TMA4850
Eksperter i team - Matematikk innen anvendelser
Prosjektrapport

Bekkemoen, Y.
Brungot, K.
Martinussen, J. G.
Verås, H.
Wilhelmsen, H.

April 2019

Innhold

Sammendrag

Vi beskriver her arbeidet utført i Eksperter i team-landsbyen *TMA4850 - Matematikk i anvendelser*. Målet for landsbyen er å utvikle et selvvalgt matematikkrelatert produkt som både dekker deltagernes tverrfaglige kompetanse og samtidig har et samfunnsnyttig aspekt.

Dette er gjort ved å implementere et spill som går ut på å bevege et spillobjekt gjennom fysiske kraftfelt. Underveis i spillet er det mulig å endre på fysiske parametere relatert til bevegelse, for eksempel friksjonskoeffisienten. Intensjonen er at spilleren skal kunne forstå de underliggende sammenhengene mellom de fysiske konstanter og hvordan objekter beveger seg.

Gruppens faglige kompetanse dekker kybernetikk, datateknologi, statistikk, fysikk, og industriell økonomi og teknologiledelse med fordypning i datateknologi.

Gyroskopet til en mobil ble tatt i bruk for å styre et spillobjekt der signaler fra mobilen må prosesseres. Signalene ble deretter sendt fra mobilen til en datamaskin vha. en nettverksprotokoll der kø-teori ble anvendt for å sende meldingene. Bevegelsen til spillobjektene påvirkes av inputsignalet og løsningen til et sett med fysiske differensialligninger. Bayesiansk statistikk har blitt anvendt for å analysere spillresultatene, for å rangere alle historiske spillere etter ferdighetsnivå.

I tillegg til matematiske prinsipper, krevde implementasjonen av spillet kompetanse innen programvareutvikling. Dermed fikk samtlige gruppemedlemmer tatt i bruk sin fagkompetanse.

Hvor samfunnsnyttig et spill er, vil alltid være diskuterbart. Det håpes imidlertid at ved riktig bruk, som for eksempel i en undervisningsammenheng, at spillet kan bidra med en praktisk innfallsvinkel til å øke forståelsen av teoretiske begreper innen fysikken.

Spillutvikling er en iterativ prosess, og det er flere mulige aspekter som kan videreutvikles og forbedres i spillet. Blant de viktigste potensielle forbedringene er mulighet for lagspill og motspillere, og å implementere en motspiller basert på kunstig intelligens.

Introduksjon

Matematikk i anvendelser er et tema som oppfordrer til bruk av kompetanse innen matematikk, anvendelse og regnekraft(computing). Dette samspillet illustreres av MAC-modellen. I denne oppgaven ble landsbytemaet tolket som en mulighet til å anvende en matematisk tilnærming til å utforske en problemstilling som faller utenfor de tradisjonelle definisjonene av det matematiske fagfeltet. Sentralt for Eksperter i Team (EiT) er også *samarbeidskompetanse gjennom erfaringslæring*. Dette ble tolket som en mulighet til å utvikle ferdigheter innenfor teamarbeid, og å lære mer om hvordan man anvender eget fagområde til å lage et helhetlig produkt, på tvers av flere fagfelt. Dette gir medlemmene en mulighet til å trene på formidling av egen fagkompetanse i et tverrfaglig team, og utvide sitt perspektiv på egen fagkompetanse. Et tredje viktig aspekt ved oppgaven er at produktet skal være *innenfor et samfunnsrelevant tema*.

Med utgangspunkt i disse kriteriene, ble det bestemt at gruppen ønsket å utvikle et spill som går ut på å bevege et spillobjekt gjennom fysiske kraftfelt. Gruppen mener spillutvikling nettopp er et eksempel på matematikk i anvendelser. Fra lineær algebra i bilderendring til matematisk modellering av spillmekanismer, ligger matematikken som en grunnstein i spillutvikling. Spill krever store mengder regnekraft, ofte med begrenset beregningstid. Dermed fanger MAC-modellen opp en rekker aspekter ved spillutviklingen. Spillutviklingen gjorde det også mulig for hvert enkelt gruppemedlem å bruke sin fagkompetanse i ulike deler av prosessen. Dette blir forklart i detalj i `Metode`-kapittelet. For å imøtekomme kriteriet om samfunnsnyttighet, ligger spillets fokus i å bidra til økt fysikkforståelse- og interesse, hvilket blir utdypet senere i dette kapittelet.

Tverrfaglighet og arbeidsfordeling

Den tverrfaglige gruppen består av fem medlemmer med høy grad av teknisk kompetanse. Den tverrfaglige kompetansen dekker følgende fagområder:

- Datateknologi, med fordypning i kunstig intelligens
- Industriell matematikk, med fordypning i statistikk
- Industriell økonomi og teknologiledelse, med fordypning i datateknologi
- Kybernetikk og robotikk, med fordypning i fartøystyring
- Teknisk fysikk, med fordypning i numerisk fysikk

Denne tverrfagligheten økes ytterligere ved at et gruppemedlem har en Bachelorgrad innen matematisk optimering, og at flere gruppemedlemmer har stor interesse for programmering og bruker mye tid på dette utenfor studietid. Industriell økonomi og teknologiledelse kan sees på som en hybrid mellom ulike fagområder. Med tanke for fordypningen, ble det naturlig at det tverrfaglige bidraget fra Industriell økonomi og teknologiledelse ville være i en data-tekologisk retning.

Det viste seg imidlertid å være vanskelig å definere et prosjekt som dekket fagfeltet til alle gruppemedlemmene i nøyaktig like stor grad. Det ble dermed nødvendig at noen av fagfeltene fikk en større plass enn andre. Et eksempel på dette kan være at spillet krever mye programmering for å virke, men spisskompetansen til datateknologistudenten, kunstilig intelligens, ikke kom frem under dette prosjektet. Fysikkstudenten fikk på den andre siden mulighet til å løse differensialligninger numerisk, hvilket er langt nærmere fordypningen til fysikkstudenten.

Arbeidfordelingen ble bestemt ved at spillet deltes opp i mer eller mindre uavhengige moduler. Gruppemedlemmene kunne derfor bruke sin fagkompetanse på å utvikle hver sin modul, som så ble koblet sammen til ett helhetlig produkt. Modulene, med tilsvarende fagområde er gitt i følgende tabell:

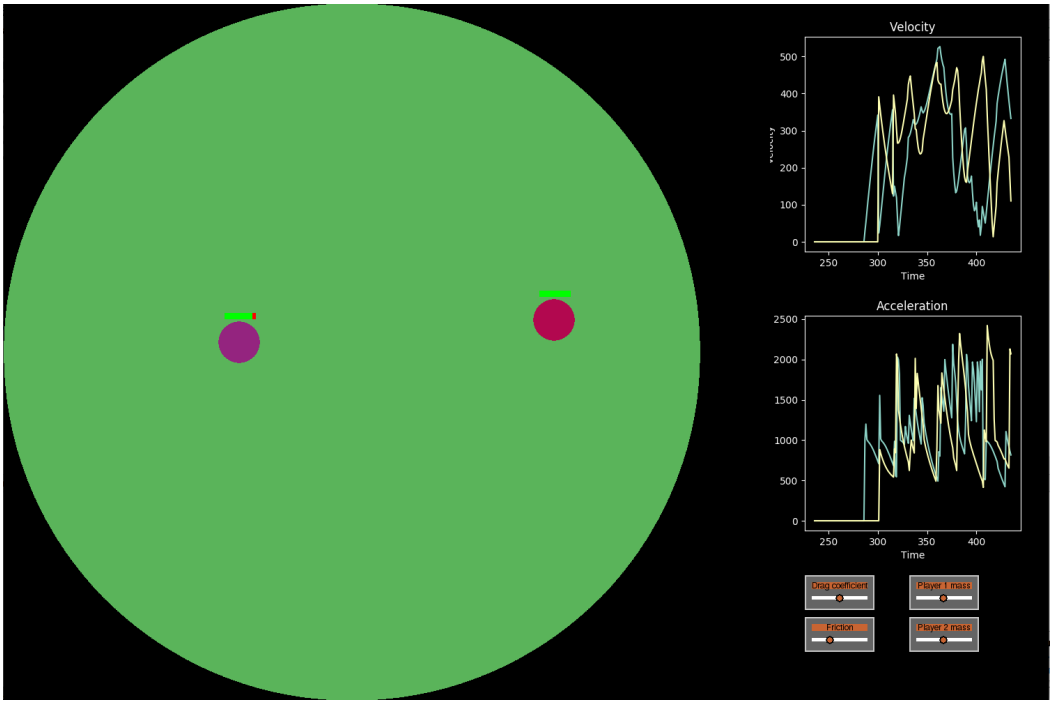
Tabell 1: Oversikt over moduler med assosierte fagområder i spillet. Implementasjon av modulene blir beskrevet i detalj i hvert sitt delkapittel i `Teori og metode`-kapittelet.

Modul	Fagområde
Grafikk	Datateknologi
Spill	Datateknologi
Input	Kybernetikk
Fysikk	Fysikk
Statistikk	Statistikk

En av de tverrfaglige utfordringene ble dermed å lage gode løsninger for å knytte modulene sammen til et helhetlig prosjekt.

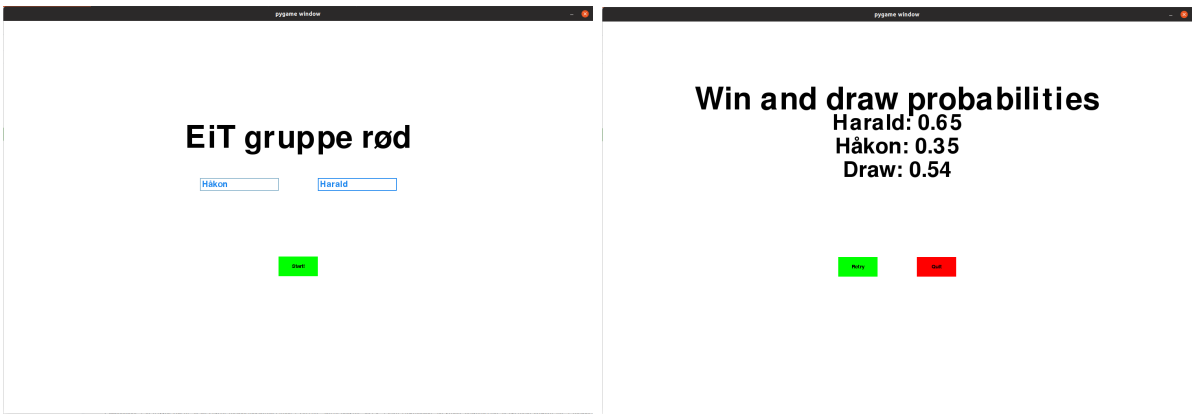
Innledende beskrivelse av produktet

Målet med spillet er å overleve ved å holde seg innenfor et markert område på et todimensjonalt brett. Kommer man utenfor dette området vil man miste *helsepoeng*, og går man tom for helsepoeng stopper spillet. Flere spillere kan spille mot hverandre. Man ønsker å dytte motstanderen utenfor det markerte området, slik at motstanderen mister sine helsepoeng. For å unngå å bli dyttet kan man bruke et skjold, men man kan da ikke bevege seg. For å styre spillet kan man enten bruke piltastene, eller koble til en mobiltelefon. Mobilen vil da bruke gyroskopet til å styre bevegelsen til et gitt spilobjekt. Spillbrettet er vist i Figur 1



Figur 1: Utklipp som viser spillets grensesnitt.

Spillet er implementert slik at bevegelsen til spillobjektene er styrt av fysiske lover. Disse lovene, beskrevet i `fysikkmodul` kapittel, er avhengige av ulike fysiske parametere. Tre av parameterene, `friksjonskoeffisient`, `luftmotstandskoeffisient` og `spillobjekt-masse` er implementert slik at de kan varieres mens spillet kjører. Settes friksjonskoeffisienten lik null, forsvinner friksjonen, settes luftmotstandskoeffisienten lik null, forsvinner luftmotstanden, og settes massen lik null, forsvinner akselerasjonen. Variasjon av parametrene kan altså brukes til å forstå hvordan



(a) Startskjermen (b) Sluttskjerm

Figur 2: Utklipp som viser grensesnittet av start- og sluttskjermen i spillet.

fysiske lover fungerer. Samtidig gir det brukeren mulighet til å lære om hastigheter, krefter og kollisjon gjennom prøving, lek og utforskning. For å tydeliggjøre effekten av å variere parametrene, er det blitt implementert grafer som oppdateres i sanntid. Disse viser farten og akselerasjonen til spilleobjektene.

Regjeringen publiserte i 2010 en strategi for styrking av realfag og teknologi [8]. Her heter det at "Høy kompetanse i realfag og teknologi er en forutsetning for å møte dagens og morgendagens store utfordringer. Vi trenger tilstrekkelig antall mennesker med innsikt slik at vi kan forstå utfordringene og handle på rett måte." [8]. Hvor samfunnsnyttig et spill egentlig er kan diskuteres, men formålet med dette spillet er å tilby et produkt som kan bidra til nettopp det å øke folks kompetanse, interesse og forståelse for realfag og teknologi.

På side 30. i rapporten sier kunnskapsdepartementet at "For å øke rekrutteringen til realfag og teknologiske fag [...], er det viktig å gjøre realfagene mer virkelighetsnære og meningsfylte for elevene."[8]. En naturlig arena hvor man ser for seg at spillet har potensiale er i forbindelse med undervisning. Dette spillet vil kunne fungere som en praktisk tilnærming for å øke fysikkforståelsen og mestringsfølelsen for fysikk. I undervisningssammenheng kan spillet introduseres som et praktisk alternativ, eller supplement til tradisjonell teoretisk undervisning. Økt forståelse og mestringsfølelse kan bidra til at elever får økt motivasjon innenfor realfag. Kunnskapsdepartementet skriver i sin rapport at "Mange har vansker med matematikkfaget når det gjelder både ferdigheter og motivasjon", og dette spillet kan nettopp være med på dette. Spillet dekker kun en liten del av fysikkpensum, men ved å gi brukeren mestringsfølelse her, kan gi motivasjon til å utforske andre fagområder innenfor realfagene.

Spillet er publisert som *åpen kildekode*, der kodebasen finnes på `GitHub`[1]. En sentral tanke har vært at spillet skal kunne videreføres og videreutvikles av andre brukere etter deres behov. Koden som blir beskrevet utfyllende i `spill` kapittelet er objektorientert med en abstrakt klasse `ArenaLayer`. Denne klassen kan arves hver gang en bruker ønsker å legge til en ny kraft. Det håpes med dette at nye fysiske kraftfelt skal kunne implementeres og undersøkes i spillet. Dette åpner for at brukere kan lære mer om både fysikk og om generell utvikling av kode, og intensjonen er igjen, ved å gi brukeren muligheten til å eksperimentere og få følelsen av å lage et eget produkt, kan spillet bidra til både mestringsfølelse og interesse for fysikk og realfag.

Teori og metode

Vi vil i dette kapittelet presentere i detalj hvordan spillet er bygget opp, fra dets grunnlag i matematikken, til selve koden og hvordan det hele flettes sammen til det ferdige spillet. Som beskrevet i introduksjonen, er prosjektet basert på tverrfaglighet, og naturlig måte å fordele arbeidet på var å splitte koden opp i moduler, slik beskrevet i Tabell 1. Det bør nevnes at det var kontraktfestet at alle deltagerne i gruppen skulle ha en overordnet forståelse av alle moduler, også modulene som ikke direkte var relatert til personens fagområdet. Dette ble ansett som nødvendig for å sikre et helhetlig produkt der alle modulene interagerte på en naturlig måte. På den måten kunne tverrfagligheten utnyttes til å skape et felles, helhetlig produkt. Dette vil i tillegg bidra til å øke gruppemedlemmenes faglige kompetanse utenfor deres egne fagområder, og øke forståelse for hvordan deres kompetanse påvirker et slikt prosjekt.

Teknologier

For å utvikle spillet ble en rekke teknologier vurdert. Hvis feil teknologi velges kan det bli dyrt i form av tid og ressurser. Teknologiene som ble valgt er listet i Tabell 2, der de viktigste teknologiene blir presentert med en diskusjon av fordeler og ulemper.

Tabell 2: Alfabetisk oversikt over software brukt i implementasjonen av spillet.

Modul	Verson
HyperIMU	3.0.4.6
Numpy	1.14.5
Matplotlib	3.0.3
Pygame	1.9.6
Pytest	4.2.0
Python	3.7.3
Scipy	1.2.0
TrueSkill	0.4.5

Python

I prosjektet ble *Python* brukt som hovedprogrammeriskspråk. Python er et høynivå programmeringsspråk som oftest betyr at koden er både enklere å lese og å skrive, enn tilsvarende lavnivåkode. Det er også et generelt programmeringsspråk, som betyr at det kan brukes til mange former for programmeringsoppgaver.

Python kommer med mange ferdige rammeverk. Dette gjør at man ofte kan gjenbruke funksjonalitet istenden for å utvikle den samme funksjonaliteten fra bunnen av. Alt dette gjør Python til et godt valg for en gruppe uten mye tidligere samarbeidserfaring, og med begrenset med tid.

Ulempen med Python er at det ikke er et kompilert språk, hvilket kan skape ytelsesproblemer om man ikke skriver optimal kode. Dette er viktig for et spill, da man ønsker et brukervennlig produkt uten forsinkelser. Python har muligheten til å bruke rammeverk skrevet i kompilerte språk. Dermed kan man, ved god bruk av rammeverk, fremdeles oppnå akseptabel ytelse.

Pygame

Pygame er et rammeverk i Python som kan brukes til å lage spill. Det er pakket rundt et annet rammeverk i C kalt *Simple DirectMedia Layer* som brukes til å styre multimedia. Styrken til Pygame er at man kan skrive Python kode, og samtidig ha hastigheten til programmeringsspråket C. En annen ting som gjør Pygame atteraktiv er at det fungerer på alle operativsystemer, hvilket gjør utviklingen lettere. Det er et konsist rammeverk, og gjør mye arbeid for programmereren. Det gjør at det krever lite kode for å få opp et fullverdig spill på kort tid.

Selv om Pygame er et Python rammeverk, gir den utviklere full kontroll på alle aspekter ved et spill, som om det var skrevet i et lavnivå programmeringsspråk. Dette gjør at det er få begrensninger på hva man kan lage i Pygame, fra et spill med få funksjonaliteter til et spill med komplekse mekanismer. Dette gjør det mulig å tilpasse kompleksiteten til spillet avhengig av hvor mye tid og ressurser man har tilgjengelig.

Grafikken i Pygame er slik at man har et lerret som representerer spillvinduet. Grafikkelementer som man ønsker å tegne, legges på dette lerretet. Hvis man ønsker å fjerne et element fra dette lerretet, må hele lerretet tegnes over. Alle elementene på lerretet blir da fjernet, og må tegnes på nytt. For å unngå å tegne alle elementene på nytt kan man i steden tegne nye elementene over elementer som allerede er tilstede, slik at gamle elementer sjules. Rammeverket gjør mye av det tunge arbeidet ved spillprogrammering, men om det introduseres for mye funksjonalitet vil det gå på bekostning av ytelsen.

Numpy

Numpy er et bibliotek i Python som i hovedsak brukes til vektor- og matriseoperasjoner[3]. Rammeverket har sin egen datastruktur `ndarray`, hvor det er definert en rekke elementvise opperasjoner. Dette er spesielt nyttig i fysikk modulen og `arena`-klassen, hvor man ønsker å lagre spillebrettet som en matrise, og gjøre elementvise opperasjoner på fysiske vektorer som posisjon eller kraft. Numpy er optimalisert med kompilert C kode, slik at opperasjoner på ndarrays ofte går langt fortere enn tilsvarende kode skrevet i ren Python.

Matplotlib

Matplotlib er et populært bibliotek for 2D plotting i Python[7]. Rammeverket blir brukt til plotting av hastighets- og akslerasjonsgrafene i sanntid. Matplotlib er lett å bruke og komme igang med, og i tillegg har flere i gruppen erfaring med biblioteket. Ulempen med Matplotlib er at det først og fremst er laget for å lage statiske grafter, og ikke optimalisert for å ha ytelse til å plotte i sanntid. Men samtidig er det en av få biblioteker som er kompatibel med Pygame.

HyperIMU

HyperIMU er en kraftig sensorapplikasjon som hjelper utviklere og forskere til å samle inn mange typer sensordata ut av en enhet for å kunne utføre analyse og behandling. En Smartelefon inneholder flere typer innebygd sensorer som kan brukes til å utvikle et utallige antall applikasjoner på mange felt. Ulempen med HyperIMU er at det skjeldent oppdateres, hvilket gjør rammeverket mindre robust.

Spilloppbygning

Ved bruk av et objektorientert design, ble modulene fra Tabell 1 implementert som uavhengige komponenter. Dermed blir det lettere å videreutvikle spillet for nye personer. Oppbygning består av en `spill`-klasse som holder kontroll på alle mekanismer og tilstander i spillet. `spill`-klassen bruker andre objekter til for eksempel å tegne grafikk eller flytte på spillere. Det faktiske arbeidet skjer altså ikke i spillklassen, men i objektene spillklassen styrer slik at `spill`-klassen kun er et tilknytningspunkt. `Spill`-klassen har egne metoder for å kjøre spillintro, spillet og spillavslutning separat slik at utvidelse av introduksjonen og avslutningen er uavhengig av de andre delene av spillet.

Arena

`Arena`-klassen representerer banen i spillet, og inneholder elementer som både skal brukes i forbindelse med fysikkmodulen og grafikkmodulen. Dette reflekteres i de to metodene til klassen, `draw` og `force`.

```
class Arena:
    [...]

    def draw(self, screen) -> None:
        """Draw the arena onto the given screen."""
        Circle(self.position[0], self.position[1],
               self.color, self.radius).draw(screen)

    def force(self, player: Player, input_force: np.ndarray, max_input: float, dt: float
              ) -> np.ndarray:
        """Calculate the force from all ArenaLayer objects, for a given player"""
        total_force = np.zeros(2, dtype=float)
        for layer in self.layers:
            total_force += layer.force(
                player=player,
                input_force=input_force,
                total_force=input_force + total_force,
                max_input=max_input,
                dt=dt,
            )
        return total_force
```

`draw` har til hensikt å tegne spillbrettet til skjermen. Dette blir gjort for et svart rektangel med en grønn sirkel, som vist i Figur 1.

`force` har til hensikt å regne ut kraften i et punkt på spillbrettet, og returnere denne kraften til fysikkmodulen. For å tydeliggjøre at det er uavhengige fysiske kraftfelt på spilbrettet, ble kraftfeltene representert ved en `ArenaLayer`-klasse. Da spillet har to kraftfelt, ble det implementert en `FrictionLayer`- og en `AirResistanceLayer`-klasse, som begge arver fra `ArenaLayer`. Virkemåten er beskrevet i Fysikkmodul kapitelet, men det kan nevnes at begge klassene har som oppgave å finne krefter assosiert med kraftfeltete, for alle spiller-objektene, og returnere kraften til `force`, som returnerer videre til fysikkmodulen.

Spiller

`Spiller`-klassen representerer en spiller i spillet, og inneholder tilstanden til en spiller. Det som kjennertegner en spiller er at den kan styres, dytte en motstander, har helsepoeng og kan bruke skjold.

Under ser man hvordan tilstanden til en spiller er satt opp.

```
def __init__(self, x: float, y: float, data,
                 phone: ConnectPhone = None) -> None:
        """
        Initialize a player with given parameters.

        :param x: initial x-position.
        :param y: initial y-position.
        :param data: Dataclass used to store player settings.
        """
        self.mass: float = data.mass
        self.position: np.ndarray = np.array([x, y], dtype=float)
        self.velocity: np.ndarray = np.zeros(2, dtype=float)
        self.input: Input = Input(data, phone)
        self.color: Colors = Colors.random()
        self.data: type = data
        self.shield_on: bool = False
        self.health_bar: HealthBar = HealthBar(self, PlayerSettings.health)
```

For hver spiller tas det inn en `x` og `y` verdi som er startposisjonen til en spiller. `data` er et dataklasse objekt som inneholder konstanter for en spiller som radius. `phone` er en variabel som representerer en mobil, og brukes hvis man ønsker å styre en spiller med en mobil. Det som skjer i metoden `__init__` er at alle verdier som en spiller trenger blir satt opp ut ifra de gitte variablene. Numeriske verdier representeres som flyttall og Numpy arrays.

Hver spiller har helsepoeng, og tilstanden og mekanismer rundt helsepoeng styres av en egen `HealthBar` klasse. Alle mekanismer med en viss kompleksitet er faset ut til en egen klasse. Dette gjør at man lett kan utvide spillet uten at hele spillet kollapser.

En spiller har to tegne methoder `draw` og `draw_shield`. `draw` tegner spilleren til skjermen, og har ingen andre funksjonalitet utenom. `draw_shield` har lignende funksjonalitet som `draw` metoden, forskjellen er at `draw` tegner spilleren, mens `draw_shield` tegner skjoldet til spilleren. Begge metodene benytter seg av Pygame sine innebygde metoder for å utføre jobben.

Skjoldet til en spiller er laget slik at det ikke er mulig å bevege på seg når det er skrudd på. Skjoldet sin effekt er at den fyller spilleren med helsepoeng når den er på, men samtidig gjør at spilleren ikke kan bevege på seg.

En spiller har egen variabel for navn som man kan skrive i introduksjonssiden. Navnet brukes i forhold til lagring av poengstatistikk i statistikkmodulen. All statistikk til en spiller er det statistikkmodulen som håndterer. Hvordan statistikkmodulen fungerer er beskrevet i kapitelet om statistikkmodul.

Input

Styringsmekanismen til en spiller er kapselet rundt et objekt av klassen `Input`.

```
class Input:
    """Represents the input module for players."""

    def __init__(self, keys: type, input_phone: ConnectPhone) -> None:
        """
        Initialize an input module for the given player.
        :param keys: The keys for the given player.
        :param input_phone: A phone to control the input.
        """
        self.keys: type = keys
        self.input_phone: ConnectPhone = input_phone
        if self.input_phone:
            self.input_phone.set_up()
```

`Input`-objektet håndterer styringsmekanismen til en spiller, og gjør at det går an å bygge avanserte styringsmekanismer enkelt, uten at det påvirker `spiller`-klassen. Som det kan ses på koden over så initialiseres `Input`-klassen med både piltaster og mobil. Slik at hvis man ikke har tilkoblet mobil så er det fortsatt mulig å styre en spiller med taster på tastaturet. Når man styrer en spiller med en mobil vil det være støy, dette fikses ved å ha et filter som blir beskrevet i detalj i inputmodul kapitelet.

Grafikk

Alle grafikkelementer i spillet består av en egen klasse, det gjør at det er lett å utvide dem, og samtidig mulig å gjenbruke dem. Det som er felles for alle grafikkelementer er at de inneholder en `draw` metode. `draw` metoden tegner det gitte grafikkelementet på skjermen, og gjør det ved å benytte seg av Pygame sine innebygde metoder. Andre elementer som grafer benytter seg også av Matplotlib i tillegg til Pygame for å tegne.

Utenom `spiller`-klassen finnes det også andre `grafikk`-klasser som `slider`-klassen og `graf`-klassen. De implementerer også en `draw` metode ettersom de kan instansieres og tegnes på spillvinduet. `Graf`-klassen består av Matplotlib figurer som kan tegnes på spillvinduet, og det inkluderer alle former for grafer man kan tegne med Matplotlib. Selve Matplotlib figurer tegnes på spillvinduet som bilder som lastes inn i Pygame. `Slider`-klassen håndterer slides som brukes til å justere ulike variabler i spillet.

Innstillinger

For å håndtere konstanter i spillet som friksjonskoefisienten, ble det brukt dataklasser. Hovedformålet med klassene er å lagre ikke muterbare data som lett kan aksesseres og endres hvis nødvendig. Under ser man et eksempel på en dataklasse som lagrer de ulike fysikkonstantene spillet benytter seg av.

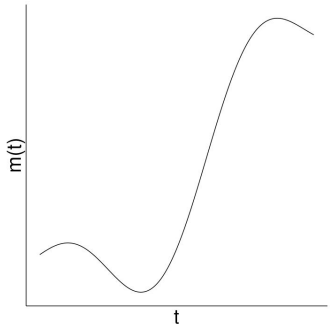
```
@dataclass
class PhysicsConsts:
    """Constants for the physic engine."""
    friction_const = 1.0        # The coefficient of friction
    drag_coefficient = 0.01     # The coefficient of air drag
    input_modulation = 1500     # Moulation to make the input force to correct magnitude
    force_modulation = 50       # Moulation to make the arena force to correct magnitude
    static_friction_limit = 5   # Velocity limit before static friction occurs.
```

Dette gir lesbar kode, og gjør utvikling og endring av spillet enklere. Spillet har ulike dataklasser for ulike deler av spillet, som fysikkonstantklassen for fysikkmodulen.

Inputmodul

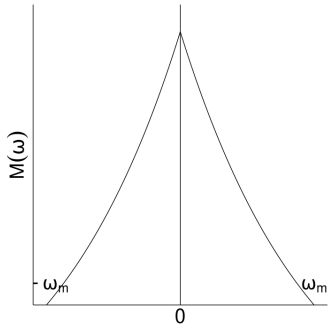
For å styre spillet ble gyroskopet til en sammenkoblet mobil tatt i bruk. Uheldigvis er det få muligheter for å modifisere signalet før mobilen sender signalet ut, og for å oppnå et naturlig inputsignal ble det implementert en inputmodul som tar seg av signalprosesseringen. Det prosesserte signalet blir så gitt videre til fysikkmodulen. Det bør også nevnes at om man ikke kobler til en mobil kan spillet styres fra tastetrykk. Et tastetryk vil representere en konstant verdi. Imidlertid trenger ikke et slikt signal å prosesseres. Derfor trenger ikke tastetrykk signalet å diskuteres i detalj, i motsetning til som mobiloutput-signalet.

Samplings-teorem



Figur 3: Illustrasjon av et generelt kontinuerlig meldingssignal, $m(t)$.

For å bruke input fra en generell inputkilde kan man bruke et verktøy kalt sampler. Sampleren har til hensikt å redusere et kontinuerlig signal til et diskret signal. Meldingssignalet er begrenset til et gitt antall frekvenser, dvs. et båndbegrenset meldingssignal. Videre kan man analysere inngangsignalet ved å Fouriertransformere meldingssignalet til frekvensdomenet, hvor hver frekvenskomponent blir representert som en amplitude, slik det er illustrert i Figur 4.

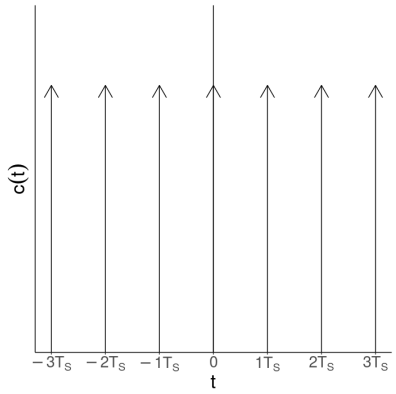


Figur 4: $M(\omega)$, frekvenskomponenter i $m(t)$.

Man oppnår et bånd hvor båndbredde er representert av $b \in \{-\omega_m, \omega_m\}$. Samplereren blir beskrevet som en multiplikator med en Dirac delta funksjon i faste samplings-tidspunkter. Ved å definere samplings-funksjonen $c(t)$ og samplings-intervallet T_s, arver man fra samplings-intervallet en samplings-frekvens ω_s. Denne samplings-frekvensen har enhet rad/s og er gitt ved $\omega_s = \frac{2\pi}{T_s}$. Samplings-funksjonen er gitt ved

$$c(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s), \tag{1}$$

hvilket representerer et impulstog, illustrert i Figur 5, med intervall lik samplings-intervallet.



Figur 5: $c(t)$, Impulstog, Dirac delta.

Sampleren satt sammen med det generelle inngangsignalet kan dermed kombineres til

$$\begin{aligned} & s(t) = m(t) \cdot c(t), \\ \implies & S(\omega) = \mathcal{F}\{m(t) \cdot c(t)\} = \frac{1}{2\pi}[M(\omega) * C(\omega)], \end{aligned} \tag{2}$$

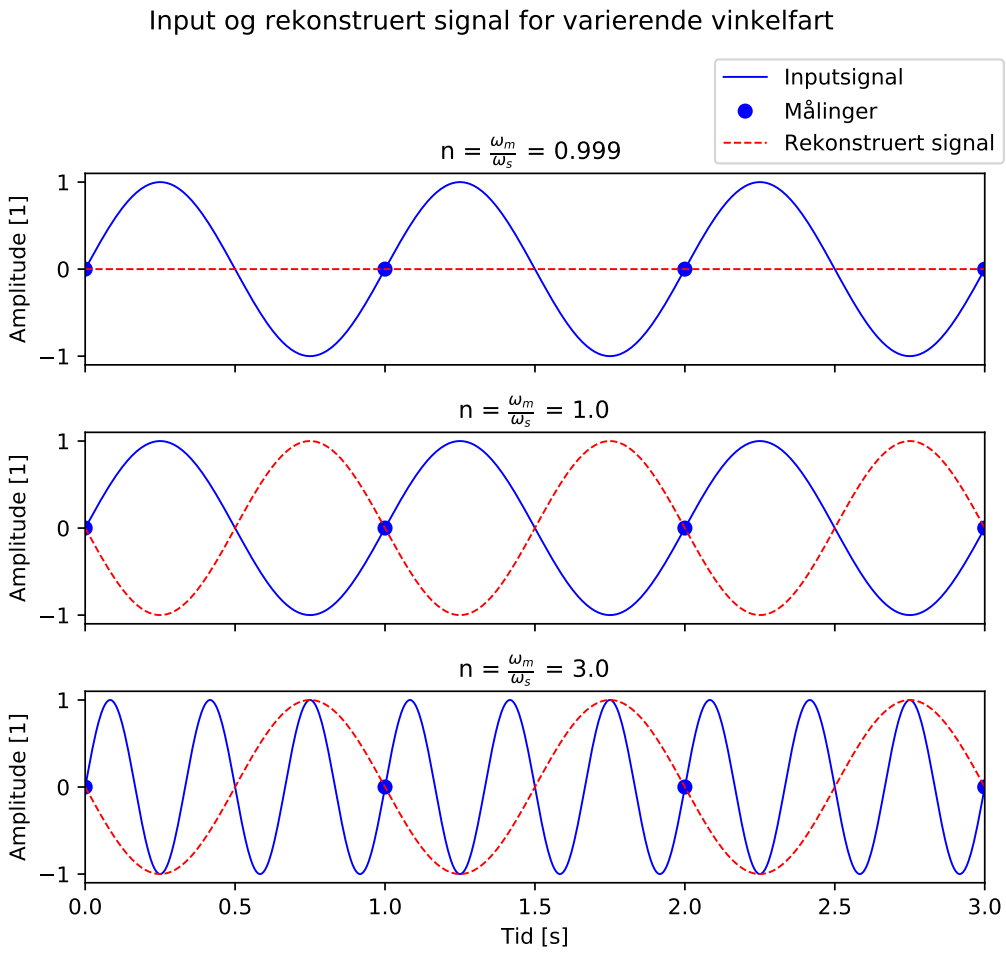
hvor bruken av transformasjon av en multiplikasjon i tidsdomenet fører til en konvolusjon i frekvensdomenet. Ved bruk av

$$\begin{aligned} & \mathcal{F}\{c(t)\} = \omega_s \sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_s), \\ \implies & S(\omega) = \frac{1}{2\pi}[M(\omega) * \omega_s \sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_s)], \\ \implies & S(\omega) = \frac{\omega_s}{2\pi} M(\omega) * \sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_s), \\ \implies & S(\omega) = \frac{1}{T_s} \sum_{n=-\infty}^{\infty} M(\omega) * \delta(\omega - n\omega_s). \end{aligned} \tag{3}$$

For å få en bedre forståelse av hvordan dette vil se ut, kan man utvider summen i Figur 3 til

$$S(\omega) = \frac{1}{T_s}[... + M(\omega + \omega_s) + M(\omega) + M(\omega - \omega_s) + ...] \tag{4}$$

hvilket resulterer i at et samplet signal vil gi en syklisk repetisjon avhengig av hvor fort man sampler signalet. Geometrisk kan vi se at dersom $\omega_s < 2\omega_m$ vil vi få en overlapp i signalet og vi vil miste informasjon dersom vi utfører en sampling med samplings-frekvens mindre enn halvparten av den største komponenten i meldingssingalet. Problemet som oppstår kalles folding og er et problem som kan løses ved enten en begrensning av båndbredden til inngangen til sampleren, eller ved å redusere samplings-tiden eller ekvivalent øke samplings-frekvensen. Foldingsproblematikken er illustrert i Figur 6. Figur 6 viser utvalgte inngangssignaler med konstant samplingstid, og det resulterende rekonstruerte signalet. Øverste plot viser det rekonstruerte signalet når ω_m nærmer seg ω_s fra venstre. I det plottet i midten er $\omega_m = \omega_s$. I det nederste plottet blir signalet gjenskapt dersom inngangssignalets frekvens er tre ganger samplingsfrekvensen. I alle tilfellene er det betydelige forskjeller i rekonstruert signal. Det er nettopp dette man vil forsøke å unngå ved å båndbegrense signalet.



Figur 6: Folding av input signal, a) i grensen $\omega_m \to \omega_s$, b) $\omega_m = \omega_s$ og c) $\omega_m = 3\omega_s$

Foldingsfilter

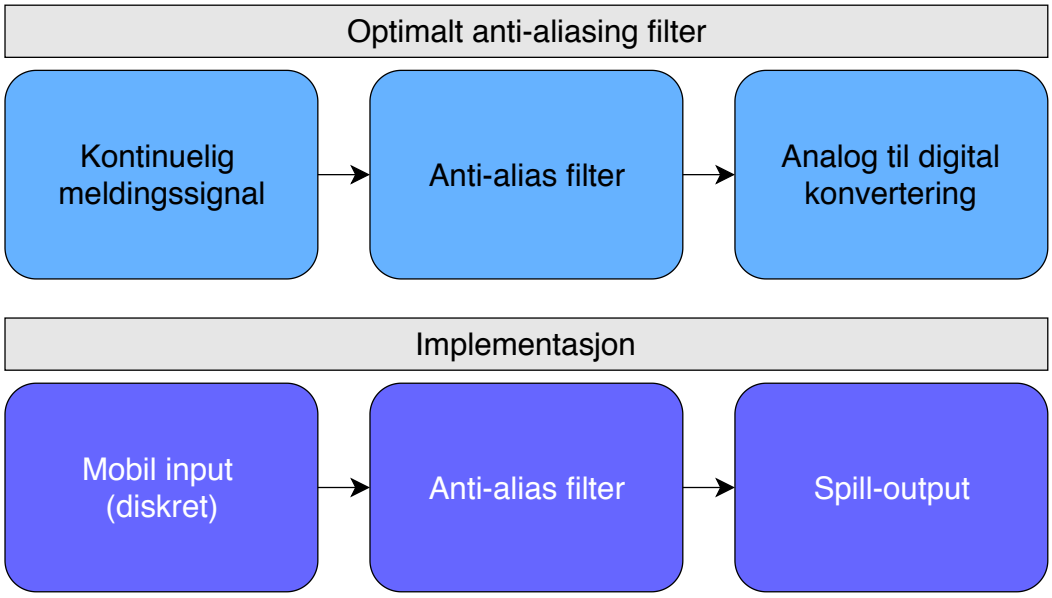
Anti-alias filtrering kan bli brukt når man ønsker å gjøre en analog til digital konvertering. I spillet brukes akselerometeret til en mobil som input kilde. Da man ikke ønsker å modifisere mobilens maskinvare, er det naturlig å vise en teoretisk fremgangsmåte, og realisere filteret digitalt. Filteret vil gi spillobjektet en reduksjon av brå bevegelser fra spilleren. Denne effekten kalles rykk, hvilket vil si den deriverte av akselerasjonen. Som vist i Figur 7 ble det laget et filter som befinner seg mellom telefonen og spillets inngangsargument.

For å analysere den ønskede oppførselen til filteret kan man sette opp en rekke krav til filteret

- Filteret skal ikke gi stasjonært avvik.
- Filteret skal ha en innsvingningstid som er regulerbar.
- Filteret burde ikke ha oversving.
- Filteret er av første eller andre orden.
- Filteret skal ha en justerbar forsterkning.

Fremgangsmåten for å lage et førsteordens lavpass filter baserer seg i hovedsak på å finne en ønsket respons fra filteret i kontinuerlig tid, for så å transformere systemet til diskret tid. Det finnes mange forskjellige metoder for transformasjon mellom kontinuerlig og diskret tid. I vårt tilfelle brukte vi *Matlab* til simuleringer og transformasjoner. I Matlab finnes programpakken *Simulink*, dette er et grafisk simuleringsverktøy som tar i bruk blokkdiagram.

Det er ikke mulig å lage et filter som gir fullstendige stoppbånd. Et optimalt filter vil tilnærmes



Figur 7: Plassering av ideelt anti-alias filter, og den faktiske plassering av filteret i spillet.

bedre med høyere orden, men implementasjon blir desto vanskeligere. Det er derfor naturlig å implementere et første orden filter for å oppfylle kravene. Et første orden filter har én bestemt knekkfrekvens. Dette er frekvensen der filteret forsterker signalet -3dB, hvilket er en $\sim$ 70% reduksjon av det originale signalet.

Spillet antas å hente input på 60 Hz, hvilket gir en samplingstid på $\frac{1}{60}$s. Knekkfrekvensen i vinkelhastighet vil være

$$\omega_c = \frac{2 \cdot f_s}{20} = 6 \cdot \pi \text{ Rad/s} \approx 18,8 \text{ Rad/s}$$

Kommunikasjon mellom kontroller og datamaskin

For at kontrolleren, en mobil i dette tilfellet, skal kunne kommunisere med datamaskinen trengs et kommunikasjonsmedium. Her er Wi-Fi anvendt for kommunikasjon mellom enhetene. Kontrollerene sender informasjon de har til datamaskinen via en Internet protokoll som kalles for *user datagram protokoll* (UDP) [12]. En protokoll kan beskrives som regler for hvordan de ulike enhetene skal kommunisere. UDP er en protokoll som ikke har en garanti at all data som sendes vil mottas. Dette kan føre til mer effektiv kommunikasjon mellom ulike enheter, da man ikke må man vente til at begge enhetene blir enig om at en melding har blitt mottatt før en ny melding kan sendes. Ulempen er at hvis for mange meldinger ikke kommer frem kan spillet bli hakkete.

Kobling til fysikkmodulen

Inputmodulen får inn flyttall fra telefonen som blir filtrert med beskrevet karakteristikk. Uavhengig av om inputsignalet kommer fra taster eller mobiltelefoner, vil fysikkmodulen representere verdiene som en akslerasjon.

Fysikkmodul

For å modellere bevegelse på en realistisk måte, brukes fysikk. Modellen for bevegelse kan deles i to deler. En del er ansvarlig for å løse Newtons andre lov for å modellere bevegelsen utfra inputmodulen, i samspill med ulike modeller for fysiske felt. Den andre delen er ansvarlig for kollisjoner. Både kollisjon mellom spillerne, og kollisjon med en spiller og veggen. Fysikk-modulen blir brukt før hvert bilde blir oppdatert for å sikre at spillerne får så riktig som mulig bevegelse mellom hver oppdatering av bildet.

Uavhengig bevegelse

I klassisk mekanikk brukes Newtons lover for å beskrive klassisk beveglese. For å beskrive spillernes bevegelse i spillet vil Newtons andre lov bli tatt i bruk

$$\sum \vec{F} = m\vec{a}, \tag{5}$$

hvor $\vec{F}$ er en kraft, m er massen til objektet i bevegelse og $\vec{a} = \ddot{\vec{x}}$ er akslerasjonen til objektet. Kraften kan generelt være en funksjon av tid, posisjon, og den deriverte av posisjon. Denne oppgaven skal imidlertid begrense seg til krefter på formen $\vec{F} = \vec{F}(t, \dot{\vec{x}}, \ddot{\vec{x}})$. Dermed kan eq. (5) skrives om til

$$\sum \vec{F}(t, \vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}) = m\ddot{\vec{x}} \tag{6}$$

hvilket er en annen ordens differensial ligning i x. Målet for bevegelsesdelen av fysikkmodulen blir dermed å løse eq. (6).

Det finnes flere mulige måter å løse slike differensialligninger. Ved å skrive om eq. (6) til

$$\dot{\vec{x}} = \vec{f}(t, \vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}) = \frac{1}{m} \sum \vec{F}(t, \vec{x}, \dot{\vec{x}}, \ddot{\vec{x}}) \tag{7}$$

kan eq. (7) løses numerisk. Ved tiden $t = n \cdot \Delta t$ vil posisjonen være gitt av $x_n = \vec{x}(n \cdot \Delta t)$, og man får hjelp av Eulers metode

$$\begin{aligned} \vec{x}(0) &= \vec{x}_0, \qquad \dot{\vec{x}}(0) = \dot{\vec{x}}_0, \\ \dot{\vec{x}}_{i+1} &= \dot{\vec{x}}_n + \Delta t \cdot \vec{f}(t_n, \vec{x}_n) \\ \vec{x}_{i+1} &= \vec{x}_n + \Delta t \cdot \dot{\vec{x}}_{i+1} \end{aligned} \tag{8}$$

der $\vec{x}_0$ er start posisjonen og $\dot{\vec{x}}_0$ er starthastigheten. For et spill er naturlige initialverdier gitt ved en startposisjonen midt på brettet og ingen starthastighet.

Δt er i utgangspunktet uavhengig av spillet, og jo mindre Δt blir, jo mer nøyaktig blir differensialligningen løst [13]. Et naturlig valg er imidlertid å løse ligning eq. (8) med Δt tilnærmet bildefrekvensen til spillet, slik at spillerobjektet har en ny posisjon og fart hver gang bildet blir tegnet.

Grensebetingelser

Når spillobjektet beveger seg ut mot kanten av brettet vil det være naturlig å sette en grensebetingelse for randen av spillbrettet slik at spillerne ikke kan få en posisjon utenfor brettet. To naturlige valg er enten periodiske grensebetingelser eller å innføre en hard vegg med elastiske støt.

Modell for kollisjoner

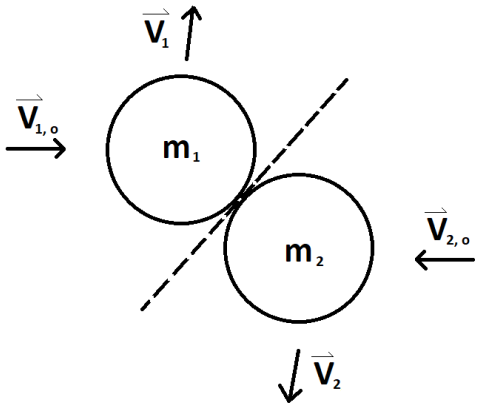
Det er to grunnleggende fysiske størrelser som må være bevart i støtet. [13] Kinetisk energi

$$E_k = \sum_i \frac{1}{2} m_i v_i^2, \tag{9}$$

og moment

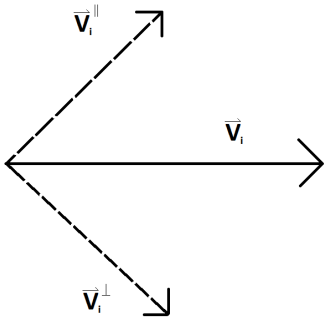
$$\vec{p} = \sum_i m_i \vec{v}_i. \tag{10}$$

En 2D kollisjon kan reduseres til et 1D problem ved å innføre en linje (plan i 3D) som står tangentielt på begge kollisjonsobjektene i kollisjonspunktet [13]. Dette er illustrert i Figur 8.



Figur 8

Hastighetsvektoren kan dekomponeres i en komponent parallelt til den tenkte linjen, $\vec{v}^{\parallel}$ og en komponent normalt til linjen, $\vec{v}^{\perp}$, hvilket er illustrert i Figur 9



Figur 9

Videre kan man anta at objektene ikke vil rotere i støtet. Det fører til at det ikke er krefter parallelt med den tenkte linjen, og som følge er $\vec{v}^{\parallel}$ bevart i støtet. Fra eq. (9) og eq. (10), der subskript _o står for initiell fart

$$\begin{aligned} m_1(\vec{v}^{\parallel}_{1,o} + \vec{v}^{\perp}_{1,o}) + m_2(\vec{v}^{\parallel}_{2,o} + \vec{v}^{\perp}_{2,o}) &= m_1(\vec{v}^{\parallel}_1 + \vec{v}^{\perp}_1) + m_2(\vec{v}^{\parallel}_2 + \vec{v}^{\perp}_2) \\ \implies \quad m_1\vec{v}^{\perp}_{1,o} + m_2\vec{v}^{\perp}_{2,o} &= m_1\vec{v}^{\perp}_1 + m_2\vec{v}^{\perp}_2 \end{aligned} \tag{11}$$

og kinetisk energi, der det blir brukt at $(\vec{v}^{\parallel} + \vec{v}^{\perp})^2 = |\vec{v}^{\parallel}|^2 + |\vec{v}^{\perp}|^2$ hvilket holder da komponentene er normale på hverandre

$$\begin{aligned} \frac{1}{2}m_1(\vec{v}^{\parallel}_{1,o} + \vec{v}^{\perp}_{1,o})^2 + \frac{1}{2}m_2(\vec{v}^{\parallel}_{2,o} + \vec{v}^{\perp}_{2,o})^2 &= \frac{1}{2}m_1(\vec{v}^{\parallel}_{1} + \vec{v}^{\perp}_{1})^2 + \frac{1}{2}m_2(\vec{v}^{\parallel}_{2} + \vec{v}^{\perp}_{2})^2 \\ \implies \frac{1}{2}m_1|\vec{v}^{\perp}_{1,o}|^2 + \frac{1}{2}m_2|\vec{v}^{\perp}_{2,o}|^2 &= \frac{1}{2}m_1|\vec{v}^{\perp}_{1}|^2 + \frac{1}{2}m_2|\vec{v}^{\perp}_{2}|^2 \end{aligned} \tag{12}$$

eq. (11) og eq. (12) danner dermed ligningsett for en effektiv 1D kollisjon. Løsningen av eq. (11) og eq. (12) krever en del algebra, men de kan man finne i f.eks i [13].

$$\begin{aligned} \vec{v}^{\perp}_{1} &= \frac{m_1 - m_2}{m_1 + m_2}\ \vec{v}^{\perp}_{1,o} + \frac{2m_2}{m_1 + m_2}\ \vec{v}^{\perp}_{2,o} \\ \vec{v}^{\perp}_{2} &= \frac{m_1}{m_1 + m_2}\ \vec{v}^{\perp}_{1,o} + \frac{2m_2}{m_2 - m_1}\ \vec{v}^{\perp}_{2,o} \end{aligned} \tag{13}$$

For å gjøre beregningene enklere kan man finne de originale 2D fartvektorene på en vinkelfri form (m.a.o., ikke dekomponert i en parallel og en normal komponent) fra følgende ligninger eq. (11)

$$\begin{aligned} \vec{v}_1 &= \vec{v}_{1,o} - \frac{2m_2}{m_1 + m_2}\frac{(\vec{v}_{1,o} - \vec{v}_{2,o}) \cdot (\vec{x}_{1,o} - \vec{x}_{2,o})}{||\vec{x}_{1,o} - \vec{x}_{2,o}||^2}\left(\vec{x}_{1,o} - \vec{x}_{2,o}\right) \\ \vec{v}_2 &= \vec{v}_{2,o} - \frac{2m_1}{m_1 + m_2}\frac{(\vec{v}_{2,o} - \vec{v}_{1,o}) \cdot (\vec{x}_{2,o} - \vec{x}_{1,o})}{||\vec{x}_{2,o} - \vec{x}_{1,o}||^2}\left(\vec{x}_{1,o} - \vec{x}_{2,o}\right) \end{aligned} \tag{14}$$

Dermed kan alle elastiske kollisjoner mellom spillobjekter beskrives fra ligning eq. (14). Det samme kan gjøres med veggen, i gensen $m_{vegg} \to \text{inf}$ og $\vec{v}_{vegg} = \vec{0}$. Det er imidlertid enklere å sette disse grensene inn i eq. (13). Fra L'Hôpitals regel får vi da at $\vec{v}^{\perp}_{\text{spillobjekt}} = -\vec{v}^{\perp}_{\text{spillobjekt,o}}$ og $\vec{v}^{\parallel}_{\text{spillobjekt}} = \vec{v}^{\parallel}_{\text{spillobjekt,o}}$. Man må altså bevare komponenten parallelt med veggen, og snu fortegn på komponenten normalt på veggen.

Modell for krefter

Fra input modulen hentes det ut en gitt akslerasjon for alle spill objektene $\vec{a}_i$. Denne akslerasjonen blir antatt konstant for hvert tidsintervall Δt, og ved å multiplisere m_i har man en konstant kraft for hvert Δt. Dette blir implementert på følgende måte.

```
for player in self.players:
    if player.shield():
        continue

    # Force from input
    force = PhysicsConsts.input_modulation * player.input.get_move()
    # Force from arenaLayers
    force += self.arena.force(player, force, PhysicsConsts.input_modulation, dt)

    # Eulers method to update velocity and position
    acceleration = force / player.mass
    player.velocity += acceleration * dt
    player.position += player.velocity * dt
```

Fysikkmotoren er altså avhengig av å hente input fra to andre moduler, `Input` via `Player` modulen og `Arena` modulen. Detaljer om inputmodulen er beskrevet i seksjon Inputmodul. `Arena` modulen har en klasse `Arena`, som inneholder en liste av objekter av klassen `ArenaLayer`. Videre blir det definert to klasser som arver fra ArenaLayers, `FrictionLayer` og `AirResistanceLayer`, som begge har funksjonen `force()` som returnerer henholdsvis friksjons- og luftmotstandskreftene.

Friksjonskreften blir modellert som en konstant μ, som avhenger av overflaten multiplisert med spillobjektet sin normalkraft $N_{=}m_i g$, der g er tyngdeakselerasjonen

$$\vec{F}_f = -\mu m g \cdot \hat{v} \tag{15}$$

Det er to typer friksjon, statisk og kinetisk friksjon. Statisk friksjon oppstår når det ikke er bevegelse, mens kinetisk friksjon oppstår når to overflater beveger seg relativt til hverandre. I spillet blir dette implementert slik at man må ha en input kraft større enn den maksimale statiske friksjonen for å starte bevegelsen. Når spillobjekter beveger seg får objektet en kraft kun avhengig av underlaget. Hvis farten kommer under en terskelverdi blir farten satt til 0, og kraften blir satt til null. Dette er en modell for den statiske friksjonen.

Luftmotstanden avhenger av en konstant K_d multiplisert med farten kvadrert

$$\vec{F}_d = -K_d \, |\vec{v}|^2 \cdot \hat{v} \tag{16}$$

ved å kombinere eq. (15) og eq. (16) har man et uttrykk for kraften som hindrer bevegelsen.

Både firksjonskoeffisienten og luftmotstanden kan varieres. Ved å variere konstantene vil det være en lik endring over hele brettet. Begge konstantene er med andre ord ikke posisjonsavhengige. En mulig forbedring av spillet kunne vært å gjøre konstantene posisjonsavhengige, slik at det for eksempel blir glattere på enkelte steder av brettet.

Statistikkmodul

Motivasjon

De fleste konkurransedrevende spill har funksjonalitet for å rangere spillere. Dette kreves for å implementere visse spillmekanismer:

- *Poengtavle* - Poengoversikt for alle historiske spillere.
- *Kampoppsett* - Rettferdig oppsett av motstandere fra en populasjon.
- *Vinnersannsynlighet* - Prediksjon av utfallet til en gitt kampkonfigurasjon.

En enkel løsning for dette problemet er *poengakkumulering*. Hvis spillere har muligheten til uavhengig å høste poeng i løpet av en spilløkt, vil det være mulig å rangere spillere basert på totalt innhøstede poeng over alle historiske spilløkter. Et slikt system vil være begrenset til spilltyper hvor poenghøsting faller seg naturlig. I tillegg vektlegger et slikt poengsystem spillere som har spilt mer enn andre spillere, ikke nødvendigvis spillernes underliggende ferdigheter. På grunn av dette så er poengakkumulering ofte utilstrekkelig for de fleste formål.

En mer generell tilnærming må til. Ved å fokusere på konkurranseutsatte spill med to deltakere, deltaker i og j, kan resultatet av en fullført spilløkt, $\vec{s}$, representeres som:

$$\vec{s} = (i, j, u) \qquad \Longleftrightarrow \qquad u := \begin{cases} 0, & \text{uavgjort} \\ 1, & \text{deltaker i vant} \\ -1, & \text{deltaker j vant} \end{cases} \tag{17}$$

Spillet er derfor begrenset til en-mot-en spillmekanismer som resulterer i enten én vinner eller uavgjort. Sjakk er et eksempel på et slikt spill. Man kan nå arrangere n sekvensielle spilløkter og registrere utfallet av alle spilløktene i matrisen S:

$$S = \begin{bmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vdots \\ \vec{s}_n \end{bmatrix} = \begin{bmatrix} i_{11} & j_{12} & u_1 \\ i_{21} & j_{22} & u_2 \\ \vdots & \vdots & \vdots \\ i_{n1} & j_{n2} & u_n \end{bmatrix}, \qquad \forall k \in \{1, 2, ..., n\} : \; i_{k1} \neq j_{k2} \tag{18}$$

Antall deltakere D tilfredsstiller $2 \leq D \leq 2n$ siden spillere ikke kan spille mot seg selv.

Modell

Arpad Elo utviklet rangeringssystemet *Elo* for sjakk i 1959, som ble tatt i bruk av verdens sjakkforbund (FIDE) i 1970 [4]. Det er en generalisering av Elo som har blitt implementert i spillet, så denne rangeringsmodellen vil nå bli forklart.

Elo-systemet tilderer rangeringer r_i til hver deltaker i. Jo høyere rangeringen, jo bedre er den respektive spilleren. Når to deltakere, i og j, møtes i en sjakkamp, antar Elo at hver deltaker presterer uavhengig av hverandre. Prestasjonene angis som p_i og p_j og modelleres som realiseringer fra to uavhengige normaldistribusjoner med lik varians β^2 og ulike forventningsverdier r_i og r_j, med andre ord deres respektive rangeringer [10], gitt av

$$\begin{aligned} p_i &\sim \mathcal{N}\left(r_i, \beta^2\right) \\ p_j &\sim \mathcal{N}\left(r_j, \beta^2\right) . \end{aligned} \tag{19}$$

Ved å definere $\Delta_{ij} := p_i - p_j$, indikerer $\Delta_{ij} > 0$ at deltaker i vinner mens $\Delta_{ij} < 0$ indikerer at spiller j vinner. Dette er en lineær kombinasjon av to uavhengige normalfordelte variable med

$$\begin{aligned} \mathrm{Var}(\Delta_{ij}) &= \mathrm{Var}(p_i - p_j) = \mathrm{Var}(p_i) + \mathrm{Var}(p_j) = 2\beta^2 \\ \mathrm{E}[\Delta_{ij}] &= \mathrm{E}[p_i - p_j] = \mathrm{E}[p_i] - \mathrm{E}[p_j] = r_i - r_j \end{aligned}$$

Δ_{ij} er derfor *også* normalfordelt med følgende parametrisering

$$\Delta_{ij} \sim \mathcal{N}(r_i - r_j, 2\beta^2). \tag{20}$$

Under denne tolkningen så kan utfallsvariabelen u, definert i ligning (17), for en kamp mellom spiller i og j omdefineres som

$$u := \mathrm{sgn}\ \Delta_{ij},$$

hvor $\mathrm{sgn}(x)$ er *signumfunksjonen* eller *fortegnsfunksjonen*. Et uttrykk for sannsynlighet for at spiller i vinner mot spiller k kan gis ved

$$P(u = 1 \mid i, j) = P(p_i > p_j \mid r_i, r_j) = P(\Delta_{ij} > 0 \mid r_i, r_j) = \Phi\left(\frac{r_1 - r_2}{\sqrt{2}\beta}\right) ,$$

hvor $\Phi(z) = P(Z \leq z)$ er den kumulative fordelingsfunksjonen for standard normalfordelingen $\mathcal{N}(0, 1)$.

Estimasjon av rangeringer

Man ønsker å finne gode tilnærminger for de ukjente rangeringene r til spillerne som har deltatt i tidligere spill, altså utelukkende med informasjonen i S. Rangeringene r tillater arbitrær translasjon og skalering, og man er også frie til å vektlegge nye observasjoner som man selv ønsker. Elo-systemet er arvtakeren til Harkness-systemet [6] og derfor, av historiske årsaker, er mange av disse parametriseringene valgt for å overenstemme med Harkness-systemet i så stor grad som mulig. Elo-systemet starter derfor med å tildele rangering 1500 til alle deltakere,

$$r_i^{(0)} \leftarrow 1500\ .$$

Etter at spiller i og j har spilt, ønsker man å oppdatere rangeringene $r_i^{(k)}$ og $r_j^{(k)}$ til $r_i^{(k+1)}$ og $r_j^{(k+1)}$ under restriksjonen

$$r_i^{(k)} + r_j^{(k)} = r_i^{(k+1)} + r_j^{(k+1)}.$$

Dette forhindrer inflasjon og deflasjon i rangeringene over tid. Summen av rangeringene til deltakermassen er derfor til enhver tid konstant, $1500 \cdot D$. Forventningsverdien til rangeringene blir $\mathrm{E}[r] = 1500$. Det er her antatt at begge spillere har deltatt k tidligere spilløkter for å forenkle notasjonen.

Videre ønsker man å finne en oppdatering $-1 < \delta < 1$ slik at

$$\begin{aligned} r_i^{(k+1)} &\leftarrow r_i^{(k)} + u \cdot \delta \\ r_j^{(k+1)} &\leftarrow r_j^{(k)} - u \cdot \delta\ , \end{aligned}$$

hvor det observerte utfallet u er *mer* sannsynlig under de nye rangeringene. Elo-systemet bruker oppdateringen

$$\delta = K\left(\frac{u+1}{2} - \Phi\left(\frac{r_i^{(k)} - r_j^{(k)}}{\sqrt{2}\beta}\right)\right) ,$$

hvor K, den såkalte K-faktoren, er gitt ved $K := \alpha\beta\sqrt{\pi}$, der $0 < \alpha < 1$ er vektingen av den nye observasjonen i forhold til det tidligere rangeringsestimatet. For å få en intuisjon for denne oppdateringen, kan følgende eksempler settes opp:

- Spiller i `vinner` mot en identisk god spiller j, altså $u = 1$ Oppdateringen blir derfor fastsatt til $\delta = K(2/2 - \Phi(0)) = K(1 - 0.5) = K/2$. I omvendt tilfelle, $u = -1$, blir $\delta = -K/2$, ser man forventet symmetri.
- En ekstremt god spiller i `vinner` mot en ekstremt dårlig spiller j. Man har derfor $r_i^{(k)} - r_j^{(k)} \approx \infty$ og $u = 1$. Derfor får man $\delta \approx K(2/2 - \Phi(\infty)) = K(1 - 1) = 0$. Den nye observasjonen gir lite ny informasjon om spillerne som reflekteres i den neglisjerbare oppdateringen.
- Den ekstremt gode spilleren i `taper` mot den dårlige spilleren j. Denne gangen er derfor $u = -1$. Vi får derfor $\delta \approx K(0/2 - \Phi(\infty)) = K(0 - 1) = -K$. K-faktoren gir oss en øvre grense på hvor mye spillere kan rykke opp i rangering når de vinner mot en mye bedre spiller, med tilsvarende nedrykk for den gode spilleren.

K-faktoren bør reflektere i hvilken grad tilfeldig støy (flaks) bidrar til spillets utfall. Verdens sjakkforbund setter for øyeblikket $K = 10$ for erfarne spillere og gradvis mindre K for nyere spillere [5].

Generalisering til lagspill

Siden spillet utviklet i dette prosjektet kan utvides til å støtte lagspill, ble det implementert en rangeringsmetode som legger til rette for en eventuell lagspillutvidelse i fremtiden.

TrueSkill™ er et Bayes-analytisk rangeringssystem utviklet av Microsoft for å håndtere spill med arbitræreb lagoppsett [10], som en generalisering av Elo hvilket utelukkende håndterer en-mot-en spill. En implementasjon av algoritmen har blitt publisert av Microsoft i Python-pakken `trueskill` [9], og denne har blitt implementert som en del av rangeringssystemet i spillet.

Deltakerne antas fortsatt å prestere etter respektive normalfordelinger $p_i \sim \mathcal{N}(r_i, \beta^2)$, men nå tildeles en *a priori* fellesdistribusjon.

$$p(\vec{r}) := \prod_{i=1}^{D} \mathcal{N}(r_i; \mu_i, \sigma^2)$$

for rangeringene. Deltakerne deles så inn i k ikke-overlappende lagfordelinger $A := \{A_1, ..., A_k\}$. Spillets utfall er en rangering av disse lagene $\vec{u} = (u_1, ..., u_k) \in \{1, ..., k\}$. Hvis $u_i = u_k$ har lag i og j gjort det like godt og hvis $u_i > u_k$ så har lag i prestert bedre enn lag j.

Lagnummer j sin totale prestasjon t_j anses som en sum av alle lagmedlemmenes individuelle prestasjoner

$$t_j := \sum_{i \in A_j} p_i \ .$$

Bayes regel gir oss nå *posteriori* distribusjonen for rangeringene

$$p(r_1, r_2, ..., r_D \mid \vec{u}, A) = \frac{P(r_1, r_2, ..., r_D \mid \vec{u}, A) \cdot p(\vec{r})}{P(\vec{u} \mid A)} \ .$$

Estimasjon av *individuelle* rangeringer r_i ved hjelp av *lagrangeringene* er en formidabel utfordring som ikke blir gått inn på her. Det kan nevnes at TrueSkill estimerer rangeringene ved hjelp av et såkalt *gaussisk tetthetsfilter* [11].

Resultat og diskusjon

Resultatet av prosjektet ble et spill hvor to spillere, ved å bevege seg igjennom fysiske kraftfelt, forsøker å dytte hverandre utenfor en ring. Grensesnittet er vist i figur 1. Ved oppstart av spillet møter man en startskjerm der spillerne kan angi navn, og starte spillet ved å trykke på `Start!`-knappen, som vist i Figur 2a). For å øke brukervennligheten kan spillet styres enten via piltastene på tastaturet eller en mobiltelefon, koplet opp mot datamaskinen. Spilleren som først mister sine helsepoeng har tapt runden. Når en runde er ferdig vises ulike spillstatistikker, med mulighet for å starte et nytt spill ved å klikke på `retry`-knappen. Grensesnittet for avslutningsskjermen er vist i Figur 2b. Egen og motstanders hastighet og akselerasjon blir vist i sanntid under kjøring av spillet, som vist i Figur 1. I spillet er det mulig å justere luftmotstanden, friksjon og massene på spillobjektene. Både sanntidsfremvisningen av akselerasjon og hastighet, og justeringen av luftmotstand, friksjon og masse kan bidra til økt forståelse av fysikk. Det gir en visuell karakter til et ellers teoretisk fag, der spillerene kan se hvordan spillobjektet påvirkes av, og oppfører seg, under ulike betingelser.

Koden er lansert som åpen kildekode på GitHub repoet `jakobgm/eit-game`. [1]

Sammenlignet med kommersielle spill framstår spillet svært enkelt, både visuelt og funksjonalitetsmessig. Dette skyldes antageligvis både gruppemedlemmenes manglende erfaring med spillutvikling, og manglende tid. For å utvikle et ferdig spillprodukt kreves det ofte flere år med utvikling [2]. Samtidig bidrar enkelheten til å utvide spillets bruksområder, da det kan fungere som et templat andre kan arbeide utfra og utforme etter egne behov. For samtlige gruppemedlemmer har læringskruven vært bratt, og alle har måtte utvide sitt syn på hva deres fagområde kunne bidra i spillutviklingen. Spesielt for gruppemedlemmene med fagbakgrunn utenfor datateknologi ble det tidlig klart at fokuset og tankemåten i spillutviklingen måtte ligge på brukervennligheten fremfor tradisjonelle fremgangsmåter innen de representerte fagområdene. For å få en naturlig oppførsel i spillet må størrelsen av valgte parametere ofte begrenses, mens man blant annet innenfor fysikk ofte arbeider med store parametre. Eksempler på dette er at tidssteget i fysikkmodulen måtte forkortes, eller at antall krav for foldingsfilteret i inputmodulen måtte reduseres, for å minimere beregninger og unngå at spillet ble for tregt.

Det ble tidlig klart at Python ikke var et ideelt programmeringsspråk rent ytelsesmessig. Et mål på ytelsen til programmet ble satt til bildefrekvensen, altså hvor ofte programmet overskrev bildet på skjermen. En uoffisiell standard for spill er ofte en tid rundt $1/60 \approx 0.017$ s. Ved å kjøre spillet på en beregningsorientert PC med en Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz prosessor, med 16 GB ram, brukte spillet i løpet av 100 bildeoppdateringer i snitt 0.035 s. Visuelt er bevegelsen fremdeles realistisk, uten forsinkelser eller lagging". Imidlertid er det bekymringsverdig at spillet allerede bruker dobbelt så lang tid som ønskelig, spesielt med tanke på senere implementasjon av ny funksjonalitet, eller ved kjøring på svakere datamaskiner.

En løsning kunne vært å benytte et kompilert programmeringsspråk, som C eller C++. Kompilerte programmeringsspråk krever på den andre siden oftest mer arbeid i utviklingsprosessen, hvilket gjør valg av programmeringsspråk til en vurderingssak. Mange av beregningene kunne vært flyttet fra prosessoren over til grafikkkortet. I utgangspunktet ville det krevd en innføring i nye programmeringsspråk, men det skal ikke utelukkes at det finnes rammeverk som gjør bruken av grafikkkort langt enklere å implementere. Hvis ytelsen hadde vært hovedfokus hadde det gått på bekostning av samfunnsnytten for eksempel på grunn av mangel på funksjonalitet.

Spillet er ikke brukertestet. Under utvikling ble spillet hyppig testet av gruppemedlemmene, men samtidig vet gruppemedlemene hvordan produktet er bygd opp og møter dermed ikke de samme hindringene som en vanlig bruker møter. En uavhengig part kunne potensielt funnet mange svakheter som ikke var observert, og kommet med nyttige tilbakemeldinger både til spillets funksjon,

nyttighet og brukervennlighet. Slik spillet er utviklet er det ingen klart definerte brukergrupper, og det hadde derfor vært ønskelig å teste spillet i ulike brukergrupper, for å utforske hvor spillet har størst potensialet for å bli brukt.

Spillets samfunnsnytte ligger i å øke forståelse for fysikk gjennom et interaktiv og visualiserende spill. I tillegg ønsker gruppen å øke interessen, kompetansen og forståelsen for realfag og teknologi gjennom spillet. Spillet er først og fremst rettet mot fysikk, men det er lagt inn mulighet for å legge til egne fysikkelementer i spillet som gir muligheten for folk å øke sin forståelse for programmering ved å legge til nye fysikkelementer selv.

For å få modulene til å fungere sammen, var det nødvendig med god struktur og godt dokumentert kode. En naturlig følge av at gruppemedlemmene representerer ulike fagområder er at spillets kode ble skrevet slik at den ikke var forståelig for alle gruppemedlemmene. Ikke fordi programmeringsspråket var ukjent, men fordi det faglige innholdet etterhvert ble avansert. Selv om det ikke var nødvendig for alle medlemmene å forstå alle detaljer i den fagspesifikke koden, var alle avhengige av å forstå tilstrekkelig til at de kunne anvende resultatene og bidra til å kople de ulike modulene sammen til det helhetlige spillet. Gruppen var derfor avhengig av at samtlige gruppemedlemmer presenterte de nødvendige og relevante aspektene ved sitt fagfelt og sin del av prosjektet på en slik måte at det ble forståelig for gruppen som helhet.

Til tross for flere negative bemerkninger rundt spillet, må resultatet bedømmes ut fra tid og ressurser til disposisjon. Brukertesting ville forbedret brukeropplevelsen, men å skaffe frivillige brukere i riktig brukersegment ble ikke prioritert. Dette ville kostet tid, som kunne gått på bekosting av implementasjon av annen funksjonalitet. Ytelsen ble kritisert, men spillet fungerer akseptabel på en god nok datamaskin. Gruppen som helhet er fornøyd med hvordan de har arbeidet for å få alles kompetanse og faglige bakgrunn representert i det ferdige produktet. Selv om kodestandarden kunne vært forbedret, er koden godt kommentert slik at det skal være lett for andre å ta over prosjektet. Dette er en god egenskap ved koden, for ofte bidrar manglen på kommentarer i kode til å gjøre det vanskelig for utenforstående personer å bidra til utvikling. Spillets kode er åpen kildekode og gode kommentarer i koden er da en forutsetning for effektiv videreutvikling.

Tverrfaglige utfordringer

Hovedsaklig brukte gruppen sin tverrfaglighet gjennom utvikling av modulene som beskrevet i Tabell 1. Disse ble deretter koblet sammen for å danne et helhetlig spill. En slik prosess fører med seg mye koordinering og planlegging for at uavhengige komponenter skal fungere sammen, og det har gjort av man ofte må utvide synet sitt på egne faglige bidrag. Etter hvert ble spillkoden for avansert til at alle gruppemedlemmene kunne forstå alle detaljene, samtidig som at alle var avhengige av å kunne forstå og bruke resultatene i sitt videre arbeidet. Av dette følger det at prosjektet var avhengig av at hvert gruppemedlem måtte formidle sin del av arbeidet på en måte som det tverrfaglige miljøet kunne forstå. Dermed ble det raskt tydelig at prosjektet ble en overgang fra å arbeide i et homogent fagmiljø, til å anvende sin spisskompetanse innenfor et fagfelt som en del av større prosjekt med et heterogent fagmiljø. Det vil her følge en kort oversikt over hvordan gruppemedlemmene utvidet sitt perspektiv på fagkunnskap gjennom det tverrfaglig arbeidet.

Fra en datateknikk-students perspektiv var det nytt å jobbe i et tverrfaglig gruppe med fokus på sanntidsprogrammering. Som oftest jobber man i en gruppe med personer med omtrentlig samme faglig bakgrunn, og alle prosjekter gjennom studiet har vært sammen med personer som går på samme studieprogram. Selv om spisskompetansen er ulik, er det likt nok til at det å jobbe

i denne EiT-gruppen har vært en ny opplevelse, og økt forståelsen for å jobbe i et tverrfaglig team. Det er alltid viktig å ha god ytelse på programmer man skriver, men med et spill ble det tydelig at det var ekstra viktig at ytelsen var god slik at brukeopplevelsen ble bra. Uten god ytelse ville ikke spillet vært brukervennlig. Programmene man lager gjennom studiet er ofte ikke veldig store, men gjennom EiT, har student erfart og lært hvordan man bygger opp et komplekst program som krever mye samhandling mellom ulike moduler som andre fra andre fagfelter lager.

Kybernetikk-studenten opplevde at det i begynnelsen var vanskelig å kommunisere sitt fagområde. I tidligere prosjekt har grunnprinsipper innenfor kontroll-teori, en basal del av kybernetikkfagfeltet, vært kjent for alle gruppemedlemmene. Kontrasten fra å arbeide i en gruppe der alle medlemmene har god innsikt i og forståelse for de grunnleggende begrepene innenfor kybernetikk til et tverrfaglig gruppe, ble i begynnelsen opplevd stor. Å konstruere abstrakte eksempler på en slik måte at gruppemedlemmene enkelt kunne forstå og relatere seg til, var en utfordring. I likhet med datastudenten opplevde kybernetikk-studenten kontrasten mellom et homogent fagmiljø og tverrfaglig team som stor. Abstraheringen av problemer var ikke bare nyttig for gruppen generelt, men gjorde det nødvendig for kybernetikk-studenten å tenke på nye måter rundt problematikken, hvilket bidro til økt forståele av eget fag generelt, og abstrahering spesielt.

Kompetanse fra Industriell økonomi og teknologiledelse har bidratt som en hybrid innen datateknologi, gruppearbeid, statistikk og matematikk generelt. Til tross for at dette kan relateres til de ulike kompetansene representert, mangler fagretningen en klar nisje i gruppen. Dette har ført til faglige bidrag i flere moduler, men når det faglige nivået har blitt for høyt har det blitt naturlig å gå over på andre deler av prosjektet, og overlate arbeidet til den på gruppen med spisskompetanse på området. Til tross for at det hadde vært ønskelig å bidra enda mer til det faglige, er det samtidig viktig å jobbe med helheten i prosjektet, og å knytte modulene sammen.

Fra fysikkstudentens perspektiv var det å fokusere på brukervennlighet et nytt perspektiv på anvendelsen av eget fagområde. I fysikken er man ofte opptatt av å finne et tallsvar eller en underliggende sammenheng mellom størrelser. Det var derfor uvant å produsere en faglig rapport uten grafer, der fokus heller var knyttet opp mot å lage et helhetlig produkt. Resultatfokuset gjorde også at valg av parametere måtte velges bassert på best mulig brukervennlighet. Bland annet er størrelsen på spillbrettet gitt ved et antall piksler, ikke i SI enheter. Kreftene i spillet ble derfor funnet ved å multiplisere inputparameterne (av lenge en) med en modulasjonskonstant som som ble bestemt utfra hva som så naturlig ut, ved observasjon av spillet. Det hadde vært mulig å lage en regel for å knytte piksler til SI enheter, men det ville antagelig ikke gjort spillet bedre. Denne måten å tenke på var ny for fysikkstudenten, men det illustrerer samtidig hvordan fagfelt kan kobles opp på nye måter, der tradisjonelle fremgangsmåter noen ganger må vike plass.

Statistikeren i gruppen har innsett at bruksområdene til statistikk ikke nødvendigvis er tydelig for alle medlemmer i et tverrfaglig team. For mange er statistikk utelukkende et verktøy for å behandle data, men statistikk kan også brukes til å modellere en lang rekke med fenomener og fatte beslutninger om *hvilke* data som kan være av betydning og derfor bør tas vare på. Det å tidlig dele statistiske innfallsvinkler til en problemstilling har derfor vist seg å være viktig. Det har også vært lærerikt å benytte kunnskap innen numerisk matematikk sammen med en fysiker, da disse to fagfeltene er tett knyttet sammen.

Gruppens tverrfaglige utfordringer må sees i lys av at gruppemedlemmene kommer fra relativt like fagområder. Samtlige gruppemedlemmer er sivilingeniørstudenter, der studieløpene er relativt like de to første årene. Gjennom god planlegging og tidlig fokus på struktur og god gruppedynamikk, ble mange fallgruver unngått. Dermed endte prosjektet opp med relativt få tverrfaglige utfordringer. I stedet for å fokusere på tverrfaglige problemer har gruppemedlemmene benyttet sjansen til å lære fra hverandre og betrakte eget fagområde fra nye perspektiver. Deltakerne sitter dermed igjen med nyttig kunnskap både innen nye fagområder, samt med kunnskap om hvordan

man bedre kan integrere bidrag fra sitt fagområde inn i et tverrfaglig prosjekt.

Videre arbeid

Prosjektet kan betraktes som et innledende prosjekt, med potensiale for videreføring i flere retninger. Hovedpunktene der spillet har forbedringspunkter, med tilhørende fagområder, kan oppsummeres i Tabell 3. Videre blir de viktigste forbedringsområdene sett fra hvert fagområde oppsummert i hvert sitt avsnitt.

Tabell 3: Oversikt over forbedringpunkter i spillet med tilhørende fagområde.

Forbedringspunkt	**Fagområde**
Interaksjonsdesign	Datateknologi
Kunstig intelligens	Datateknologi
Flerspiller-funksjonalitet over Internet	Datateknologi
Forbedre ytelse vha. lavnivå programmeringsspråk	Datateknologi
Bruke eksternt grafikkort	Datateknologi
Nye fysiske krefter	Fysikk
Varierende krefter på banen	Fysikk
Egenprogrammerte styringsenheter	Kybernetikk
Videreutvikling av statistikkmodul	Statistikk

Datateknologi

Det visuelle aspektet ved spillet har fått lite oppmerksomhet, da fokuset primært har vært rettet mot å få de strukturelle rammene på plass. Økt brukervennlig, komplekse visualiseringer og sammenhenger i grafikken ansees som et av de større forbedringspotensialene i spillet. En naturlig utvidelse av prosjektet ville vært å involvere en fagperson med kompetanse innen interaksjonsdesign. Mangel på slik fagkompetansen ble en begrensende faktor, som førte til at dette området ikke ble prioritert. Samtidig kan det argumenteres for at spillets enkle grensesnitt gjør at spillet fungerer til nettopp det spillet er ment å fungere som - et undervisningssupplement. Ved å ha et minimalistisk design og enkelt brukersnitt, unngås forstyrrende elementer og brukerens fokus ligger på spillets innhold og funksjon som et undervisningssupplement.

Et tidlig mål i prosessen var å implementere komponenter innenfor kunstig intelligens, som blant annet hadde krevd å kartlegge hvordan en slik datastyrt enhet skulle fungert, og hvilke teknikker som ville ha fungert. Dessverre er det ofte vanskelig å få et slikt system til å fungere bra uten å investere mye tid, og med målet om å dekke alle gruppemedlemmenes fagområder, ble planen om kunstig intelligens lagt til side til fordel for økt funksjonalitet i resten av arbeidet. Feltet *reinforcement learning*, som dreier seg om å lage spillere med kunstig intelligens, har eksistert lenge. Her har det vært store gjennombruddet de siste årene takket være mer datakraft, og det er et område det vil være naturlig å videreføre spillet innenfor. Ofte foretar disse spillerne valg i spillet basert på en innebygd nyttefunksjon, for eksempel en funksjon av din og motstanderens helsepoeng.

Spillet bruker allerede kompetanse innen telematikk med kommunikasjon av kontroller og datamaskin over Wi-Fi. En mulig videreføring av produktet er å implementere flerspillerfunksjonalitet (multiplayer mode) over Internet. På sikt håpes det at spillet kan spilles via en online-plattform slik at det appellerer til et større publikum.

Kybernetikk

HyperIMU lar brukeren koble opp mobiltelefonen sin som styringsenhet i spillet. Denne applikasjonen benytter et filter som de har laget selv for å skille ut støy fra bevegelsene til mobiltelefonen. Denne applikasjonen virker ikke alltid som ønsket, og en mulighet ville vært å designet programvaren fra bunnen av. Imidlertid anses dette holdepunktet for å være relativt tidkrevende med begrenset gevinst.

Fysikk

Det legges til varierende krefter på banen på spillet, som for eksempel ulik friksjonskoefissient på ulike deler av banen. Slik spillet er implementert er friksjonskoeffisienten lagret som en matrise med verdien en i all punkt. En naturlig utvidelse av fysikkmodulen er derfor å lage funksjonalitet som kan endre tallene på denne matrisen. Når tallene endres er det allerede implementert funksjonalitet for å finne friksjonen, og resultatet ville vært at man beveger seg med ulik hastighet på forskjellige deler av brettet. Fysikkmodulen er laget slik at det er lett å legge til nye fysiske krefter. Tanker er at en av veiene videre i forhold til fysikk kan være å legge til nye krefter som magnetisme og elektrisitet. Det kan også være interessant å legge inn vind i spillet.

Statistikk

Når det gjelder statistikk hadde det vært interessant å utvikle en statistisk modell som prøver å modellere hvordan forskjellige spillere utvikler seg over tid, og som kan si hvordan ferdigheten på ulike spillere er i forhold til hverandre. Hvis spillet blir populært nok er det også mulig å sjekke om spillerprestasjoner faktisk er normalfordelt, eller om det er en annen fordeling.

Konklusjon

Med et mål om å kombinere landsbytemaet *matematikk i anvendelser*, utnytte gruppemedlemmenes tverrfaglighet og å holde oppgaven innenfor et samfunnsrelevant tema, ble det utviklet et undervisningsorientert spill som går ut på å bevege et spillobjekt gjennom fysiske kraftfelt.

Matematikken har vært en sentral byggestein i spillutviklingen. Det matematiske fokuset for oppgaven har blant annet vært signal-prosessering, kø-teori, numerisk løsning av fysiske differensialligninger og Bayesiansk statistikk.

Hovedfokuset har vært på utvikling av spillkoden. Ved å introdusere moduler for hvert fagfelt, som senere ble knyttet sammen til den helhetlige spillkoden, fikk alle fagområdene en naturlig plass i kodebasen.

Intensjonen med spillet har vært å illustrere enkelte fysiske lover, på en måte som kan bidra til økt forståelse av fysikk. Ved å variere fysiske parametere, kan brukeren visualisere hvordan ulike parametre virker inn på vanlig bevegelse. Spillet gir en praktisk innfallsvinkel til undervisning av teoretiske begreper innen fysikk. Da spillets kode er åpen kildekode, er det muligheter for brukere til å videreutvikle spillet slik at det passer deres behov. Spillet har dermed et undervisningspotensiale både innenfor fysikk og innenfor programmering - hvilket er hva regjeringen ønsker for den norske skolen i fremtiden.

Til tross for at det presenteres et fungerende spill, er spillet både visuelt og funksjonalitetsmessig svært enkelt. Intensjonen har imidlertid ikke vært å konkurrere mot kommersielle spill, hvilket ville vært urealistisk med tanke på kunnskap og tid. Fokuset har heller vært å bygge opp et helhetlig produkt fra grunnen av, hvor mye krefter har gått med på å få de tverrfaglige modulene til å fungere sammen til et helhetlig produkt.

Videre kan spillets enkelhet åpne for videreføring og videreutvikling av spillet. Ytelsesmessig er allerede spillet tregere enn en typisk spillstandard, og det er foreslått å enten skrive om kodebasen til et kompilert språk, eller utnytte grafikkort for raskere beregninger. Andre måter å videreføre arbeidet kan være å implementere en mulighet for lagspilling, og å implementere en motspiller basert på kunstig intelligens.

Bibliografi

[1] Y. Bekkemoen mfl. *A game developed for TMA4850: Experts in Teams: Applied Mathematics*. URL: https://github.com/JakobGM/eit-game (sjekket 22.04.2019).

[2] Erik Bethke. «Game Development and Production». I: Wordware Publishing, Inc., 2003, s. 21.

[3] NumPy developers. *NumPy*. 2019. URL: https://www.numpy.org/index.html (sjekket 10.04.2019).

[4] Arpad E. Elo. *The rating of chessplayers, past and present*. New York: Arco Pub., 1978. ISBN: 0668047216 9780668047210. URL: http://www.amazon.com/Rating-Chess-Players-Past-Present/dp/0668047216.

[5] World Chess Federation (FIDE). *FIDE Chess Rating calculators: Chess Rating change calculator*. URL: https://ratings.fide.com/calculator_rtd.phtml (sjekket 22.04.2019).

[6] Kenneth Harkness. *The Official Chess Handbook*. McKay, 1967.

[7] J. D. Hunter. «Matplotlib: A 2D graphics environment». I: *Computing In Science & Engineering* 9.3 (2007), s. 90–95. DOI: 10.1109/MCSE.2007.55.

[8] Kunnskapsdepartementet. *Realfag for framtida*. URL: https://www.regjeringen.no/globalassets/upload/kd/realfagstrategi.pdf (sjekket 22.04.2019).

[9] Microsoft. *TrueSkill v0.4.5 - PyPi*. URL: https://pypi.org/project/trueskill (sjekket 22.04.2019).

[10] R. Herbrich T. Minka og T. Graepel. «TrueSkill(TM): A Bayesian Skill Rating System». I: MIT Press, jan. 2007, s. 569–576. URL: https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system/.

[11] T. Minka. «A family of algorithms for approximate Bayesian inference». PhD thesis. 2001.

[12] J. Postel. *User Datagram Protocol*. United States, 1980.

[13] H. D Young og R. A. Freedman. «University Physics with Modern Physics». I: 11th. Pearson, 2004, s. 300–306.