Integrasjon av Wolfram|Alpha i Unity3D "Weapons of Math Instructions"

en bacheloroppgave for SIM Subsea ved Høyskolen i Bergen

Eivind Askeland
Thomas Rossebø Hansen
Simen de Lange

10. juni 2013

HØGSKOLEN I BERGEN

Avdeling for Ingeniørutdanning
Institutt for Data- og Realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Integrasjon av Wolfram Alpha i Unity 3D	*Dato:* Vår 2013 *Rapportnummer:*
Forfatter(e): Eivind Askeland Thomas Rossebø Hansen Simen De Lange	*Antall sider u/vedlegg:* 52 *Antall sider vedlegg:* 3
Studieretning: Dataingeniør	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Harald Soleim, Jon-Eivind Vatne	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Sim Subsea	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Pål Solberg Trefall	*Telefon:* 93 47 97 84

Sammendrag:

Denne rapporten viser hvordan det er mulig å integrere Wolfram Alpha i spillmotoren Unity 3D. Den viser og hvordan vi har laget spillet vårt "Weapons of Math Instructions".

Hovedfokuset i rapporten er en drøfting av problemene som følger med når Wolfram Alpha skal integreres inn i Unity 3D og hvordan vi har løst disse. Vi har i tillegg kommet frem til en alternativ løsning som passer fint inn i vårt spill uten bruk av Wolfram Alpha.

Stikkord:

Unity 3D	Wolfram Alpha	Spillprogrammering

Forord

Dette dokumentet er en sluttrapport til en bacheloroppgave ved Høgskolen i Bergen. Oppgaven gikk ut på å lage et lite spill for å intergrere "kunnskapsmotoren" Wolfram|Alpha i spillmotoren Unity3D. Vår løsning var å lage et artillerispill vi har kallt "Weapons of Math Instructions" som gjør dette.

Oppdragsgiveren vår har vært SIM Subsea, ved Pål Trefall, som møter samme problemstillingen i sitt prosjekt. Veiledere ved HiB har vært Harald Soleim og Jon-Eivind Vatne.

Denne rapporten inneholder en drøfting av problemsituasjonen og en beskrivelse av vår løsning. Vi har forsøkt å holde et teoretisk nivå slik at andreklassestudenter skal kunne lese rapporten.

Takk til

Pål Trefall, oppdragsgiver og Unity3D-guru. Pål har vært tilgjengelig, enten i person eller på skype, til enver tid og har bidratt med verdifull tilbakemelding på både spillmekanikk og arbeidsmetoder i Unity3D. En spesiell takk for å ha gitt en oppgave som er spenende men samtidig åpen nok til at vi har fått rom til å utfolde oss.

Harald Soleim, vår veileder, som har vært svært interessert i fremdriften til prosjektet, skaffet betatestere, rettet rapporten hundre ganger, og mye annet.

Jon-Eivind Vatne, matematiker og veileder, som har hjulpet oss med rapporten og matematikken i spillet.

Innhold

Figurer

1 Innledning

1.1 Organisering av rapporten

Rapporten er skrevet slik den skal leses. Det er lagt opp til at det ikke skal være nødvendig å bla frem og tilbake. I teksten er det lagt inn referanser til forskjellige avsnitt internt i rapporten, og til andre ressurser. I pdf-versjonen er det mulig å trykke på disse for å komme til selve referansen, mens du i tekstversjonen må bla selv. Alle figurer er listet opp under "Figurer" i starten av oppgaven.

1.2 Oppdragsgiver - SIM Subsea

SIM Subsea[1] er et dataspill som skal lære ungdom om undervannsinstalasjoner. Hovedeier og utviklingsansvarlig er Høgskolen i Bergen. Andre samarbeidspartnere er NCE Subsea, Hordaland Fylkeskommune, Bergen kommune, VilVite-senteret, Renatesenteret, Vest Næringsråd, MediArena, Business Region Bergen, Framo Engineering, Aker Solutions, FMC Technologies, Virkemiddel for regional innovasjon.

1.3 Problemstilling

Prosjektet går ut på å integrere Wolfram|Alpha[2] med spillmotoren Unity3d[3] ved å lage et spill som benytter seg av denne funksjonaliteten. Hva spillet går ut på og hvordan det ser ut er opp til oss. Vi har valgt å lage et såkalt artillerispill, lignende klassikere som Worms[4] og Scorched Earth[5] og nyere spill som Angry Birds.[6] Hovedforskjellen er at der man i disse spillene mer eller mindre bevisst definerer en vektor som definerer en andregradsfunksjon, vil man i spillet vårt selv skrive inn den matematiske funksjonen spilleren ønsker at prosjektilet skal følge.

1.4 Hovedidé for løsningsforslag

Hovedidéen var å lage et artillerispill hvor spilleren må skyte et mål med en kanon. I stedet for at prosjektilet følger en ballistisk bane må spilleren selv definere banen til prosjektilet som en funksjon $f(x)$.

Når spilleren skyter ville vi sende fuksjonen til WA og få tilbake en liste over prosjektilets posisjon over tid. Dette er beskrevet på side 6.

1.5 Prosjektform

Det er ikke blitt valgt en spesifikk prosjektform for dette prosjektet. Det begynte på dag en med å dele prosjektet opp i oppgaver til den enkelte. Etter dette har vi hatt uformelle møter hver morgen hvor det har blitt gått gjennom hva som trengs, og hvem som har ansvar for å få fikset dette.

2 Definisjon av Oppgaven

SIM Subsea ønsker å se på måter å integrere data fra Wolfram Alpha[2] med spill laget i Unity3D.[3] Selve oppgaven har ingen formell definisjon, da det er en del av oppgaven å designe et spill fra bunnen av som bruker matematikk; det er de tekniske løsningene som er spesielt interessant for SIM Subsea.

Gjennom diskusjon har studenter, veiledere og arbeidsgiver kommet fram til å lage et artillerispill. Hovedfokus for interaksjonen vil være å la Wolfram Alpha tolke matematiske uttrykk og gi svar tilbake i form av en graf, som så leder prosjektilet mot et mål i spill verdenen.

3 Analyse av problemet

3.1 Intergasjon av Wolfram|Alpha i Unity3D

3.1.1 Hente informasjon fra Wolfram|Alpha

Beskrivelse
Hovedformålet med oppgaven var å finne en god måte å hente informasjon fra Wolfram|Alpha (WA) inn i Unity3D og bruke den i spillet. Dette må nødvendigvis skje ved å opprette en tilkobling til WA fra Unity3D over internett og sende en forespørsel via den.

Løsning
WA har et grensesnitt laget spesifikt for integrering over internett. Det er basert på XML og HTTP GET-forespørsler. En forespørsel på formen `http://api.wolframalpha.com/v2/query?input=[søkestreng]&appid=[appid]` vil bli besvart med en xml-fil som inneholder WA sine søkeresultater.[7]

Unity3D har en ferdig klasse kalt "WWW" for å laste ned ressurser fra internett.[8] Vi bruker denne til å kjøre en nedlasting av xml-filen i bakgrunnen. Når filen er ferdig nedlastet kan vi bruke .NET sin xml-funksjonalitet[9] til å hente ut de verdiene vi trenger.

3.1.2 Forsinkelse

Beskrivelse
Når vi sender en forespørsel til WA tar det 2–3 sekunder før vi er ferdige å laste ned svaret. På grunn av denne tidsforsinkelsen tar det en liten stund fra spilleren trykker til det faktisk skjer noe i spillet, og det ser ut som om spillet har hengt seg.

Løsning
Vi kan maskere dette en del ved å skape inntrykk av at det skjer ting i spillet som f.eks. å spille en lyd og vise en animasjon. Dette er litt vanskelig slik vi bruker WA; vi har ingen naturlig animasjon for å tegne en graf.

Vi har valgt å gjøre bruk av WA valgfri med lokal parsing av utrykk som et alternativ. Bruk av en lokal parser vil også gjøre det mulig å spille uten internettilkobling.

3.1.3 Sandbox

Beskrivelse
Unity sin sin Webplayer kjører i en sandkasse (sandbox) som setter visse restriksjoner på hvilke operasjoner som er lovlige å utføre. Bl.a. innebærer dette at vi ikke kan laste ned innhold fra internett med mindre serveren/adressen vi forsøker å laste ned fra har en fil `crossdomain.xml` i rot-domenet sitt som sier at det er lov å laste ned fra denne siden.[10] WA har ikke denne filen.

Vi hadde i starten problemer med nedlasting av ressurser fra internett som vi trodde var relatert til Unity3D sin sandkasse. Dette har vist seg å ikke være tilfelle, men det oppdaget vi først etter at vi hadde funnet en måte å få tilgang til ressurser utenfor sandkassen.

Løsning
Vår løsning er å sette opp en reléserver som har filen `crossdomain.xml`, og som kan videresende innhold fra andre addresser på nettet. Vi har laget denne serveren selv og kalt den "catbox".

3.1.4 Lasting av gif

Beskrivelse
For å vise grafer fra Wolfram|Alpha i spillet ønsker vi å laste dem ned fra nettet og legge på et objekt i spillet som en tekstur. Unityklassen WWW[8] som vanligvis kan gjøre dette har kun støtte for png- og jpg-bildefiler. Wolfram|Alpha eksporterer sine bilder i gif-format.[7]

Løsning
Vi kan se flere mulige løsninger:

- Konvertere bildet før vi forsøker å vise det. Unity3D bruker .NET api 2.0, men biblioteket som vanligvis brukes til bildekonvertering er ikke støttet for inkludering her.[11] Alternativt kan vi skrive koden for å konvertere fra gif-format selv. Gif er ikke et veldig komplisert format,[12] men det er uansett en del arbeid å skrive en parser fra bunnen av.
- Laste ned og lagre filen på harddisk. Klassen "Resources" har støtte for gif-format, men brukes kun til å lese filer fra harddisk.
- Konvertere fra gif- til png-format på reléserveren. Dette kompliserer serveren litt og øker belastningen på denne.

Vi har valgt det siste alternativet.

3.2 Utviklingsverktøy

3.2.1 Spillmotor

Valget av spillmotor er en del av oppgavespesifikasjonen, og må derfor bli Unity3D. Unity3D[3] er et av verdens mest brukte spill programmeringsverktøy. Store deler av dette er gratis å bruke. Det er mulig å lage spill til Windows, Linux, Mac, IOS, Webplayer, og Android, helt gratis. Det finnes også betalversjoner av dette verktøyet. Disse versjonene åpner opp diverse ekstra funksjoner som f.eks. rendre film til tekstur, og flatelys. Det var ikke nødvendig for dette prosjektet å bruke noen av funksjonene som ble låst opp ved betalversjon, så vi har holdt oss til det som er tilgjengelig gratis.

3.2.2 IDE

MonoDevelop[13] følger med Unity, og er Unitys standard koderedigeringsverktøy. Man kan sette det til noe annet, som f.eks. Eclipse,[14] eller Notepad om man skulle ønske det, men vi holdt oss til MonoDevelop fordi denne var spesialtilpasset Unity.

3.2.3 Programmeringsspråk

Unity har støtte for Boo, som er basert på Python,[15] JavaScript[16] og C#.[17] I vår gruppe var den ingen som hadde særlig kunnskap til noen av disse språkene, men C# var det som var nærmest noe av det vi hadde prøvd før. C# er en blanding av C++ og Java, og hvis man kan disse så er det ikke noe problem å lære seg C#.

3.2.4 Versjonskontroll

Vi ønsket å ha programmet under versjonskontroll. Dette er i vår mening et krav i alle prosjekter som ikke er helt trivielle. Valget stod mellom git eller mercurial, som begge er distribuerte versjonskontrollsystemer.

Git har litt bedre støtte for å forgrene koden og kan hoppe mellom ulike grener av koden og dele spesifikt en gren med andre. Vårt valg ble derfor git.[18]

3.2.5 Dokumenter

Bachelorprosjekt involverer nødvendigvis en del dokumenter. Valget stod mellom Microsoft Word, LibreOffice, Google Docs eller LaTeX. Vi ønsket å ha rapporten under versjonskontroll også. I tillegg ønsket vi å kunne håntere konflikter i dokumentet når flere redigerer samtidig v.h.a. de samme verktøyene som hånterer konflikter i kildekode. Vi har også god erfaring med bruk av LaTeX til større dokumenter. Valget ble derfor LaTeX.[19]

LaTeX er et tegnsettingsverktøy som produserer dokumenter med en høy kvalitet fra tekstfiler med en egen syntaks. En god ressurs for skriving av LaTeX dokumenter ligger på Wikibooks sine nettsider.[20]

3.2.6 Wolfram|Alpha

"Kunnskapsmotor" på internett. Man kan spørre WA om nesten alt, på naturlig engelsk eller egen WA-syntaks. Den henter data fra mange ulike kilder, og presenterer dataene på forskjellige måter. Man kan f.eks. skrive "The chance of rain in Bergen, Norway today", og du får opp værmeldingen for Bergen de neste dagene. Det vi er interessert i fra Wolfram|Alpha er matematikk. WA kan returnere grafer, bilde av det matematiske uttrykket på matematisk form (sqrt(1/2)) blir til $\sqrt{\frac{1}{2}}$, forenkling av uttrykket, den deriverte, og mye annet. Vi kan også be om å få en tabell av Y-verdier for en funksjon av x. Alt dette gjøres vha. et web-API.

4 Utforming / løsninger

4.1 Evaluering av matematiske uttrykk

Prosjektet vårt er nødt til å evaluere et matematisk uttrykk gjentatte ganger for å få et prosjektil til å følge en bane. Valgene våre var:

- La Wolfram|Alpha regne ut alt
- Bruke en parser lokalt på maskinen som kjører spillet

Det å bruke Wolfram|Alpha var førstevalget, men dette viste seg å skape store praktiske problemer. Når man sender et uttrykk til WA, vil det ta ca. to sekunder før man får svar, litt avhengig av nettverkstilstander og uttrykkets kompleksitet. Dersom man skal få et prosjektil til å følge en bane, må man evaluere et uttrykk på en lang rekke posisjoner. Dersom uttrykket skal evalueres for 200 x-verdier, noe som kan være en underdrivelse i prosjektet vårt, vil det ta i underkant av syv minutter før prosjektilet kommer frem. Dette var uaktuelt, så prosjektet bruker en lokal parser.

Lenge etter at parseren var implementert, fant vi ut at Wolfram|Alpha kunne returnere en tabell med Y-verdier for et uttrykk. Det holder dermed med kun én spørring til WA for å få tilbake posisjonene langs banen. Se 5.2 for mer info.

4.1.1 Parser

Et syntakstre er en datastruktur som inneholder den grammatiske strukturen av et uttrykk.[21] Dette gjør bl.a. at en datamaskin kan evaluere uttrykket på en langt enklere måte enn dersom det er representert som en rekke tegn. Dette har stor nytte når man skal få maskinen til å forstå et språk, som f.eks. norsk, engelsk eller C++, eller matematiske uttrykk (som teknisk sett er språk). Det å finne meningen til en tekststreng, f.eks. ved å bygge et syntakstre, er jobben til en parser.[22] C# har ingen innebygde funksjoner (som vi kunne finne) for å parse, og det å lete etter en parser som passet våre behov, og som ikke er kommersiell, ville ta lengre tid enn å lage en selv. Vi skrev derfor vår egen parser.

Parseren vår er designet for å forstå samme matematiske syntaks som Wolfram|Alpha. Dette fordi uttrykket skal, uten å modifiseres, kunne sendes til WA, som returnerer f.eks. en graf av dette uttrykket. En liten forskjell i syntaksen kan føre til at grafen ikke stemmer overens med banen prosjektilet tar når uttrykket evalueres lokalt. Grafen er bare et bilde som vises til spilleren, og har ingenting å gjøre med evalueringen.

Parseren fungerer i grove trekk slik:

1. Modifisér uttrykket til å passe en standard syntaks
    - Gjør alle store bokstaver små
    - Fjern alle blanke tegn
    - Gjør om komma (,) til punktum (.)

- “Snarveier” som “5x” blir til “5*x”

2. Del opp uttrykket i såkalte tokens, som representerer byggesteinene i uttrykket
3. Bruk disse tokenene til å bygge syntakstreet

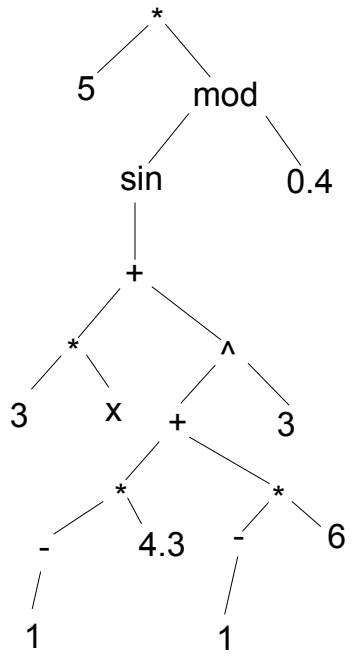
Merk at denne listen ikke er komplett. Et eksempel på parsingen av et vilkårlig valgt uttrykk:

Uttrykk:
`5Sin(3x+(-4.3-6)^3) mod .4`

Etter modifisering:
`5*sin(3*x+((-1)*4.3+(-1)*6)*0.3)mod0.4`

Lag tokens av uttrykket. Hver parentes havner i en egen liste:

Liste 0 `5 * sin ref1 mod 0.4`
Liste 1 `3 * x + ref2 * 0.3`
Liste 2 `ref3 * 4.3 + ref4 * 6`
Liste 3 `- 1`
Liste 4 `- 1`



Figur 1: Syntakstre

Treet kan deretter evalureres ved å oppgi en x-verdi (og en y-verdi, dersom det hadde vært med i uttrykket), og traversere treet rekursivt infix fra rotnoden. Dette returnerer en numerisk verdi. Verdiene oppgis som en instans av klassen “Dictionary<char, double>”.

4.2 Spilldesign

Designet på spillet, og da hovedsaklig det tekniske, har vært gjennom flere revisjoner i prosjektet. I denne seksjonen beskrives utviklingsforløpet til spillet.

4.2.1 Selve spillet

Den opprinnelige planen var mer eller mindre en klone av Scorched Earth.[5] Terrenget skulle defineres av en matematisk formel, men prosjektilene skulle følge andregrads-funksjoner. Dette ble for simpelt.

Senere begynte vi å leke med tanken på Battleship,[23] hvor to spillere med hvert sitt brett ber skipene sine om å skyte over en hindring langs en matematisk bane de selv definerer. Spillerne kan ikke se skuddene før de kommer over på deres halvdel av spillområdet, og man kan derfor bruke disse banene til å få prosjektilet til å se ut som det kommer fra et helt annet sted. Hindringen ville være plassert i midten, og kunne vært et isfjell som spillerne også hadde mulighet til å skyte torpedoer under, og å smelte seg igjennom for å få fri sikt til motstanderen. Dette er idéen vi begynte å utvikle, men utviklingen tok en annen retning.

4.2.2 Poengsystem

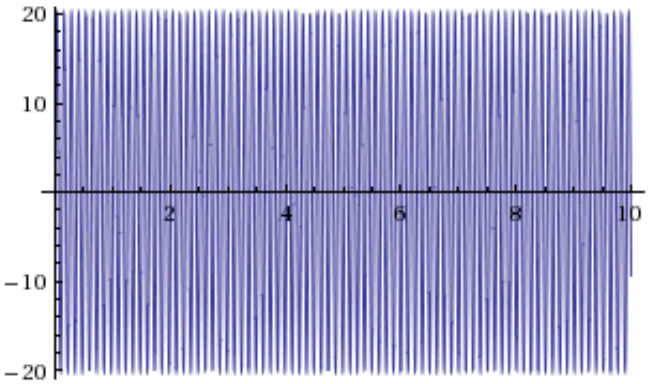
Antall forsøk
Det første poengsystemet vi hadde var basert på "liv" (forsøk), slik som gamle arkade-spill.[24] Man hadde en kanon som skjøt mot et mål, og hadde noen hindringer på veien dit. Som idéen har vært nesten hele veien, skulle prosjektilet følge en bane definert av spilleren for å unngå hindringene og deretter treffe målet. Ett poeng ble gitt for hvert forsøk man hadde igjen når målet ble truffet, og man begynte med tre forsøk. Gikk man tom for forsøk, ble man sendt tilbake til hovedmenyen, men ingenting hindret spilleren i å gå tilbake til den utfordringen igjen og prøve på nytt. Dersom man traff målet på siste forsøk, og dermed fikk bare ett poeng, kunne spilleren memorisere funksjonen han/hun brukte og ganske enkelt ta utfordringen på nytt, bruke funksjonen på første forsøk, og få tre poeng.

Slike livsystem har vært utdatert siden arkadenes gullalder på 80-90-tallet, og var da designet utelukkende for å få folk til å bruke mer penger. Grunnen til at vi hadde dette systemet var at vi ikke kom på noe bedre. Spillet var også direkte kjedelig i denne fasen.

Stjerner
Senere kom vi opp med et mer interessant poengsystem som adresserte problemene med det forrige: Stjerner. Det blir plassert stjerner mellom kanonen og målet, og poeng blir gitt etter hvor mange stjerner prosjektilet treffer før det når målet. Stjernene kan plasseres hvor som helst, og utfordringene kan designes slik at en eller to stjerner kan nås

med relativt enkle funksjoner, mens en tredje stjerne krever en mer komplisert funksjon. Alle stjernene kommer tilbake etter at prosjektilet har truffet målet (eller blitt ødelagt på andre måter), og man får ikke to poeng dersom man tar én stjerne i ett skudd, og en annen i neste. Man får poeng etter hvor mange man får med seg på ett eneste skudd.

Senere kom vi også opp med antistjerner, som tar bort poeng dersom man treffer de. Disse kan brukes for å hindre spillere i å bruke "brede" funksjoner som f.eks. $20\sin(50x)$ (figur 2), som er i stand til å ta alle stjerner mellom -20 og 20 på Y-aksen.



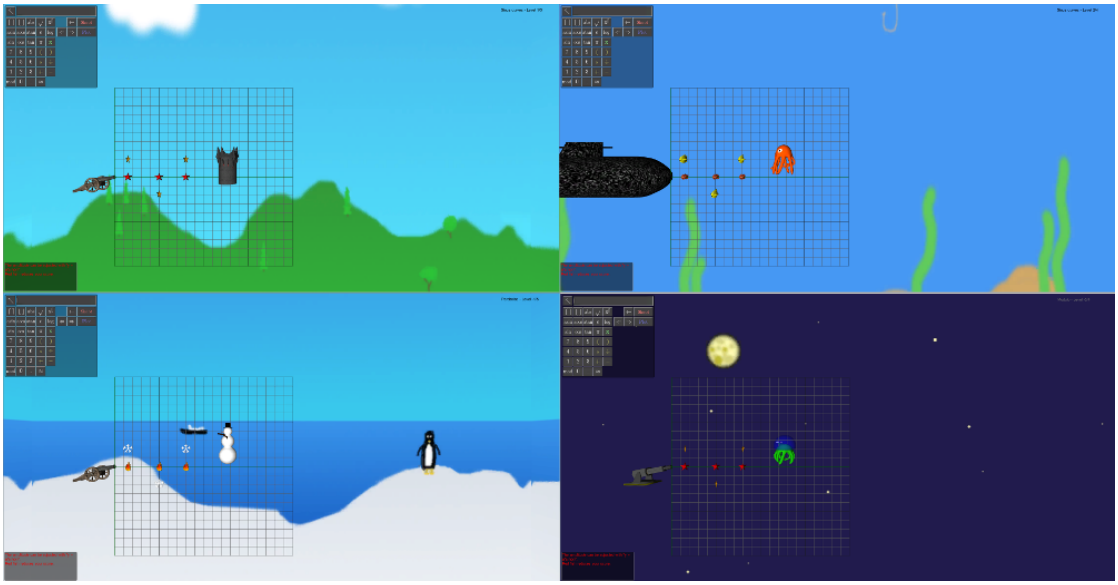
Figur 2: $20\sin(50x)$

I tillegg kan vi også legge inn stjernefragmenter i en utfordring. Disse vil kreve at man får tak i et gitt antall av dem før man får et poeng. Dette kan brukes til å tvinge spilleren til å bruke en spesifikk bane for å få poeng, og vil hovedsaklig legges inn i opplæringsdelen av spillet (tutorialen).

4.2.3 Temaer

Etterhvert ble utfordringene visuelt veldig like. Det var bare posisjonene på objektene som endret seg. Et visuelt uniformt spill blir fort kjedelig, så vi begynte å tenke på forskjellige temaer. Dette ble førte også til en enorm endring av den underliggende spillogikken. Mer om det senere.

Temaer er en ren visuell endring av utfordringene. De tillater oss å bytte ut modellene til kanonen, stjernene, målet, og det meste annet i scenen. F.eks. kan vi ha et tema hvor bakgrunnen er en jungel, kanonen er et urmenneske med bue, prosjektilet er en pil, og stjernene er fugler. Et annet tema kan være under vann. Bakgrunnen kan da være havbunnen, kanonen kan være en ubåt, prosjektilet en torpedo, og målet en kjempeblekksprut. Spillet er fortsatt akkurat det samme, det bare ser annerledes ut.



Figur 3: Samme brett med 4 forskjellige temaer

En av tankene med å ha temaer var at vi kunne ha en enkel historie i spillet. Man begynner i steinalderen, hvor man ikke har bruk for mer kompliserte funksjoner enn rette linjer. Deretter havner man i en annen tid hvor man har bue, og trenger andregradsfunksjoner, og så videre opp til moderne tid, eller forbi. Dette ble svært løst implementert.

4.3 Tospiller

Muligheten for å utfordre en venn i spillet er noe som kan løfte spillet en stor del. Uten en tospillerdel vil spillet være over når alle utfordringene er gjort. Spillet har da liten gjenspillingsverdi. En tospillerdel vil gjøre spillet mer attraktivt, spesielt på en tilsetning som expo. Målet er å skyte motspillerens kanon.

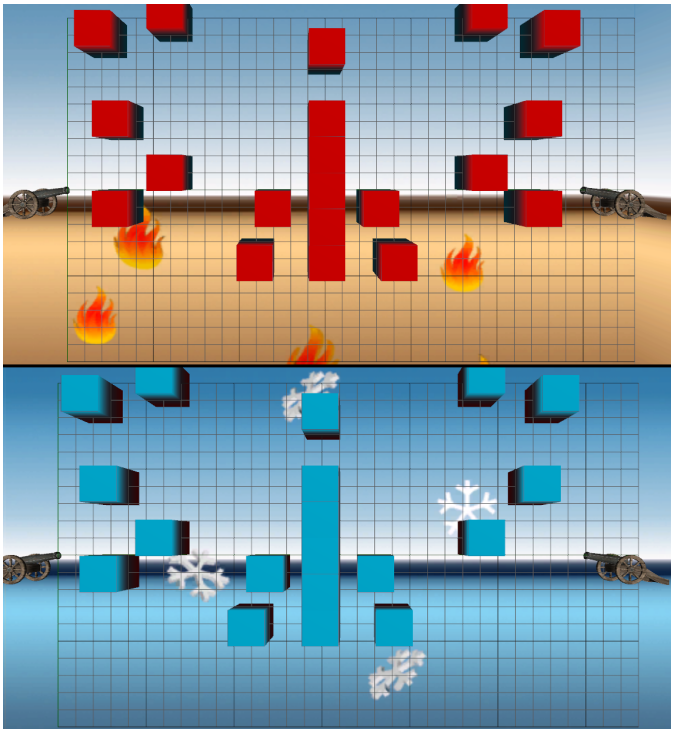
Det er flere problemer med å lage en tospillerdel. De største utfordringene vi kom til var:

- Levelmanageren vår må gjøres om
  - Må tåle to kanoner
- Brett må lages på en helt annen måte
  - Lage brett tilfeldig?
  - Speile brett?
  - Lage alt selv?

Hvis brett skal lages tilfeldig må alle brett bli laget slik at kanonene alltid står på samme sted. De vil da ha hindringer som vil bli lagt til på tilfeldige steder hver gang du starter et nytt spill. Det skal da være mulig å skyte på hindringene, og når de blir truffet vil de eksplodere og forsvinne fra brettet. Dette vil gjøre at det er mulig å gjennomføre alle baner.

Den alternative måten er at vi lager brettene selv. Dette vil føre til et begrenset utvalg av brett, og spillere kan memorisere funksjoner som passer til disse. Det blir da ingen utfordring å spille mot noen. Det blir også tungvindt å lage disse brettene. Enten må leveleditoren modifiseres, eller så må vi selv skrive XML-filene som definerer disse brettene.

Speiling av brett sørger for at ingen av spillerne får en fordel, med unntak av å begynne først. Når en hindring legges til, vil denne legges på spilleplanet. I tospillermodus er det ett spilleplan for hver spiller, men disse ligger inni hverandre. Forskjellen på disse planene er hovedsaklig lokalt origo, og retningen på X-aksen: de går motsatt vei. Dette gjør at vi enkelt kan speile et brett. Dersom man har koordinatene til en hindring, legges denne til en gang i hvert plan. Brettet vil da være likt for begge spillerne.



Figur 4: Bilde av tospillerdelen

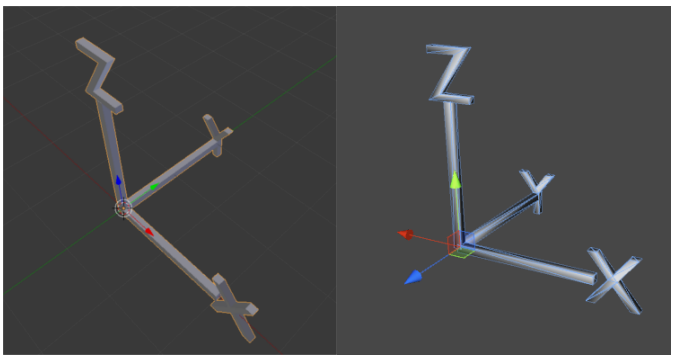
I stedet for å skrive om levelmanageren, ble det skrevet en ny til bruk i tospiller. Tospiller bruker også en egen Unityscene, som med unntak av levelmanageren er lik den som brukes i enspiller. Tospillerlevelmanageren, heretter kalt “Hotseatmanager”, genererer et kamera, et lys, to kanoner, og et tilfeldig antall tilfeldig plasserte hindringer. I tillegg har hotseatmanageren kontroll på hvilken spiller det er sin tur, og den har ansvaret for å flytte og snu kameraet, slik at det alltid skytes fra venstre mot høyre. Den husker også hvilken funksjon som ble skrevet inn sist det var den aktuelle spilleren sin tur, slik at han/hun slipper å skrive den om igjen hver gang.

Vi endte opp med å la tospillerbrettene være tilfeldig generert og speilvendt. Da vil det ikke være mulig for spillere å memorisere funksjoner, og vil dermed kun ha sin matematikkunnskap som fordel. Det vil alltid være minst én hindring på linjen mellom kanonene, for å forhindre at noen vinner ved å skyte en rett linje på første skudd.

4.4 Bruk av Blender

Blender[25] er et gratis 3D-modelleringsverktøy med åpen kildekodelisens. Programmet er på høyde med kommersielle titler, som Maya[26] og 3DS Max.[27] Å integrere Blender med Unity er svært enkelt. Dersom Blender er installert på maskinen, vil Unity automatisk bruke dets biblioteker til å konvertere Blenders .blend-filer til et format Unity håndterer (fbx[28]). Det er imidlertid en del fallgruver:

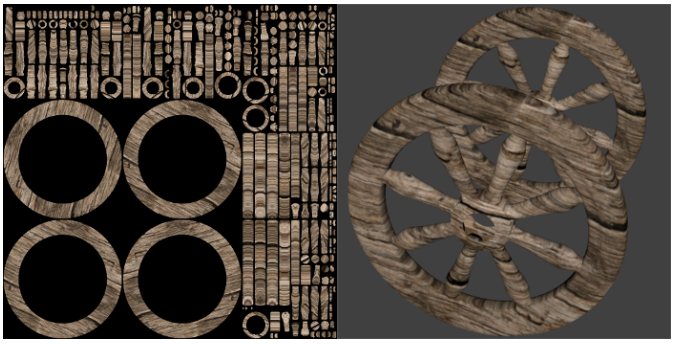
Akser I Blender er "rett forfra" det samme som å se langs positiv retning i Y-aksen. Da vil X-aksen gå mot høyre, og Z-aksen rett opp. I Unity er "rett forfra" det samme som å se langs negativ retning i Z-aksen. Da vil X-aksen gå mot venstre, og Y-aksen rett opp. Dette gjør at modeller må speiles og roteres i diverse akser for at rotasjon 0,0,0 i Unity skal være riktig.



Figur 5: En modell av aksene, laget for å passe Blender sine akser. Venstre: Blender. Høyre: Unity.
De fargede pilene viser de faktiske aksene i det aktuelle programmet. X, Y, Z

Partikler Blender har et svært kraftig partikkelsystem hvor det er relativt enkelt å lage flotte effekter som eksplosjoner, bobler, og annet. Unity nekter å anerkjenne eksistensen til partikkeleffekter laget i Blender, noe som gjør at man må bruke Unity sitt eget partikkelsystem hvis man ønsker slikt.

Teksturering Unity krever et teksturkart (UV-kart) for å legge tekstur på modeller. I Blender kan man legge en vilkårlig tekstur på et vilkårlig objekt og si at den skal pakke det på som en kube, kule, flate eller annet. Det er heldigvis en grei sak å lagre dette som et UV-kart. Først lager man teksturkoordinater for objektet ved å pakke det ut (UV-unwrap). Deretter oppretter man et bilde som man ønsker å lagre teksturen i, og deretter "baker" man teksturen til bildet. Slik kan man lage UV-kart uansett hvordan modellen opprinnelig er teksturert.



Figur 6: Til venstre: Teksturkart. Til høyre: Modell teksturert med teksturkartet. Teksturen var opprinnelig et bilde av treverk, som ble pakket på modellen som en kube.

5 Implementering

5.1 Arkitektur

Spillogikken er ganske sentralisert. Den er samlet i tre hovedklasser:

5.1.1 GameManager

Ser man bort fra Unitymotoren, styres spillet av klassen NewAndBetterGameManager, heretter referert til som bare gamemanager eller GM. Denne klassen er en singleton, som betyr at det kan ikke eksistere mer enn én instans av den, og denne instansen er tilgjengelig fra hvor som helst ellers i programmet. Gamemanageren håndterer poeng, og holder styr på nummeret på utfordringen som spilles, og laster nye ved behov. Den fungerer også som et grensesnitt mot en instans av klassen ConfigManager, som styrer det meste av konfigurasjonen.

5.1.2 LevelManager

Alle utfordringene foregår i en og samme scene. Scenen bygges dynamisk av klassen LevelManager, som ligger på et tomt objekt som allerede eksisterer i scenen. Denne spør gamemanageren om hvilke objekter som skal være i scenen, og posisjonen til disse. Sammen med denne informasjonen får den også en filbane som peker til en mappe. I denne mappen finner den prefabs, som er en slags mal på et spillobjekt, som skal instansieres i scenen. Hvert tema har en egen slik mappe, med sin egen versjon av stjerner, kanoner, mål, og annet. LevelManager vil så plassere instanser av disse prefabene i scenen på de posisjonene den får oppgitt.

I tillegg til plassering av objekter, holder LevelManager styr på bl.a. kollisjoner og andre hendelser i scenen. Den kontrollerer også hva som vises, og hvor det vises. Det er også denne som initialiserer spørringer mot Wolfram|Alpha, som håndteres av klassen Wolfram.

LevelManageren har også ansvaret for å fortelle kanonen at den skal skyte prosjektilet. (Avsnitt 5.2)

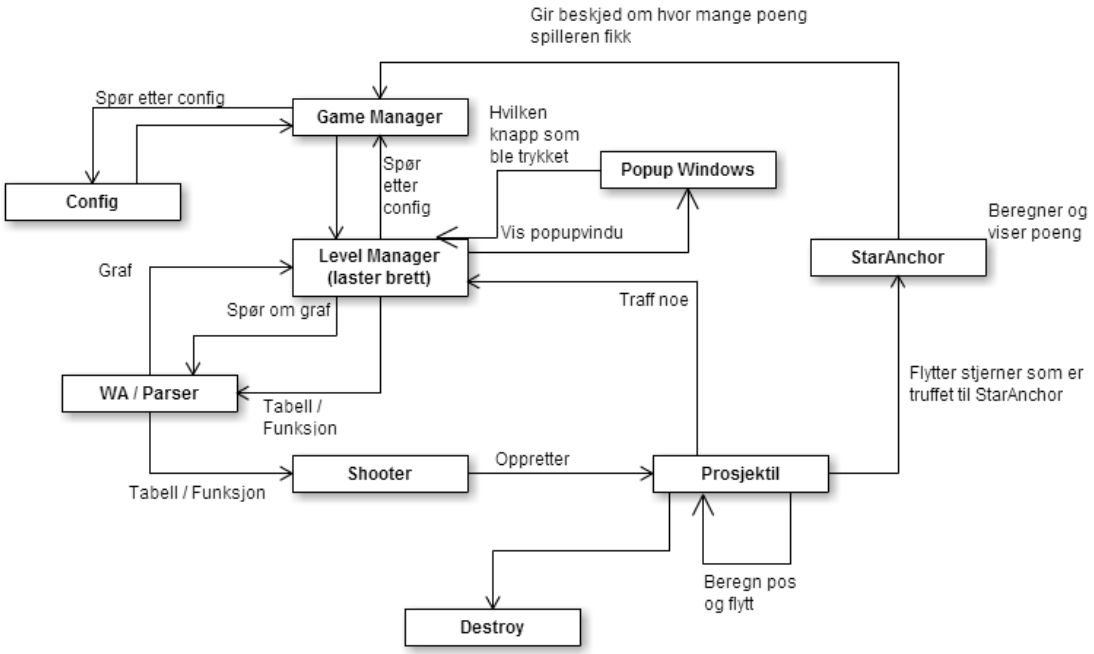
Poeng
Poengene lagres i GameManager, men de beregnes av en klasse som heter StarAnchorScript. Hver gang en stjerne, antistjerne, eller et stjernefragment treffes, vil dette overta kontrollen over dette. Den vil beregne antall poeng utifra disse, og gi dette til GameManager. Den har også ansvaret for å vise poengene etter at prosjektilet har truffet noe.

5.1.3 ConfigManager

Klassen ConfigManager håndterer (nesten) alt av konfigurasjon og innstillinger. Denne har ansvaret for å laste XML-filer, og å parse disse. Den viktigste XML-filen er

levels.xml, som inneholder konfigurasjonen for alle utfordringene. Den har delvis støtte for å laste vilkårlige slike XML-filer fra mappen Unitysandkassen gir tilgang til, men kan ikke dette for tiden. Tanken bak dette er at spillere selv kan lage egne sett med utfordringer, og dele disse med venner. Foreløpig kan ConfigManager kun laste filer fra Resources-mappen. Den andre XML-filen denne klassen håndterer, er themes.xml, som definerer navn på temaer. Den genererer den ovennevnte filbanen til temamapper, som sendes til LevelManager. Se (5.3) for formatet på XML-filene.

I tillegg til dette laster den brukerinnstillinger. Lagringen av disse skjer hovedsaklig i SettingsScript.cs men foregår også andre steder i koden. Disse lagres et sted vi ikke har direkte kontroll på (plattformavhengig), men vi har tilgang til dem gjennom Unityklassen PlayerPrefs.



Figur 7: Arkitekturen til spillet

5.1.4 Wolfram

Klassen Wolfram er bindeleddet mellom Unity3D og Wolfram|Alpha. Informasjonen vi henter fra WA er en tabell med Y-verdier mellom kanonen og målet, og et bilde av grafen. Når Wolfram får en forespørsel om en av disse starter en prosess i bakgrunnen som laster ned og sender ressursen til LevelManager når den er ferdig. Vi har valgt å

hente informasjonen asynkront for å ikke "fryse" spillet mens vi laster ned ettersom dette kan ta litt tid.

Å starte nedlasting av en ressurs asynkront er i Unity3D relativt enkelt. Funksjonen

```
1  IEnumerator download( string url ) {
2    WWWW www = new WWW( url );
3    yield return www;
4    // Not reached until download complete/error
5    if ( www.errror ) {
6      // Handle
7    }
8    string content = www.text;
9    // ...
10 }
```

kan startes asynkront med

```
1  StartCorutine( download( "example.com" ) );
```

Den vil da kjøre i parallell med resten av spillet. På linje 3 vil funksjonen gi fra seg kontroll til Unity3D og den vil fortsette på linje 4 først når nedlastingen er ferdig eller eventuelt en feil har oppstått.

Å sende en forespørsel til WA er også ganske rett fram. For å f.eks. søke etter strengen "sin(1 * x)" sendes en HTTP-forespørsel til adressen
`http://api.wolframalpha.com/v2/query?appid=xxx&input=sin(%201%20*%20x%20)&format=image,plaintext`.
WA vil da besvare forespørselen med en xml-fil.[7] En forkortet versjon kan se ut som:

Listing 1: Eksempel xml-fil

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true'
  error='false'
  numpods='7'
  ...
 <pod title='Input'
   id='Input'
   ... >
  <subpod title=''>
   <plaintext>sin(1 x)</plaintext>
   <img src='...'
     alt='sin(1 x)'
     title='sin(1 x)' />
  </subpod>
 </pod>
 <pod title='Result'
   id='Result'
   ... >
  <subpod title=''
    primary='true'>
   <plaintext>sin(x)</plaintext>
   <img src='...'
     alt='sin(x)'
     title='sin(x)' />
  </subpod>
 </pod>
 <pod title='Plots'
   id='Plot'
   ... >
  <subpod title=''>
   <plaintext></plaintext>
   <img src='...'
     alt=''
     title='' />
  </subpod>
 </pod>
 ...
</queryresult>
```

En forenklet metode for å hente ut nettadresse til en graf kan se ut som `getGraph`. Denne metoden bruker .Net sin funksjonalitet for parsing av xml.[9] Feilhåndtering er ignorert for lesbarhetens skyld.

Listing 2: getGraph

```
string getGraph( string xmlFileContent ) {
  XmlDocument doc;
  XmlNode     root;
  XmlNode     res;
  XmlNodeList nodes;
  string      imgUrl;

  doc = new XmlDocument();
  doc.LoadXml( xmlFileContent );
  root = doc.FirstChild;

  nodes = root.SelectNodes( "/queryresult/pod" );
  foreach ( XmlNode node in nodes ) {
    switch ( node.Attributes.GetNamedItem("id").InnerText ) {
    case "Plot" :
      res = node.SelectSingleNode( "subpod/img" );
      imgUrl = res.Attributes.GetNamedItem("src").InnerText;
      break;
    }
  }

  return imgUrl;
}
```

Når vi laster ressurser fra nettet er det en tidsforsinkelse på 2-3 sekunder. Vi har forsøkt flere måter å korte ned på denne ventetiden. Tiden som går med til å sende data over en begrenset båndbredde kan vi ikke gjøre noe med, men vi kan forbedre andre faktorer. Det første vi gjorde var å begrense søket til WA til kun de delene vi er interessert i. Da vi WA bruke mindre tid på å generere data vi ikke vil ha og filstørrelsen på svaret blir mindre.

Vi har også eksperimentert med å ikke lage nye WWW-objekter for hver nedlasting; ved å se på nettverkstrafikken mens vi startet en ny nedlasting så vi at det tar nesten et sekund fra objektet opprettes til det begynner å laste ned. Vi observerte tilsvarende oppførsel i både Unity3D sin WWW-klasse og .Net sin WebClient-klasse. [i]

i. Takk til gruppe 15 (Hans Kristian Brendmo og Stian Nesse Moe) som fant dette ut.

Vi forsøkte å laste ned ressurser på en annen måte, v.h.a. én instans av et WebClient-objekt. Da opplevde vi en marginal forbedring i nedlastingstid, men dette påvirket responstiden i spillet, som hadde en tendens til å "fryse" i starten på en nedlasting. Nedlasting fikk også en økt kompleksitet. Responstiden i spillet vil sannsynligvis forbedres av å laste ned asynkront. Et problem ved å kun ha en WebClient er at dersom man klikker først "plot" og deretter "shoot" før plotten er fremme, vil WebClient allerede være opptatt, og dermed kaste et unntak. For å omgå dette må vi enten opprette en kø, eller avbryte forrige spørring. Dette fant vi ikke verdt innsatsen for en marginal forbedring. Gevinsten ved å parse uttrykket lokalt er mye større.

5.2 Styring av prosjektil

Prosjektilet er det spillobjektet spilleren har kontroll over. Det å få dette til å bevege seg troverdig langs grafen til funksjonen spilleren definerer, er en viktig del av spillet. Spilleren kan, i options-menyen, velge om det er Wolfram|Alpha eller parseren (4.1.1) som skal styre prosjektilet.

5.2.1 Posisjonering

Prosjektilet beveger seg med konstant fart langs x-aksen. Den nøyaktige avstanden mellom hver utregning av posisjonen er avhengig av tiden mellom tegningen av to bilder på skjermen, og vil dermed variere fra maskin til maskin.

Uavhengig av om det er WA eller parseren som styrer prosjektilet, vil det alltid starte i enden av kanonens løp. Grafen man får tilbake fra WA vil ikke gjenspeile dette, ettersom logikken som gjør dette utføres kun på tabellen som kommer tilbake.

Wolfram|Alpha

Dersom Wolfram|Alpha er valgt som backend, vil spillet sende funksjonen til WA og spørre etter en tabell med Y-verdier. Mengden verdier som returneres er indirekte begrenset av WA-lisensen vår. Hver verdi tar tid å regne ut, og vi får begrenset tid per spørring. Foreløpig hentes verdier i steg på 0.15 langs x-aksen, noe som gir noen hundre y-verdier tilbake, avhengig av grensene på det aktuelle spilleplanet. En delta X på 0.15 gir veldig synlige "trappetrinn" når prosjektilet beveger seg. Det er derfor implementert lineær interpolasjon mellom verdiene fra tabellen, slik at prosjektilet får en mye jevnere, men fortsatt ikke perfekt, kurve.
For å få prosjektilet til å starte i enden av kanonens løp, vil den første verdien i tabellen trekkes fra alle verdiene i tabellen.
Eksempel:

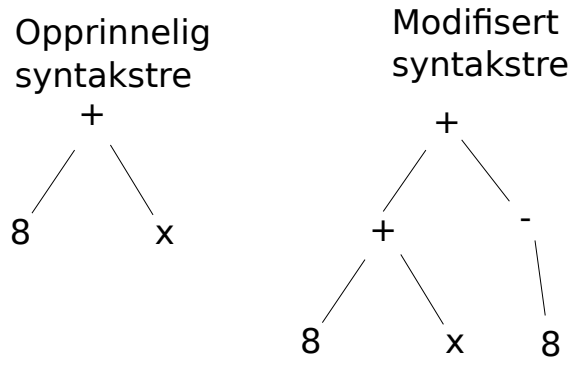
5	3	8	...	9

Trekk fra første elementet, 5, fra alle:

0	-2	3	...	4

Parser

Hvis backend er satt til lokal, vil vår hjemmelagde parser bygge et syntakstre som representerer funksjonen. Da kan vi ganske enkelt gi en vilkårlig X-verdi til dette treet, og få helt nøyaktig Y-verdi ut. Oppløsningen på kurven er da ikke begrenset av tiden vi får på WA sine servere. Interpolasjon er heller ikke nødvendig, ettersom treet regner ut riktig Y-verdi for nøyaktig denne X-verdien.
For å få prosjektilet til å starte i enden av kanonens løp, vil synakstreet utvides. Det opprettes en ny addisjonsnode som får det opprinnelige treet til venstre, og en substraksjonsnode til høyre, som substraherer verdien til uttrykket på X=0.

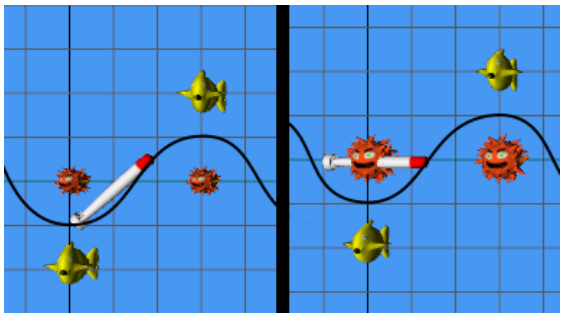


5.2.2 Rotasjon

Kuleprosjektiler kan følge den banen de vil uten å se unaturlig ut (hvis man ser bort fra mangelen på fysikk). Avlange prosjektiler, som torpedoer og piler, er nødt til å rotere seg slik at de følger banen til funksjonen. Rotasjonen til prosjektilet i et gitt punkt er lik tangens invers av tangenten til kurven i dette punktet. Ideelt sett skulle vi brukt derivasjon til dette, men det er svært komplisert å skrive kode som kan derivere matematiske uttrykk. Wolfram|Alpha kan gjøre dette for oss, men med tanke på at det tar ca. 2 sekunder per kall valgte vi å gå for andre løsninger.

Den aktuelle formelen for å regne ut vinkelen er:

$$\alpha = \arctan(\frac{\Delta y}{\Delta x})$$



Figur 8: Venstre: Prosjektil som roterer seg etter banen. Høyre: Prosjektil som hele tiden har samme rotasjon. Merk at det er kun fronten på prosjektilet som kan treffe objekter

Wolfram|Alpha
: Tangens til funksjonen regnes ut fra tabellen. Tabellen har en fast delta X, og Y-verdiene til de to X-verdiene i tabellen som ligger nærmest den faktiske X-verdien vil brukes til å regne ut delta Y.

Det er ingen interpolasjon på denne rotasjonen. Det er tydelig dersom man har brukt lokal utregning lenge. Grunnen til dette er at lineær interpolasjon ikke vil ha noe effekt, og kvadratisk eller annen polynomisk interpolasjon vil være som å prøve å approksimere funksjonen programmet nettopp har sendt fra seg. Vi antar at kvadratisk interpolasjon[29] vil fungere greit, men dette er ikke implementert.

Parser

Dersom den lokale parseren brukes, får man langt mer nøyaktig rotasjon. Mens tabellen fra WA har en delta X på 0.15, kan vi sette parserens delta X så lav vi vil. Vi har satt den til 0.00001 (10^{-5}), altså 15000 ganger mer nøyaktig. Delta Y blir da regnet ut fra Y-verdien til den aktuelle X-posisjonen, og en tenkt posisjon 10^{-5} enheter lenger frem. Det er da ikke nødvendig med noe interpolasjon for å få rotasjonen til å være jevn.

5.3 Konfigurasjonsfiler

Opprinnelig ble alle utfordringer laget som scener i Unity. Dette var enkelt, og utfordringer ble raskt produsert. Ulempen med dette var at det var svært statisk. Dersom man ønsket å endre på en utfordring, måtte man åpne den aktuelle scenen i Unity. Dersom man ønsket å gjøre en endring på alle utfordringene, måtte man åpne alle hver for seg og gjøre endringene.

Som en forbedring til dette begynte vi å bruke maler (templates). En mal var et objekt som representerte et objekt i den ferdige utfordringen, og som ble byttet ut med det riktige objektet når utfordringen ble lastet. Dette tillot oss også å ha forskjellige temaer, hvor utfordringens utseende er bestemt av en liste i en konfigurasjonsfil. Dette ga oss mer frihet, men det var fortsatt tungvindt å endre på annet enn utseendet.

Etterhvert fant vi ut at vi kunne droppe maler, og lagre alle posisjoner i en fil. Dette gjør definisjonen av utfordringene separat fra selve spillet, og åpner mange muligheter. Alle utfordringene lastes nå dynamisk i én scene, noe som gjør at vi ikke trenger å bytte scene hver gang vi vil endre en utfordring. Scenen inneholder to objekter. Begge er usynlige for spilleren. Det ene, LevelManager, har et script som laster utfordringen, og håndterer hendelser i scenen. Det andre har et script som håndterer alt som har med Wolfram|Alpha å gjøre.

Det å definere utfordringene i konfigurasjonsfiler tillater spillere å lage sine egne, og å dele disse med andre. Alt lagres som XML-filer, som håndteres av klassen ConfigManager. Utfordringene (levels.xml) lagres på dette formatet (forenklet):

```
<levels>
  <level theme="int">
    <light><pos>...<rot>...</light>
    <camera><pos>...<rot>...</camera>
    <cannon><pos>...<rot>...</cannon>
    <target>
      <x>float</x>
      <y>float</y>
    </target>
    <star>
      <x>float</x>
      <y>float</y>
    </star>
      ... Vilkaarlig antall stjerner, inkludert 0 ...
    <antistar>
      <x>float</x>
      <y>float</y>
    </antistar>
      ... Vilkaarlig antall antistjerner, inkludert 0 ...
    <starfragment>
      <x>float</x>
      <y>float</y>
    </starfragment>
      ... Antall stjernefragmenter maa vaere et produkt av 5*n
          ...
    <bounds>
      <lowerx>0</lowerx>
      <upperx>40</upperx>
      <lowery>-20</lowery>
      <uppery>70</uppery>
    </bounds>
  </level>
  <level>
    ... Samme som over. Vilkaarlig antall utfordringer, minst 1
        ...
  </level>
</levels>
```

Kanonens posisjon og rotasjon er definert i verdenskoordinater. Denne konfigurasjonen definerer også spilleplanet. Koordinatene til stjerner, stjernefragmenter, antistjerner og målet er i dette planet. “bounds” er spilleplanets grenser. Dersom prosjektilet går utenfor disse, regnes det automatisk som tap. Temaet er et heltall som brukes til å slå opp i en annen konfigurasjonsfil. Denne er mye enklere, og er ikke tilgjengelig for brukeren.

Eksempel på temafilen:

```
<themes>
  <theme>Default</theme>
  <theme>Subsea</theme>
  <theme>Radio</theme>
  <theme>Hotseat</theme>
</themes>
```

Temaet er lagret som en tekststreng. Denne strengen er navnet på en mappe i Unityprosjektets Resources-mappe. Scriptet som håndterer konfigurasjonsfilene vil slå opp tallet fra levels.xml, og returnerer en URL til den aktuelle temamappen, f.eks. “Themes/Subsea”.

Brukerinnstillinger, som f.eks. hastighet på prosjektil, visning av rutenett og annet, håndteres også av ConfigManager.

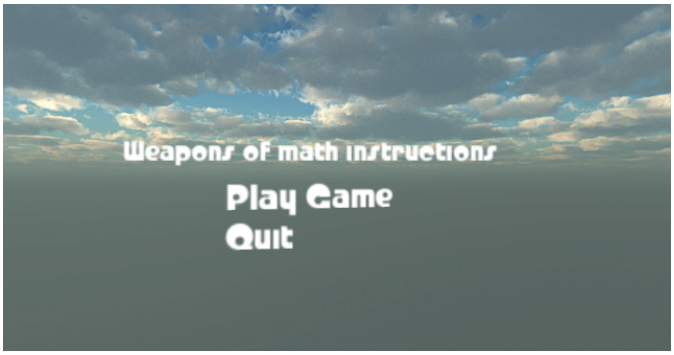
5.4 Brukergrensesnitt

Gruppen gikk mye frem og tilbake da vi skulle lage brukergrensesnittet. I starten ble det kun brukt Unity sitt eget brukergrensesnitt OnGUI.[30] I dette definerer man tekstbokser, knapper og lignende direkte i en metode kalt OnGUI i skriptet. Dette gjør at det blir veldig mye kode som blir skrevet veldig mange ganger. Skal du for eksempel ha 15 knapper, må det skrives 15 ganger at du skal ha en knapp, hvor du vil ha den og hvilken størrelse den skal ha.

I et forsøk på å finne et bedre alternativ ble NGUI (Next Generation User Interface)[31] prøvd. Det så lenge ut som om dette hadde alt spillet trengte. Det var lett veldig lett å bruke, og det så bra ut. Med NGUI var det mye mer gjenbruk av kode, og du slapp å definere alle knapper i skript en etter en. Det eneste problemet som dukket opp var inntastingsfeltet. Etter mye prøving og feiling ble det avdekket at dette hadde store mangler. Det var kun én måte å utløse det, og det var ved å trykke enter tasten, det var ikke mulig å kopiere inn tektst eller å navigere i feltet ved å bruke musen eller piltastene.

Dette gjorde at brukergrensesnittet kom til et stopp. Spillet trengte grensesnittet til NGUI, men det måtte også ha funksjonaliteten til OnGUIs grensesnitt. På grunn av dette måtte det bli brukt litt av begge grensesnittene. Vi måtte gå vekk fra NGUI på alle løsninger hvor det var behov for et inntastingsfelt, mens på alle andre (som f. eks menyer) kunne det fortsatt brukes. Derfor er nå alle former for inntastingsfelt og kalkulatorer laget ved hjelp av OnGUI, mens alle former for menyer er laget av NGUI.

Vi har prøvd flere forskjellige typer inntastingsfelt, inntastingskjermer og menysystemer:



Figur 9: Gammelt menysystem

Figur 9 viser det første menysystemet som ble tatt i bruk. Her er det bare tektsbokser som sender brukeren videre når de blir trykket på. Et veldig enkelt system som fungerer helt greit, men det er ikke akkurat vakkert.

Figur 10: Nytt menysystem

Figur 10 viser menysystemet som har blitt brukt. Dette var menysystemet som vi bestemte oss for å bruke, dette ser mye mer moderne og profesjonelt ut.

Figur 11: NGUI inntastingsfelt

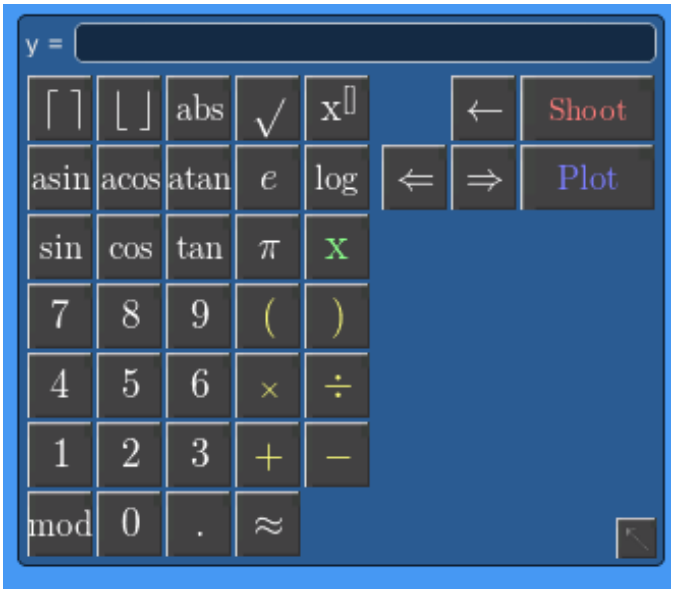
Figur 11 viser skjermen slik vi ville at den skulle se ut, men grunnet store mangler med NGUI var ikke dette mulig å oppnå med de funksjonalitetene vi skulle ønske. Grunnet manglene her vil mange brukere som bruker mobile plattformer ha store problemer med å forstå hvordan det skal fungere.

Figur 12: OnGUI inntastingsfelt

I figur 12 er inntastingsfeltet endret til Unitys eget system, OnGUI. Dette arbeider mye bedre opp mot brukeren. Selve brukergrensesnittet har fortsatt store mangler med tanke på mobile platformer, og derfor ble det senere bestemt at det måtte legges til en form for kalkulator som det var mulig å trykke på for å få utrykk som kunne brukes.

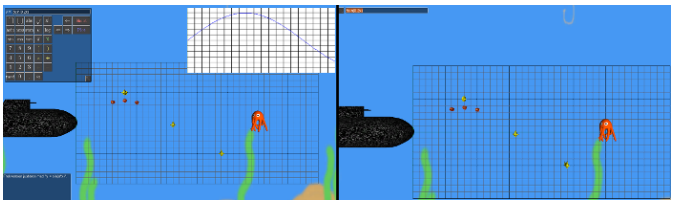
Figur 13: OnGUI inntastingsfelt med Kalkulator v.1

Her brukes fortsatt OnGUI, men det er lagt mye bedre opp mot mobile platformer. Her kan man trykke på kalkulatoren, og det vil komme opp i inntastingsfeltet slik som det ville kommet hvis du hadde skrevet det på tastaturet. Denne kalkulatoren fungerte meget bra, men den så ikke helt fantastisk ut.



Figur 14: OnGUI inntastingsfelt med Kalkulator v.2

Her ble det brukt det samme systemet som i den andre kalkulatoren, men det er bare lagt på mye finere teksturer på knappene og lagt til noen ekstra funksjoner. Knappene er også gjort litt større. Dette måtte til fordi det var svært vanskelig å treffe de små knappene på en smarttelefon. Det er også endret litt på fargene til knappene, her er noen knapper satt til andre farger enn de andre. Det er gjort for at de skal skille seg litt ut, f. eks "x" er gjort til en annen farge fordi den brukes ofte, og for at den ikke skal forveksles med gangetegnet, som også er en slags "x". Selve teksten på knappene er generert v.h.a. et lite perlskript som kaller LaTeX.[19]



Figur 15: Det endelige brukergrensesnittet

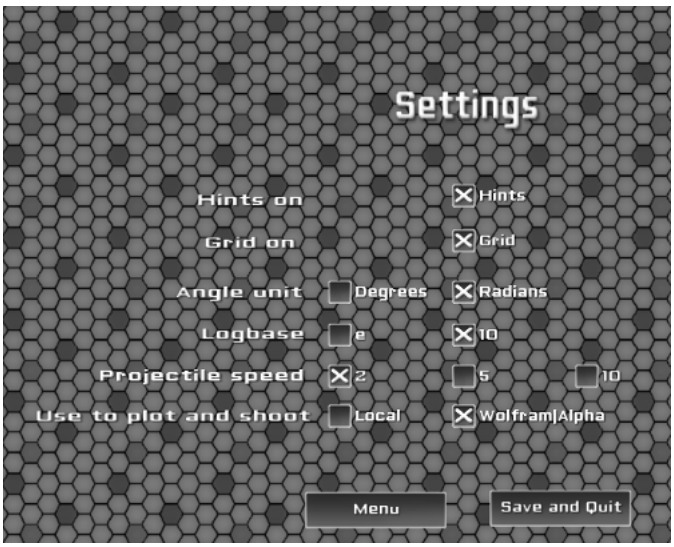
I tilbakemeldinger fra brukerne kom det fort frem at modeller burde være større i selve spillskjermen. Grunnet dette ble det endret på hvordan selve "skyteskjermen" skulle se ut. Kanonen og de andre modellen blir ikke skalert ned og satt i hjørnet av skjermen når kalkulatoren er oppe. Istedet er disse like store hele tiden. Modellene er flyttet litt ned slik at kalkulatoren ikke skal stå i veien. Det samme er skjedd med hint og plot. Dette

brukergrensesnittet ser bedre ut, og du slipper at modellen i vinduet skal endres frem og tilbake hele tiden. Det er også gjort slik at hvis kalkulatoren minskes, så vil også hint og plottet fjernes fra skjermen. Dette kommer tilbake hvis kalkulatoren forstørres igjen.

5.4.1 Innstillinger

Det må være mulig for spilleren å endre på innstillingene. Dette ble gjort ved å legge til en "Options"-knapp på hovedmenyen. Enkelte ting noen liker, kan andre finne irriterende. Under følger en liste over hva spilleren kan endre:

- Om hint skal vises
- Om "Griddet" skal vises
- Om det skal brukes radianer eller grader
- Hvilken logaritme base som skal brukes
- Prosjektilets hastiget
- Om det skal brukes Wolfram|Alpha eller det lokale systemet.



Figur 16: Brukervalg

Grid og hint er det mest synlige spilleren kan endre på. Griddet viser størrelsen på spillebrettet, og gjør det mulig for spilleren å telle seg frem til hvilke konstanter som skal brukes i et uttrykk.

Hint er kjekt å få dersom man trenger litt hjelp, men kan være irriterende for mer avanserte brukere. Det er derfor mulig å slå av dette.

Spilleren har også mulighet til å velge om de trigonometriske funksjonene skal bruke grader eller radianer. Radianer anbefales, ettersom utfordringene er lagt opp for det-

te. Det er upraktisk å bruke grader, ettersom det fører til en veldig lang periode, som strekker seg langt utenfor spilleplanet dersom den ikke endres.

Logaritmebasen kan óg endres. Valgene er 10 og e (2.71828). Det er foreløpig ingen utfordringer som bruker logaritmer, men ingenting hindrer spilleren i å bruke det likevel.

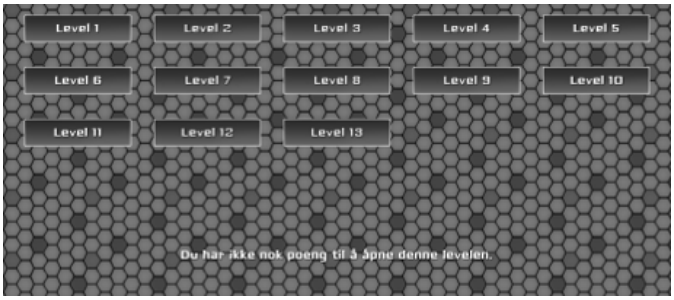
Prosjektilets hastighet bør også kunne endres på. Dette fordi en treg maskin vil bruke lengre tid mellom tegningen av to bilder på skjermen. Ettersom prosjektilets hastighet over skjermen er uavhengig av hvor fort bilder tegnes, og det kun sjekkes for kollisjoner i et bilde, kan prosjektilet på en treg maskin hoppe over en stjerne eller annet. "Hastigheten" man stiller på er egentlig en skalar for tiden, som har direkte innvirkning på prosjektilets X-posisjon.

Muligheten til å velge mellom det lokale systemet og Wolfram|Alpha var noe som måtte være med. Dette fordi Wolfram|Alpha bruker altfor lang tid på å få ut resultater. Det tar ca. 2-3 sekunder å få opp en graf over hvordan funksjonen din kommer til å se ut, og det samme for å få i gang et skudd. Dette føles som ganske lang tid, og når det eksisterer et lokalt system som fikser dette på noen millisekunder, må brukeren få mulighet til å velge selv.

5.4.2 Brett

Det ble mye frem og tilbake på hvordan de forskjellige brettene skulle vises. Først hadde vi det slik at alle brett ble vist på en skjerm med en gang, dette gjorde at det var umulig å vite hvilket temaet de forskjellige brettene på. Når ble det gjort likt andre spill som er lignende vårt som f. eks "Angry Birds", hvor det er en hovedskjerm hvor man kan trykke på de forskjellige verdenene eller temaene, og får da opp hvilket baner på det temaet eller verdenen du har mulighet til å spille.

Versjon 1

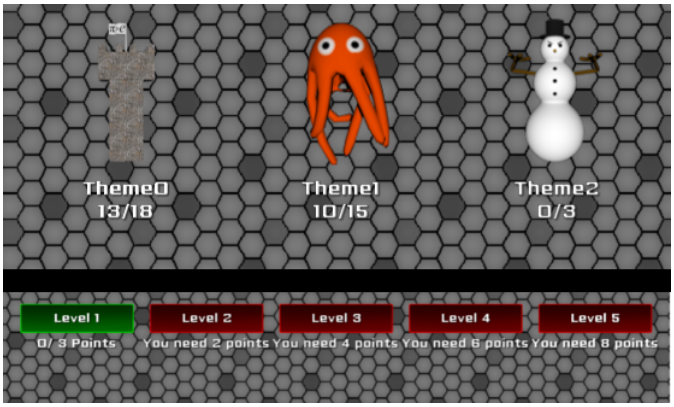


Figur 17: Levelsscreen v.1

I denne versjonen ble det ikke tenkt så mye på hvordan ting gikk opp mot brukeren.

Det ble mest tenkt på at ting skulle fungere. Grunnet dette ble det ikke tatt hensyn til ting som at man f. eks skulle få vite hvor mange poeng man hadde eller hvor mye man trengte for å åpne en bane siden denne versjonen var tenkt å byttes ut uansett. Her fikk man opp en liten tekst boks hvor det stod at man ikke hadde nok poeng når det ble prøvd å åpne en bane som var låst. Dette var en tungvint måte å vise frem at brukeren ikke kunne spille den banen. Det skjedde også ofte at spilleren trykket på baner han/hun ikke hadde tilgang til.

Versjon 2



Figur 18: Levelsscreen v.2

I denne versjonen ble det endret en god del. Først og fremst ble måten baner ble vist på endret ganske mye. Før ble alle baner vist på en scene, mens den nå viser alle de forskjellige "Temaene" på en scene, og når disse blir trykket på kommer det opp en ny scene med alle brettene på dette temaet. Det ble også lagt inn en mulighet for å se hvor mange poeng du har fått på de forskjellige temaene og banene du kan spille, mens det står hvor mange poeng du trenger for å komme til en bane som er låst. De låste banene er nå helt låst. Knappene er røde, og det skjer ingenting når man trykker på dem. De banene man har lov til å spille, vises som grønne.

5.4.3 popupscreens

Det blir brukt flere forskjellige popup vinduer. De blir brukt til å gi forskjellige muligheter til brukeren. Det finnes 3 forskjellige popupvinduer når en bruker har skutt: Ett for når brukeren bommer på målet, ett for når brukeren treffer men ikke har nok poeng for å gå til neste bane, og ett for når brukeren har klart banen og har mulighet for å gå videre. Det blir brukt forskjellige NGUI "skjermer" for hver av disse. I tillegg finnes det en popupskjerm for når hint skal vises. Denne kommer kun opp når brukeren går på en

bane første gang. Hvis brukeren prøver banen på ny ved å trykke “Retry” eller “r” for å starte banen på ny vil ikke hintet vises.

5.5 Lage brett

I starten ble alle brett laget ved Unity sitt "Drag-and-drop" system, dette systemet er veldig lite dynamisk, og du må i tillegg være i Unityeditoren for å ha mulighet for å lage brett. Dette vil si at det er umulig for bruker å lage egne utfordringer, og det er også umulig å legge til brett etter slipp uten å måtte legge ut spillet på ny. Alt i alt ble dette en for tungvint måte å lage brett på, derfor måtte det lages en mer dynamisk løsning.

5.5.1 Lagre til fil

Løsningen ble at vi lagret alle objekter som var på et brett i en XML fil. Denne filen er bygget opp på denne måten:

```
<brett>
  <tema> "tema nummer" <tema>
  <kamera> .. </kamera>
  <lys> .. </lys>
  <Kanon>
    <posisjon>
      x, y , z
    </posisjon>
    <rotasjon>
      x, y, z
    </rotasjon>
  </Kanon>
  <Stjerne>
    x,y
  </stjerne>
  ...
  <Maal>
    x, y
  </maal>
  <grenser>
    <oevre x> 40 </oevre x>
    <nedre x> -40 </nedre x>
    <oevre y> 70 </oevre y>
    <nedre y> -70 </nedre y>
  </grenser>
</brett>
```

Her er tema lagt slik at det blir lagret som et tall, det blir da sjekket opp mot en tabell med alle forhåndsbestemte temaer som er i spillet, hvis ikke teamet finnes blir det satt til standard temaet.

Lys, kamera og kanon blir lagret med posisjon og rotasjon, dette gjøres for å vite i hvilken retning objektet peker. Det er veldig viktig å få med dette, spesielt på kameraet og kanonen. Kameraet kan peke i helt forskjellige retninger på de forskjellige brettene, du kan også skyte i forskjellige retninger, så det er viktig å få med seg hvordan kanonen er rotert.

På lyset er det egentlig kun rotasjonen vi trenger, dette er på grunn av at det er et retningsbestemt lys. Det at et lys er retningsbestemt betyr at det treffer alle objekter likt fra en bestemt vinkel uansett hvor de er uavhengig av lysets posisjon. Posisjon er kun tatt med for at det gjorde kode arbeidet litt lettere, og det gjør ingenting negativt for selve xml-filen.

Med stjerner, antistjerner og andre objekter blir det kun tatt med x og y posisjon som er relativ til foreldrens posisjon som er kanonen. Rotasjon er ikke relevant til andre elementer enn lyset, kanonen og kameraet.

Det lagres også hvilke grenseverdier som prosjektilet som blir skutt ut skal få lov å bevege seg i, det vil si at hvis prosjektilet beveger seg ut av de gitte grenseverdiene vil det bli ødelagt og spilleren må forsøke på ny. Disse verdiene hjelper også til med å lage et rutenett som skal hjelpe spilleren med å beregne prosjektilets bevegelser.

De dataene som blir skrevet til fil blir senere lest fra filen når banen skal spilles. Da går en xml-leser gjennom filen og plasser de gitte objektene på bretter i sin posisjon.

5.5.2 Level Editor

I editoren vår så har brukeren mulighet for å lage egne baner, det er lagt inn flere forskjellige måter å gjøre dette på.

- Du kan legge inn en formel som du har lyst til å lage en bane for , og editoren kan ved hjelp av denne sette inn 3 stjerner og 1 mål i iforhold til denne formelen for deg.
- Du kan selv sette inn i hvilke x verdier du vil ha f. eks en stjerne eller antistjerne og det vil bli satt inn for deg.

Figur 19: Level editor

Det er også mulig å sette x og y koordinater for et objekt som du vil ha i scenen. Når et objekt er satt inn er det mulig å flytte på det ved å trykke på pilene som er på det, du bestemme selv hvor mye du vil flytte objektet for hver gang du trykker på en pil. Hvis brukeren bestemmer seg for at han ikke vil ha objektet med på brettet allikevell er det mulig å slette det enkelte objektet eller å fjerne alle objekter fra scenen ved enten å trykke på krysset på det gitte objektet eller å trykke på "Clear" knappen.

Når du er fornøyd med hvordan brettet ser ut, blir det skrevet til en fil hvor x og y koordinatene til alle objekter er med. Brukeren bestemmer selv hva filen skal hete. Hvis du etter lagring vil dele banen din med noen andre er det bare for deg å sende denne filen til dem. Det er også mulig å gi dem formelen du har brukt slik at de også kan lage og spille brettet.

Selve level editoren fungerer fint, men den har ikke blitt implementert i selve spillet enda. Det står mer om dette under "Videre utvikling".

5.6 Catbox

For å omgå kompatibilitetsproblemer mellom Wolphram Alpha og Unity3D (se side 6) har vi valgt å bruke en reléserver. For å komme ut av Unity3D sin sandkasse trenger vi en ekstern prosess som kjører utenom denne. En sentral server gjør det lettere å rulle ut spillet på nye plattformer.

Vi hadde opprinnelig tenkt å bruke reléserveren til mer. Bl.a. kunne vi redusert responstid ved å mellomlagre resultater, men dette er mot lisensen til WA.[32]

For å få full kontroll over ressursbruk, og av personlig interesse for å eksperimentere med dynamisk linking, har vi valgt å skrive serveren i C. For å kunne opprette tilkoblinger mot internett v.h.a. "sockets" må den kjøre på et operativsystem som implementerer minimum POSIX.1-2001 / IEEE Std 1003.1-2001.[33] Vi har testet den på Linux og FreeBSD.

Vi har delt serveren opp i moduler med adskilte funksjoner. Modulene blir lastet under oppstart med `dlopen()`. En mulighet for utbedring her er å legge til mulighet for å oppdatere moduler mens serveren kjører. Dette ligger utenfor oppgaven.

Serveren kjører i flere tråder; en hovedtrå venter på nye tilkoblinger, og flere arbeidstråder behandler tilkoblingene. Antall tråder er per dags dato hardkodet, men bør kunne justeres uten å måtte rekompilere programmet. Et godt antall tråder er en mer en en tråd per CPU-kjerne på maskinen serveren kjører på. Dette vil redusere antall avbrytelser av tråder som må dele en kjerne.[34] Multitasking oppnås kooperativt ved at en forespørsel som ikke kan bearbeides mer legges tilbake i køen og neste forespørsel bearbeides.

Vi har kun implementert funksjonalitet som vi selv har bruk for. Dette betyr at vi ikke har en komplett webserver, bl.a. forstår serveren kun HTTP GET-forespørsler (side 53).

5.6.1 Moduler

libcatbox
Denne modulen inneholder kjernefunksjonaliteten i serveren som tolking av en HTTP-forespørsel og svar på denne. Dette er den eneste modulen som linkes til under kompilering.

fileserv
Dersom stien til en forespørsel ikke er lik navnet på noen andre moduler, antas det at det er en fil. Fileserv-modulen leter i mappen "www" etter en fil med dette navnet. Dersom den finnes blir den sendt, ellers sendes en 404 feilmelding.

For å spare minne legges filen i prosessens adresseminne. Da trenger operativsystemet bare å laste de delene av filen som blir lest inn i minnet, og minne som ikke blir

lest frigjøres. Dette gjør også at operativsystemet kan gjenbruke minne når vi har flere forespørsler etter samme filen.

relay
En forespørsel på formen `www.example.com/relay?url=[adresse]` vil bli håntert av relay-modulen. Den forsøker å opprette en tilkobling til forespurt adresse og videresender all data lest fra denne adressen til klienten helt til en av partene lukker tilkoblingen.

makepng
Denne modulen oppretter en tilkobling til en vilkårlig adresse på samme måte som relay. Den laster ned innholdet, tester om det er et bilde og konverterer i så fall til png-format. Konvertering skjer ved `convert` kommandoen fra ImageMagic.[35]

For å slippe å skrive bildet til en midlertidig fil i filsystemet sendes/leses bildene til/fra `convert - png:-` som leser og skriver til en bytestrøm.

6 Testing

6.1 Webplayer

Da spillet begynte å ta form ble det bestemt at det trengtes betatestere. Det begynte å bli stort behov for dette. Prosjektet begynte å gå litt tregt, og vi følte at vi måtte ha noe ny input for å få skikkelig gang på ting igjen. Det ble lagt ut fungerende versjoner av spillet på blant annet Linjeforeningen til Data og IT og spillmakerlauget, sammen med et googledokument med en del spørsmål de kunne svare på. Totalt var det omtrent 300 personer med i disse gruppene. Dette er betatest nummer 1. I denne testes kun spillet og hvordan brukergrensesnittet er. Det blir ikke tatt noen testing av Wolfram|Alpha opp mot vårt lokale system, dette var mye grunnet at unity sin webplayer ikke ville ha noe med Wolfram|Alpha å gjøre (Senere ble det funnet ut at dette ikke fungerte fordi Dropbox ikke likte oppkoblingen til Wolfram, det var ikke Unity sin feil).

Spørsmålene som ble stilt i denne testen er listet under:

- Hvordan var ditt førsteinntrykk av spillet?
- Hva syns du om brukergrensesnittet? Forslag til forbedring?
- Hva kan gjøres for å forbedre spillet? Noe du savner? Noe som er irriterende?
- Ikke-fungerende formler?
- Plutselige krasj?
- Annet?

Ved å stille disse spørsmålene håpet vi at det skulle komme noen nye svar. Kanskje var brukergrensesnittet vårt for dårlig, banene var kanskje for lette eller for vanskelige. Det var rett og slett umulig å få en god pekepinn på hvordan spillet lå an og om det var bra før noen utenforstående hadde prøvd det skikkelig. Det kom mange tilbakemeldinger fra testerne og vi fikk fikset bort en del bugs. Tingene som ble fikset etter denne betatesten var:

- "(a)x" ble tolket som en konstant i parseren
- "y=x" ble ikke godkjent av parseren. Nå ignoreres alt til venstre for et eventuelt likhetstegn
- Hintet kom opp hver gang man prøvde en bane på ny
- Temaskjermen ble ødelagd etter at vi la navn på dem
- Fjernet "Quit" fra menyen på webplayer. Denne hadde ingen virkning her
- Endret brukergrensesnitett i spillskjermen

6.2 Wolfram|Alpha vs Lokal

I den første betatesten ble det kun sett på selve spillet, men det var også behov for å teste hvordan brukerne reagerte på ventetiden til Wolfram|Alpha. Derfor måtte vi ha en ekstra betatest. I denne betatesten ble det rigget opp en stand i inngangen til Høgskolen i Bergen. Her satt vi opp to maskiner med spillet installert; den ene med kun lokalt system og den andre med Wolfram|Alpha. Vi håpet at testerne kunne gi oss en liten pekepinn på hvordan de opplevde ventetiden fra Wolfram|Alpha.

Under utførelsen lærte vi flere ting, det viktigste var at hvis testerne fikk teste både Wolfram|Alpha og det lokale systemet, reagerte de fleste på at Wolfram|Alpha var tregt. De som ikke hadde så god kontroll på matematikken i spillet reagerte ikke noe særlig på dette, men de som tok spillet med en gang reagerte ganske fort. Når folk kun fikk testet Wolfram| var tilbakemeldingene blandet. Omtrent en tredjedel av de som testet reagerte på at ting gikk tregt, mens de resterende ikke reagerte noe særlig. De som reagerte var de som faktisk syntes spillet var gøy og interresant, så vi føler derfor at den lokale versjonen er noe som må være med i sluttproduktet.

Det kom også frem at store deler av testgruppen ikke leste hintene. Om dette var fordi hintene ikke var godt nok synlige, eller om det bare var latskap, er noe som må bli sett nærmere på. Testerne la ikke merke til hintet som stod i en tekstboks nede i venstre hjørne før vi gjorde dem oppmerksomme på dette. Et forslag til løsning var at det kunne bli satt opp en knapp på kalkulatoren hvor det stod “hint”. Denne ville da gjøre at popuphintet som kommer i starten av banen vil komme opp på ny. Eventuelt kan hintboksen i nedre venstre hjørne gjøres mer synlig på en måte, f.eks. ved å gjøre teksten rød. I tillegg har hintet som kommer opp i begynnelsen fått en mørk bakgrunn, så det er mer synlig.

Flere testere reagerte også på at det første temaet var litt kjedelig. Som en respons på dette ble skyteskiven i dette temaet endret.

Totalt sett var det mange gode tilbakemeldinger i begge testene, det var noen som ikke likte spillet, men dette var forventet. Det var faktisk flere som sa at dette var et spill som de kunne tenkt så å spille videre enn folk som ikke viste interesse for det. Under testen ved inngangen til skolen var det faktisk en del personer som nesten spilte gjennom hele spillet før de ga seg. Alt i alt tjente spillet mye på disse testene. Det ble luket vekk en del feil fra parsetreet, brukergrensesnittet ble bedre, mistanken om at Wolfram|Alpha var for tregt ble bekreftet og det ble også luket vekk en del små fikser her og der i tillegg.

7 Prosjektledelse/-styring

Gruppen har ikke hatt en formell rolleinndeling, men en helt flat gruppestruktur. Vi var fra starten av enige om å møtes hver dag og arbeide fulle arbeidsdager. Dette har gjort at vi har hatt mulighet til å ta alle avgjørelser i plenum.

Vi satte oss ned i starten og kartla hva vi måtte gjøre og satte opp en tidsplan for dette. Se ganttdiagram side 55. Deretter satte vi av den første uken til å leke med verktøyene vi kom til å bruke for å bli bedre kjent med dem. Etter denne første uken hadde vi allerede en enkel prototype til spillet.

Vi gikk inn i en arbeidsflyt der vi fordelte oppgaver mellom gruppemedlemmene. Primært arbeidet en person om gangen med en komponent, men vi konfererte hyppig med hverandre ettersom vi satt i samme rom. Etterhvert som et gruppemedlem fullførte en oppgave tok vi en felles avgjørelse på hva som var neste naturlige arbeidsoppgave.

Gruppen har gått inn i en veldig agil arbeidsmetode uten at vi har brukt tid på å formalisere dette eller i det hele tatt vært særlig bevisst på det.

8 Oppsummering og konklusjoner

Vi er veldig fornøyde med prosjektet som helhet. Vi er spesielt fornøyde med at vi har klart å lage så mange ekstra ting til oppgaven som vi har gjort. Spillet begynner å nærme seg et ferdig produkt som det vil være mulig å lansere i stor skala. Samarbeidet innad i gruppen har vært veldig bra fra dag én, og dette har vært en viktig grunn til at prosjektet har blitt gjennomført så bra. Kommunikasjonen med både veilederne og oppdragsgiveren har vært veldig god hele veien. Pål Trefall har vært tilgjengelig til alle døgnets tider og har prøvd å hjelpe med alle utfordringer vi har møtt. I tillegg stilte han villig opp på et introduksjonskurs til Unity for oss, som vi alle trengte. Veilederne Harald Soleim og Jon-Eivind Vatne har hjulpet oss gjennom hele prosjektet ved å komme med innspill til hva som trengs i spillet, og hva som må forbedres i rapporten.

Vi var veldig fornøyde med oppgaven vi fikk. Den utfordret oss på både det intellektuelle og det kunstneriske planet. Vi mener at løsningen vi har kommet opp med er veldig god, og selve sluttproduktet har et høyt nivå. Spesielt med tanke på tiden vi hadde tilgjengelig, og at det er vårt første prosjekt i Unity 3D. Det finnes fortsatt en del ting som kan og bør endres på. Dette blir sett på senere i rapporten.

Det å arbeide med Unity har vært svært lærerikt. I begynnelsen var det uvant, og vi strevde med å få til selv de enkleste ting. Etterhvert som vi fikk mer erfaring, gikk ting langt lettere, og vi oppdaget fort at våre tidligere løsninger var overkompliserte og dårlige. Koden i spillet bærer litt preg av dette. Vi har skrevet om de fleste skriptene, men det henger fortsatt igjen litt gamle løsninger. Disse er ikke dårlige, men det er mulig å gjøre de bedre.

Integrasjonen av Wolfram|Alpha i Unity har også vært en interessant prosess. De to produktene ønsker å gjøre ting på sin egen måte, og dette har ført til at vi har måttet tenke utenfor (sand)boksen, som igjen har ført til at vi har måttet gå ut på dypere vann enn vi hadde tenkt. Ingen på gruppen hadde særlig kunnskap om serverprogrammering, men det var tydelig at vi måtte ha en server for å omgå Unity Webplayers sandkasse.

Alt i alt har det vært et utfordrende prosjekt. Vi har fått mange nye erfaringer innenfor flere områder som vi kan ta med oss videre.

9 Videreutvikling

Det finnes et par ting som ikke er helt ferdig enda:

9.1 Lydeffekter

Spillet har kun lydeffekter på knappene i menyen. Lydeffekter på handlingene i spillet ville gjort at spillet oppleves mer komplett. Å spille av lydeffekter i Unity3D er relativt enkelt men det trengs en samling av lydeffekter som passer til de ulike temaene.

9.2 Nettsjekk

Det må sjekkes om brukeren har internettilgang, eller om serveren er nede. Hvis brukeren prøver å spille med Wolfram|Alpha aktivert, vil ikke spillet gjøre noe fornuftig hvis det ikke har nettilgang. Dette er i teorien en enkel fiks. Det eneste som må gjøres er å spørre etter en fil, f.eks. "Crossdomain.xml", fra serveren. Hvis ikke brukeren får tilgang til denne er det enten et problem med nettilkoblingen, eller så er serveren nede. Da skal den automatisk bruke den lokale parseren.

9.3 Android

Det er et lite problem med skjermstørrelser når spillet blir kjørt på mobile platformer. Mobilskjermer er små sammenlignet med PC-skjermer, men har ofte samme oppløsning. Ettersom størrelsen på knappene i spillet er oppgitt i antall piksler, vil knappene bli veldig små. Dette må fikses. Det er mulig å gjøre dette ved at alt som vises i spillet er relativt til skjermstørrelsen.

9.4 Appleprodukter

Selve spillet skal i teorien fungere på disse produktene, men det er ikke mulig å teste til Mac uten tilgang til en maskin med OS X. Dette er noe vi ikke har. Det er også umulig å lage spillet til iOS uten en Applemaskin.

9.5 Brukergenerte brett

Selve leveleditoren fungerer helt fint, men den ser ikke så bra ut for øyeblikket. Det er heller ikke lagt inn mulighet for å legge til egne brett og spille disse. Det er kun mulig å spille de brettene som følger med. Mulighet for å dele de brettene man lager er også noe som må settes på plass.

9.6 Flere temaer

Selve spillet kommer aldri til å bli ferdig, det er bare fantasien vår som setter grenser på nye temaer og brett. For at spillet skal være tiltrekkende over tid så må det komme nye brett. Dette er ikke noe stort problem. Selve systemet tar lett imot nye brett og temaer. Problemet er at ingen av gruppemedlemmene er særlig kreative.

9.7 Flere brett

Vi har mange idéer til flere brett, men vi mangler temaer å putte de inn i. Vi tenkte å lage et tema hvor man må bruke all matematikk man kan. Fase- amplitude-, frekvens- og kvadraturmodulasjon av diverse periodiske bølger. Røtter, logaritmer og annet kan også brukes.

9.8 Online tospillerdel

Tospillerdelen er nå kun lokalt på en maskin. Begge spillerne må dele den samme maskinen for å spille sammen etter tur. Muligheten for å spille over nett mot venner og ukjente vil gjøre spillet mye mer attraktivt.

9.9 Andre typer funksjoner

Parametriserte funksjoner, hvor man skriver en funksjon X(t) og Y(t), vil gi mulighet for langt mer avanserte brett. Med komplekse tall kan vi lage brett hvor man skyter to prosjektiler, enten på samme eller forskjellige spilleplan. Polare koordinater, sfæriske koordinater og ikke-euklidske spilleområder. Funksjonstypene vi teoretisk kan implementere er endeløse.

9.10 Profesjonell grafisk designer

Folk som programmerer tenker vanligvis logisk, ikke kreativt. Dette ser man ofte i grafikk og brukergrensesnitt (“programmer’s art”). Det å få en profesjonell grafisk designer til å lage modeller, bakgrunner og grensesnitt ville løftet det visuelle betraktelig.

10 Betydning for SIMSubsea

De kan sannsynligvis ikke bruke våre løsninger direkte, men de kan tilpasse våre løsningsmetoder til sitt prosjekt. Vår måte å håndtere wolfram på er ganske generell og kan enkelt tilpasses andre typer anvendiner av WolframAlpha. SIMSubsea kan også lære av de problemene vi har møtt, for eksempel det faktum at wolfram kun gir fra seg GIF og Unity kun tar imot JPG/PNG var et uforutsett problem. Det kan også hende at SIMSubsea kan ved hjelp av vårt prosjekt finne ut at WolframAlpha ikke er kjapt nok for deres prosjekt. En forsinkelse på 2-3 sekunder er ganske betydelig, spesielt dersom spilleren skal gjøre mange utregninger. Det er usikkert på hvor mye dette har å si for SIMSubsea.

Referanser

[1] "SIM Subsea." http://www.simsubsea.no.

[2] "Wolfram|Alpha." http://wolframalpha.com.

[3] "Unity 3D." http://unity3d.com.

[4] "Worms." http://worms.team17.com/.

[5] "Scorched Earth." http://en.wikipedia.org/wiki/Scorched_Earth_(video_game).

[6] "Angry Birds." http://www.rovio.com/en/our-work/games/view/1/angry-birds.

[7] "Wolfram|Alpha Webservice API Reference." http://products.wolframalpha.com/api/documentation.html.

[8] "Unity 3D – WWW." http://docs.unity3d.com/Documentation/ScriptReference/WWW.html.

[9] "System.Xml." http://msdn.microsoft.com/en-us/library/system.xml(v=vs.80).aspx.

[10] "Unity 3D – SecuritySandbox." http://docs.unity3d.com/Documentation/Manual/SecuritySandbox.html.

[11] "System.Drawing." http://msdn.microsoft.com/en-us/library/system.drawing.aspx.

[12] "Graphics Interchange Format." http://www.w3.org/Graphics/GIF/spec-gif87.txt http://www.w3.org/Graphics/GIF/spec-gif89a.txt.

[13] "Mono Develop." http://monodevelop.com.

[14] "Eclipse." http://www.eclipse.org/.

[15] "Python." http://www.python.org/.

[16] "Javascript." http://en.wikipedia.org/wiki/JavaScript.

[17] "C#." http://msdn.microsoft.com/en-us/library/vstudio/kx37x362.aspx.

[18] "Git." http://git-scm.com.

[19] "LaTeX." http://www.latex-project.org.

[20] Wikibooks, "Wikibooks LaTeX." https://en.wikibooks.org/wiki/LaTeX/, 2013.

[21] "Syntakstre." http://en.wikipedia.org/wiki/Concrete_syntax_tree.

[22] "Parsing." http://en.wikipedia.org/wiki/Parser.

[23] "Battleship." http://en.wikipedia.org/wiki/Battleship_(game).

[24] "Ekstra liv." http://en.wikipedia.org/wiki/Extra_life.

[25] "Blender." http://www.blender.org/.

[26] "Autodesk maya." http://www.autodesk.com/products/autodesk-maya/overview.
[27] "3ds max." http://www.autodesk.com/products/autodesk-3ds-max/overview.
[28] "Fbx filformat." http://www.autodesk.com/products/fbx/overview.
[29] "Quadratic interpolation." http://en.wikipedia.org/wiki/Simpson's_rule#Quadratic_interpolation.
[30] "OnGUI." http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.OnGUI.html.
[31] "NGUI." http://www.tasharen.com/?page_id=140.
[32] "Wolfram|Alpha Webservice Terms of Use." http://products.wolframalpha.com/api/termsofuse.html.
[33] "IEEE Std 1003.1." http://pubs.opengroup.org/onlinepubs/007904975.
[34] "Sizing Thread Pools." http://codeidol.com/java/java-concurrency/Applying-Thread-Pools/Sizing-Thread-Pools/, Mars 2008.
[35] "ImageMagic." http://www.imagemagic.org.

A Ordliste/forklaringer

HTTP Hypertext Transfer Protocol, en nettverksprotokoll for overføring av informasjon over verdensveven. HTTP er en klient-server modell der klienten kobler seg til serveren og sender en forespørsel etter en resurs. Serveren vil da sende tilbake resursen eller en feilmelding dersom den ikke eksisterer.

IDE Integrated Development Environment, koderedigeringsverktøy med andre funksjoner.

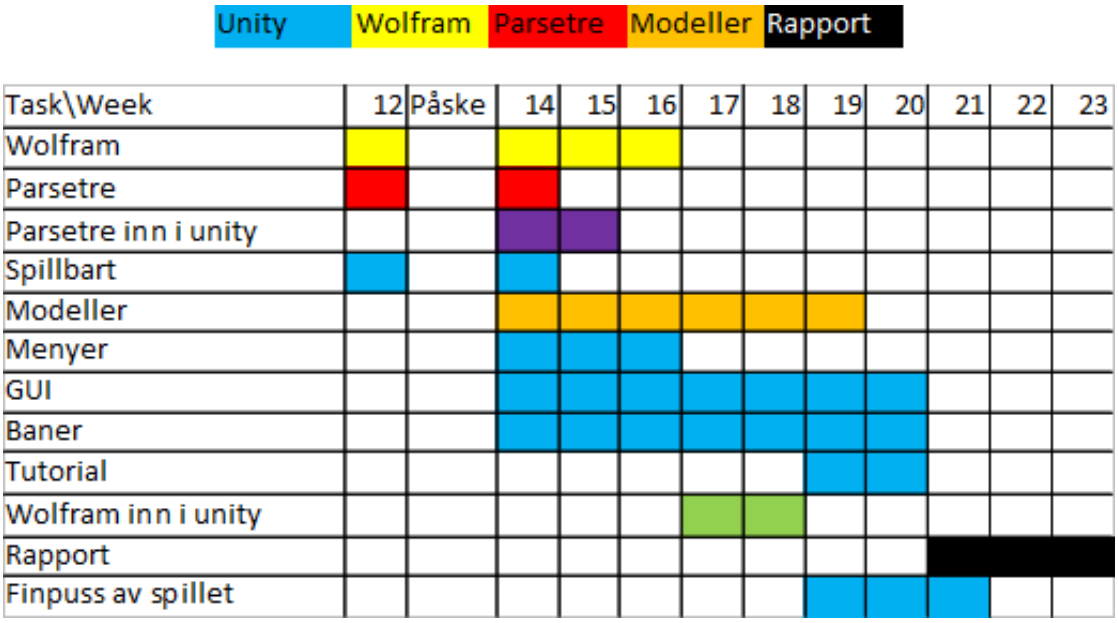
WA Wolfram|Alpha, en søkemotor på internett. WA forsøker å tolke en søkestreng i naturlig språk og generere svaret v.h.a. algoritmer i stedet for tradisjonell indeksering av søkeord.

B Risikoliste

Problem	Risiko	Konsekvenser	Hvordan hindre
Wolfram|Aplha vil ikke fungere. Dette er den viktigste delen av oppgaven. Oppdragsgiver er interessert i måter Wolfram|Alpha kan integreres med Unity 3D.	Stor	Store	Jobbe kontinuerlig og målrettet med dette fra begynnelsen.
Manglende erfaring med unity. Ingen på gruppen har jobbet med unity eller spillprogrammering generelt tidligere. Manglende erfaring fører til mye tapt tid på å lese dokumentasjon og prøving og feiling.	Medium	Medium	Ikke ta snarveier. Leke med unity i starten
Forsømming av rapportskriving. Det er fort gjort å glemme at det rapporten også må gjøres når det er så mye annet viktig.	Medium	Store	Sette av minst en dag i uken til å jobbe med rapporten

Vi har klart å komme oss rundt de fleste risikoene som var satt opp. Wolfram|Alpha ga oss litt problemer, men når vi fikk opp en server slik at vi kom oss ut av sandkassen til unity så gikk dette ganske greit. Rapport skriving har vi klar å gjøre litt hver uke, slik at det ikke har blitt et stort skippertak på slutten. Det eneste vi faktisk har slitt litt med er Unity. Dette har gjort at vi har laget litt tungvinte løsninger, og kanskje ikke gjort ting på den beste måten i starten. Men etter hvert som prosjektet gikk sin gang har vi fått mer kontroll og vi føler at vi har funnet gode løsninger.

C Fremdriftsplan



Figur 20: Ganttdiagram

Dette var Ganttdiagrammet som vi lagde den første uken. Vi har klart å holde dette ganske greit, det eneste vi ikke har holdt det er finpussing av spillet. Finpuss av spillet kommer det til å bli jobbet med helt til siste dag, dette er på grunn av at det er mange små ting som må endres og det faktum at vi sannsynligvis aldri kommer til å bli 100 % fornøyd med hvordan spillet fungerer.