at vi ikke klarte å lage det produktet vi hadde sett for oss i produktvisjonen. Deretter måtte vi simplifisere produktet siden tidsrammen ble mindre enn vi hadde planlagt. Vi valgte heller å nedskalere produktet med hensikt å få et fungerende produkt.

Produkt	produkt vision krav
Nei	Offline Racing
Nei	Trading
Nei	User owned pig
Nei	Online Racing
Nei	Upgrading
Nei	Multi platform
Nei	Breeding
Ja	Betting på gris

**Tabell 8:** User-story Produkt vs Krav

Online racing skulle enten være race hvor griser eid av forskjellige spillere er med. Eller det skulle være race hvor flere spillere kunne vedde. Mens offline racing skulle være et alternativ til dette. Her skulle brukere kunne trene grisene sine eller tjene virtuell valuta. Det racet vi endte opp med å implementere dekker ingen av disse kravene helt.

### 7.3 Prosess

Vi hadde mye å gjøre dette semesteret. Denne innleveringen var den siste vi hadde, derfor har andre fag blitt prioritert. Dette førte til intenst arbeid den siste uken. Heldigvis ble vi flinkere på Git og klarte å løse problemene vi hadde tidlig i prosjektet. Så det var ikke så vanskelig å løse de samme problemene da vi møtte på dem senere siden vi allerede hadde funnet ut hva vi skulle gjøre. Dette førte til at vi ble flinkere på Git.

### 7.4 Produkt

Før vi startet produksjonen lagde vi en produktvisjon. I denne beskrev vi funksjoner vi ønsket at produktet skulle ha.

Vi har ikke oppnådd alle målene vi ønsket. Gjennom prosjektet har vi klart å inkludere de viktigste komponentene som får programmets hoved funksjoner til å fungere. Etter hvert som innleveringsfristen nærmet seg begynte vi å innse hva som kunne gjøres i den korte tidsperiode vi hadde igjen.

Vi skulle ha brukere som kunne ha griser og eventuelt kunne parret(breed) for å lage nye griser. Disse grisene skulle ha ulike attributter som ga dem unike egenskaper til å øke sjansen til å vinne løp. Men å koble database, brukere, griser, attributter, grisene brukerene velger for løpet, nye griser som kommer og algoritmer som blir brukt, ville ha vist seg å være en svært tidskrevende implementasjon. Fra planeleggingen til det endelige produktet kan vi si at vi mangler en del side funksjonaliteter

som kunne gjort spillet bedre.

Leaderboard tabellen skulle vise brukere, antall trufflecoins de innehaber og winraten. Men siden vi koblet veddingen brukeren gjør på grisene så sent så fikk vi ikke tid til å vise winraten for veddingen til hver bruker. Dette er noe vi mangler i leaderborden som kunne ha blitt implementert, men som ikke ble gjort.

Som en ekstra inntekt for brukere lagde vi en truffleclicker som gir brukere en alternativ inntektskilde dersom de ikke vinner noe veddemål eller om de har tapt for mange og ønsker å vedde igjen. Dette er en bra side hustle for alle brukere som prøver å klatre leaderboardet.

Vi bestemte oss for at bare administratorer kan styre attributtene til grisene. Denne funksjonaliteten kom inn i bildet siden vi innså at vi ikke kunne tildele en gris til hver bruker.

## 7.5 Implementasjon

Det har skjedd mange ulike implementasjoner i dette prosjektet. Mange av dem har bydd på utfordrende problemer som vi har brukt mye tid på å løse. For eksempel å sette sammen to ulike koder, eller installere pakker.

Kasjeke det vi er mest fornøyd med er hvordan vi har fått til det utsene. I løpet av prosjekter har vi blitt rutinere på å bruke html, C sharp, “Three.js” og bootstrap for å utforme grafikk. Med dette har vi laget noen fine sider og et godt utgangspunkt til et animert race.

Vi brukte mange workarounds for å få race og betting til å fungere. Vi hadde trengt mer tid for at dette skulle fungert optimalt. Men med den implementasjonen vi har mener vi at vi har dekket kravet om å ha betting. Vi har et fungerende race, men det er anderledes fra sånn vi tenkte i oppstarts fasen. Vi mener det er så anderledes at det ikke dekker kravet om offline eller online race.

Vi er fornøyd med hvordan autentisering fungere. Dette var en lett implementasjon siden vi hadde hjelp av modeller i Rider. Noe som mangler tilknyttet detter er persistent storage. Når brukeren krysser er all data borte. Å implementere dette var ikke prioritert fordi det ikke var relevant til emnet.

Vi er også fornøyd leaderbordet, her henter vi ut informasjon om brukere fra databasen. Og presenterer den med css. Denne siden samt hjemmesiden og play siden viser hva vi har lært om webdesign i løpet av prosjektet.

Vi synes at Truffle Clickeren ble implementert på en god måte med tanke på den tidsbegrensingen vi hadde. Denne henter informasjon fra bruker modeller og oppdaterer disse. Denne funksjonaliteten er hovedsakelig implementert i backen.

## 7.6 utfordringer

I løpet av prosjektet har det selvsagt oppstått mange utfordringer og none problemer tar mer tid enn andre. I dette kapitlet vil vi nevne noe av de større problemene vi møtte. Flere av problemene oppsto fordi vi ønsket å bruke “Three.js” for å animere racet. Vi hadde god erfaring med dette bibliotekt fra før, men ikke i et prosjekt av denne størrelsen. Noe av det som har konsumert mest tid var Vue. Vi hadde ikke brukt dette eller noe lignende bibliotek før. I tillegg til alt dette var det utfordrende å implementere betting.

### 7.6.1 Kombinere grafikk og race algoritme

Algoritmen som skriver data om løpet og den grafiske representasjonen ble programmert separat. Å kombinere disse gikk ut på å kopiere dataen om løpet til grafikk funksjonen. Dette viste seg å bli en utfordring. Koden til grafikken er delt opp i moduler fordi den bruker modulbaserte biblioteker. Moduler i JavaScript er segregerte komponenter som kan laste hverandre med *export* og *import*. Disse modulene fungerer ikke med “vue.js”. En tidlig løsning på dette var å sende dataen gjennom html dokumentet. Vi valgte denne løsningen fordi den var lett. En annen løsning ville vært å bruke en module-bundler (f.eks Webpack). En module-bundler kan pakke grafikk koden inn i en fil som Vue kan aksessere.

Det var løsningen vår før vi ble mer erfarne med å sende informasjon mellom script. Da fant vi ut at vi kunne bruke “SessionStorage” til å lagre variabler. Ved å bruke dette kan vi flytte dataen i backend. Dette ble den endelige løsningen på hvordan vi kunne sende data mellom et script som var en module og et som ikke var.

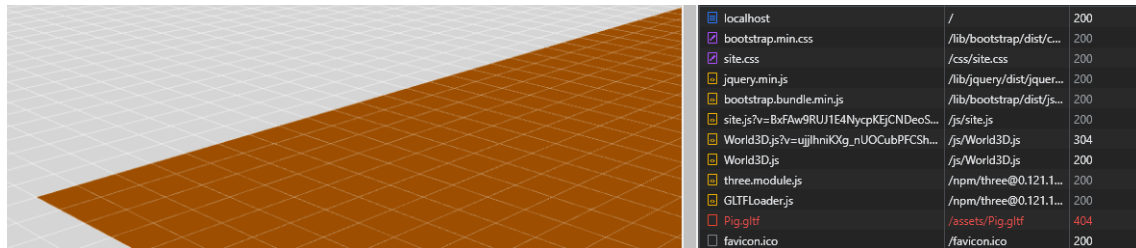
```
function input(pig1){
    sessionStorage.setItem("one", pig1);
}

function output(){
    return sessionStorage.getItem( key: "one");
}
```

Figur 14: Flytte informasjon mellom script i backend

### 7.6.2 Implementere Pig model

Et team medlem hadde erfaring med grafikk fra valgfaget DAT200. Dette utnyttet vi til å lage grafikken for selve løpet. En 3 dimensjonal verdenen er opprettet i “three.js”. Grise modeller er opprettet som *.gltf* filer. Disse blir importert inn i 3D verden. Dette ble først implementert i Web-storm, en IDE egnet for Javascript. Problemet oppsto når dette skulle overføres til Rider. Når prosjektet kjørte fra Rider fant ikke klienten *pig.gltf* filen.



Figur 15: HTTP gir 404, ingen grise modeller dukker opp

Vi visste ikke nøyaktig hva som forårsakte denne feilen. Derfor gikk feilsøking ut på å endre relaterte ting og se hva som hadde innvirkning. Vi brukte mye tid på dette uten å oppnå så mye annet enn å utelukke løsninger. På et tidspunkt vurderte vi å gi opp, endre modellen til en boks og kalle prosjektet “Box run”. Heldigvis fikk vi istedenfor hjelp av Universitetslektor Christian Auby. Han kom fram til at problemet skyldes at Rider ikke tillater ukjente filtyper. Vi fant ut dette ved å simpelt endre filtypen til *.json*. Siden *.gltf* filer bruker *.json* format trenger vi ikke gjøre noen videre endring og nå laster modellen inn.

### 7.6.3 Vue.js Singel-Page-application

Vi ble enige i starten av prosjektet at vi skulle bygge opp nettsiden vår med Vue, fordi vi har hatt litt om Vue i en av de siste forelesningene og vi snakka med Christian Auby på veiledningen om hvordan vi kunne bruke Vue i prosjektet vårt. Det førte til at vi tenkte Vue var nyttig for oss. De første ukene tenkte vi å bruke Vue til å lage prosjektet som en singel-page-application med å da dynamisk bytte siden vi viser i forhold til hva brukeren trykket på. Siden vi tenkte å få til vertical slicen først, kom vi fram til at vi bare skulle lage en midlertidig løsning for å kunne vise griseløpet med Vue. Dette viste seg å være en litt mer jobb enn det vi hadde regnet med så derfor ble det ikke noe av singel-page-application.

#### 7.6.4 Vue.js som et live løp av racing algoritmen

Vi hadde tenkte først at vi skulle bruke Vue.js til å dynamisk endre på posisjonene til grisene i løpet live(Real-time). Dette fikk vi altså ikke til fordi det ble for vanskelig å bruke Vue inne i et Javascript file, hvor algoritmen vår var plassert, sånn som vi hadde tenkt. I vårt nåværende produkt så blir vinneren av løpet kalkulert for å så bli brukt i grafikken til løpet, på view page'en. Siden Vue er et høynivå programmeringsspråk/metode så handler det mye om forståelsen av bilotekene som vue er bygget opp av. Biblioteker som for eksempel nye typer funksjoner(async/synchronous), hooks, components, templates osv. vi klarte å bruke Vue som en type GUI kontroller, hvor mange funksjonaliteter som v-show var brukt, men vi hadde mange utfordringer med alle de ukjente delen av vue som vi ikke helt klarte å implementere.

#### 7.6.5 Betting

Prosjektet er bygget opp rundt tema-et betting, så derfor var det viktig å få riktig implementert et slik system. Vi hadde flere utfordringer som omhandlet betting systemet, men alle problemene våres omhandlet hvordan vi skulle oppdatere coins i databasen. Vi gikk fra å bruke html elemeter til å lagre hvor mye som brukeren hadde vunnet eller tapt, til å bruke ViewModel for å lagre i databasen med en view som hadde to modeller. Tilslutt endte vi opp med å bruke "sessionstorage" til å bruke informasjonen i et annet view. Det var så mange utfordringer i de ulike delene at vi hadde lagd flere ulike måter å lagre informasjonen på før vi fikk det til på en mindre kompleks måte. Selv den siste versjonen som brukte "sessionstorage" har sine mangler. Den største mangelen den har er at den ikke vil oppdatere databasen før du har trykker på show results.

### 7.7 Videre arbeid

I startfasen av prosjektet var vi svært ambisiøse overfor egne evner og tidskrav. Dette førte til at vi hadde altfor høye forventninger til de ulike funksjonalitetene vi ønsket å implementere. På grunn av dette kombinert med uforutsette utfordringer fikk vi simpelthen aldri tid til å begynne på mange av de funksjonalitetene vi så for oss i et komplett produkt. I tillegg måtte vi kutte ned på enkelte av de essensielle funksjonalitetene i produktet. Dette har vi lært av og vi tar med oss et mer realistisk syn på tidsestimering og arbeidsbelastning fremover.

#### 7.7.1 Forbedre grafikk

Nameplates er kanskje den viktigste funksjonen vi mangler. Uten dette klarer man ikke skille grisene fra hverandre. En annen måte det skulle vært mulig å se forskjell



på grisene er fargen deres. Modellene blir gitt tilfeldige farger, dette var egentlig bare ment som en placeholder. Det er tenkt at hver gris skal ha en attributt i databasen som beskriver fargen deres. Dette ville nok blitt implementert som en string med en heksadesimal fargekode. Videre ville det vært naturlig at grisene hadde en løpe animasjon. Animasjoner kan bli importert på samme måte som grise modellene. Det er også mange andre mindre forbedringer vi har tenkt på underveis i arbeidet.

Fortløpende hvilke andre forbedringer vi kunne lagt til: naturlig tempo, flytte hit-box til grisens nesetipp, forbedre kamera vinkler, skybox, live ledertavle, bakgrunns detaljer og podium.

### **7.7.2 Egen gris**

I prosjektet vårt hadde vi planlagt at hver bruker skulle kunne ha sin egen gris som man kunne bruke i løpene. Med en egen gris så ønsket vi et spillopplegg der brukeren kunne ha mer innvirkning på hvert løp. Tanken var at brukeren startet med en gris med lav hastighet, men som kunne få høyere hastighet ved å bruke penger på å trene grisen. Brukeren ville kunne eie flere griser og velge hvilken gris som skal brukes i hvert løp.

### **7.7.3 Case opening**

Dette er en annen feature som vi ønsker å ha med i et ferdig produkt. Med Truffle clicker og gambling systemet vårt ønsket vi å ha muligheten til å kjøpe Loot boxes. I en loot box ville innholdet vært ulike griser med forskjellige utseender og hastigheter. Disse grisene ville også ha en attributt som bestemmer hvor skjelden en gris er, dette ville vi kalt for pig rarity.

### **7.7.4 Rarity basert racing**

For å kunne ha en balanse over hvilke griser som kan være i samme løp, ønsker vi å implementere en form for restriksjon for løp. Vi ønsker at ulike løp har forskjellige krav, som for eksempel at i et spesifikt løp kan det kun være griser fra kategorien "common rarity". Dette vil utelukke griser som har lavere eller høyere rarity-grad slik at hvert løp blir spennende og mer fair. Dette kunne også bli brukt til å øke multiplikasjonsfaktoren til veddebeløpet ditt, slik at dersom du spilte mot sjeldnere griser får du mer utbytte om du vant.

### **7.7.5 Multiplayer**

Dersom hver bruker har mulighet til å ha egen gris og kunne åpne loot box for å få flere og bedre griser, ønsker vi å ha en multiplayer modus for at spillere kan være i løp mot hverandre. Dette skaper enda mer spenning dersom man interakterer med andre

mennesker istedet for kun systemet. Dette gjør også at leaderboard funksjonen vår blir satt i et bedre lys. Vi kan utvide leaderboard med å kunne vise kun multiplayer-løp, som vil øke spillengasjement for konkurranse-interesserte spillere.

#### **7.7.6 Coin table (backend)**

Grunnet tidsmangel på prosjekt bestemte vi at pengene til brukeren skal bli lagret som en attributt i bruker-modellen. Dette er ikke så lurt på sikt, og dermed ønsker vi å kunne utvide penger til å ha sitt eget tabell i databasen med et forhold mellom penger og bruker. Dette fører til at dersom vi legger til flere pengetyper eller oppgraderinger i truffle clicker så vil det være enklere å lagre nødvendig informasjon.

#### **7.7.7 Truffle clicker**

Truffle clicker ble ikke helt ferdig i tide for innleveringen. Truffle clicker er ment som en annen måte å tjene penger på utenfor hovedspillet. Dette er ment som en slags failsafe dersom brukeren går tom for penger, så kan brukeren spille truffle clicker for å samle en pengesum for å kunne gamble videre.

#### **7.7.8 Pig breeding**

En feature vi aldri rakk å starte på var et konsept vi kalte for pig breeding. Ideen var at brukere som eide to eller flere griser skulle ha mulighet til å avle griser og produsere nye avkom med unike egenskaper. Dessverre grunnet tidsbegrensinger ble dette for ambisiøst, men i et komplett produkt hadde det vært en svært interessant funksjon å implementere.

## 8 Konklusjon

Vi ønsket å lage et nettleserbasert gambling spill basert på griseløp. Hensikten med produktet var å utvikle et alternativ til eksisterende dyre gambling tjenester. Brukeren skulle ha mulighet til å nyte en engasjerende spillopplevelse uten den potensielle konsekvensen av økonomisk ustabilitet. En betydelig del av tiden vi la ned i prosjektet brukte vi på å utvikle en robust “vertical slice” komprimert av nødvendig kjernefunksjonalitet. Resultatet vårt er dermed den nåværende versjonen av Pigmania. Et spill der brukeren trygt kan gamble med valuta uten innvirkning på egen økonomi. Ettersom dette viste seg å være såpass tidskrevende rakk vi ikke å implementere flere funksjonaliteter vi så for oss i et komplett produkt, og dette ville vært neste skritt for Pigmania. Vi er fornøyd med det endelige resultatet, spesielt grafikken. Med den funksjonalitetn vi har nå har brukere mulighet til å registrere seg og begynne å spille med en gang, samle Truffle Coins og stige opp på leaderboardet.

## 9 Gruppekontrakt

Anders Farstad Myrmel, Giar Rehani, Lars-Erik Bakkland Moi, Oskar Eftedal Markussen, Markus Roland, Shaheen Thayalan

### **Obligatoriske møter:**

Møte hver Mandag 13:00 er obligatorisk.

### **Konsekvenser av forsinkelse og manglende oppmøte:**

Problematisk oppførsel resulterer i en advarsel. Hvis et medlem mottar 3 advarsler, bør det avholdes avstemning om å ta opp problemet med kontaktperson / Scrum product owner (Christian Auby).

### **Meldeplikt:**

Dersom et gruppemedlem ikke kan delta på et møte må de varsle om dette. Varselet skal være en melding i Discord prosjekt gruppen. Meldingen må inneholde "@everyone".

### **Konfliktløsning:**


Når meningsfull konflikt oppstår må dette tas opp i et møte. I så fall kan dette tas opp i det ukentlige obligatoriske møte. Om saken haster eller problemet oppstår tidelig i uken skal det innkalles til et nytt møte. Dette møte er obligatorisk og reglene ovenfor gjelder.

### **Forventet arbeidsmengde:**

Standard arbeidsmengde per studiepoeng er 27 timer og dette projektet er verdt 4. Derfor må hvert gruppemedlem bidra med 108 timer. I løpet av 8 uker tilsvarer dette ca. 13,5 timer i uken. Om mer tid er nødvendig for å gjøre prosjektet operativt før fristen forventes det at alle gruppe medlemmer tar ansvar for det.

Alle prosjekttimer må logføres i Jira.

**Anders Farstad Myrmel**

Signature: 

Date: 18.10.2021

---

**Giar Rehani**

Signature: 

Date: 18.10.2021

---

**Lars-Erik Bakkland Moi**

Signature:  LARS-ERIK MOI

Date: 18.10.2021

---


**Oskar Eftedal Markussen**

Signature:  Oskar E. Markussen

Date: 18.10.2021

---


**Markus Roland**

Signature:  Markus Roland

Date: 18.10.2021

---

**Shaheen Thayalan**

Signature: 

Date: 18.10.2021

---

|