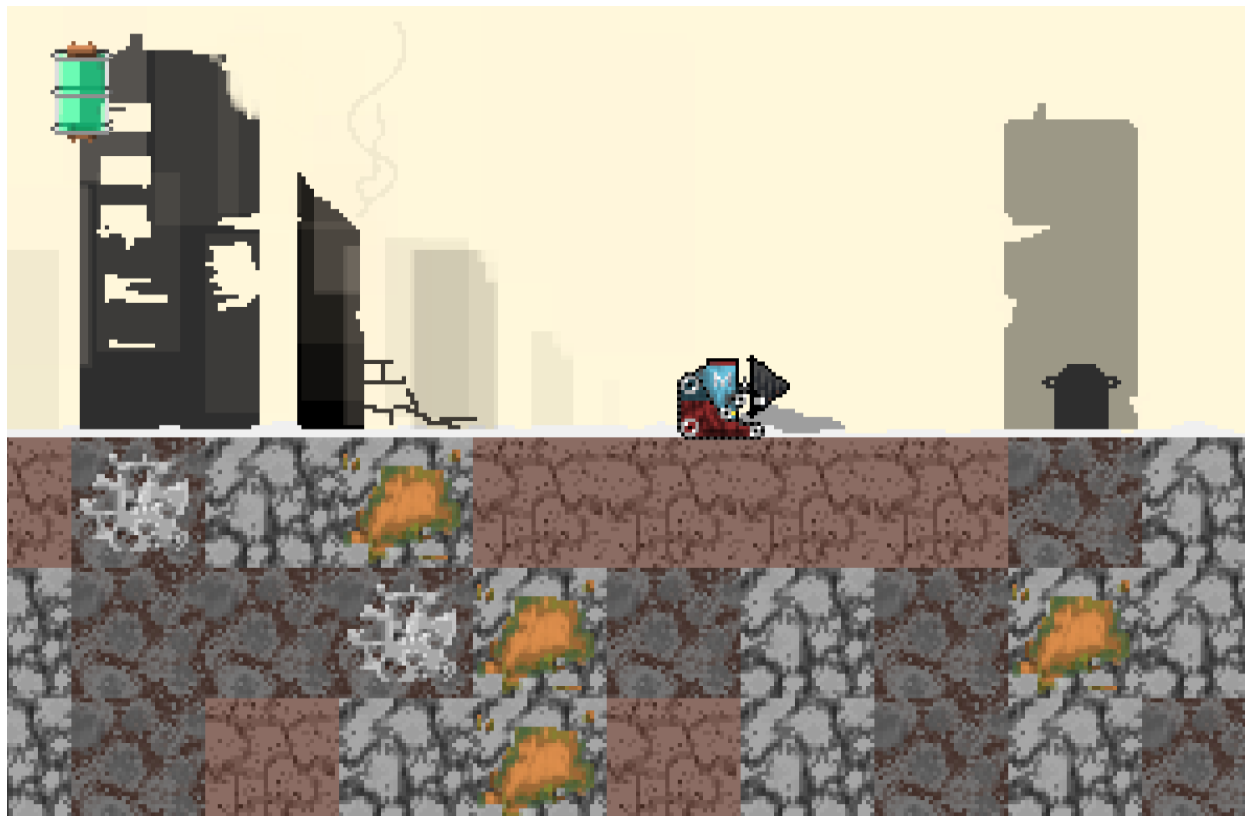


# Maersk Miner af Scrap Worthy



---

## Første Semester Prøve - Rapport

**Bjarke Alexander Reffstrup, Thorbjørn Saxe Dietrich og Thorbjørn Ulrik Mortensen**

**Vejleder: Kenneth Røjkjær Andersen, Mikael Tranekjer Debel Thomsen og Jan Philip**

**Tsanas**

**Antal anslag: 59,344**

**Afleveres d. 15-12-2022**

---

## Indledning

I denne rapport beskriver vi processen til, hvordan vi udviklede spillet Maersk Miner med udgangspunkt i opgavebeskrivelsen. Dette er vores endelige produkt, som er et spin off af det gamle, gratis spil "Motherload", som de fleste fra 00'erne husker.

Vi har derudover bagvedliggende valgt, at vi som firma har til mål, at give nyt liv til ældre web-baserede spil fra vores barndom. Dette valg grunder i at mange moderne spil ligner mere og mere hinanden. Her følger vi nostalgien og genskaber spillene, som brugerne troede de havde glemt. Vores målgruppe vil derfor være gamere i alle køn, i aldersgruppen 20-26. Denne gruppe vil vi senere referere til som Y8-generationen, da gratis spil på hjemmesiden Y8, dengang, havde sit peak.

## Indholdsfortegnelse

<b>Indledning</b>	2
<b>Problemformulering</b>	3
<b>Metoder - Ulrik</b>	4
<b>Idégenererings fase - Saxe og Ulrik</b>	5
Gameplay - Ulrik og Bjarke	7
Udformning	7
Historien - Saxe	8
<b>Business plan - Bjarke og Ulrik</b>	8
<b>Motherload som brugeroplevelse - Bjarke</b>	13
<b>Content Analyse</b>	15
MDA -Saxe og Bjarke	15
<b>Objekt Orienteret Analyse</b>	16
Systemafgrænsning - Saxe	16
Use cases - Saxe	17
Funktionsliste	20
Klassediagram - Saxe	21
SystemDefinition - Bjarke	21

<b>Objekt Orienteret Design</b> - Saxe, Ulrik og Bjarke	23
Første UML Diagram	23
Endelig UML Diagram	24
<b>Objekt Orienteret Programmering</b>	26
Implementering	26
Animation - Bjarke	26
Inventory - Ulrik	29
Lyd - Saxe	33
World Terrain - Saxe	34
<b>Hvordan den er skrevet op</b>	34
<b>Hvordan virker chunksne</b>	35
<b>Get info fra Terrain klassen</b>	38
<b>Player Terrain Interactions</b>	39
User Interface	40
Balancing - Bjarke	40
<b>Konklusion</b> - bjarke	41
<b>Litteraturliste</b>	42

## Problemformulering

Vi har valgt at arbejde med at give nyt liv til ældre spil, som man har kunnet finde i sin barndom på internettet. Vi holder primært fokus på spil fra omkring år 2000 og op efter.

- Hvilke konkurrenter har vores spil Maersk Miner, og hvor gode er vores chancer på markedet?
- Hvilke elementer fra det gamle spil, Motherload, skal vi holde fast i, så det stadig giver en følelse af nostalgi, og hvad kan vi tilføje eller fjerne for at give en bedre oplevelse i Maersk Miner?

## Metoder - Ulrik

I denne rapport har vi valgt at gøre brug af et par forskellige modeller som SMART modellen, MDA, Use Cases og funktionsliste.

SMART<sup>1</sup> modellen står for Specific, Measurable, Achievable, Relevant og Time Bound.

Grunden til at vi bruger smart, er fordi det er et simpelt redskab, som giver os mulighed for at gå i dybden og optimere vores mål til spillet baseret på nogle grundlæggende kriterier.

Specifik handler om at være så præcis og konkret som muligt med sit mål. Her skal man skrive hvorfor målet er vigtigt og hvad for nogle grænser der kan være involverede.

Measurable handler om hvordan man ville finde tal på dette Specifik-mål og hvornår man er færdig med at samle informationer.

Achievable handler hvordan man kan færdiggøre sit mål. Hvor realistisk er det mål, man har sat sig for? Her kunne et eksempel være om man overhovedet har økonomien til at udføre opgaven.

Relevant handler om hvad målet betyder for dig, om det giver mening at lave beregningerne nu, eller om det er mere relevant senere. Passer målet med hvad der er brug for, og er det værd at investere ressourcer i?

Time Bound handler om hvor lang tid man har til at opnå sit mål. Her skal man estimere en tidsplan, og sætte en deadline fra starten af.

Her har vi brugt SMART til at lave fire mål, som generelt alle sammen handler om, hvor mange spillere vi har. Det er gjort sådan, for at finde ud af hvor mange penge vi kan indtjene, og om vores firma kan leve på denne forretningsmodel.

MDA<sup>2</sup> står for Mechanics Dynamics Aesthetics, som er en måde at opdele spil i 3 styrende grupper. MDA kan ses som en formel, hvor man arbejder i hver kategori, for bedre at kunne udvikle et spil fra forskellige perspektiver. Første kategori er Mechanics, som handler om de ting brugeren ikke kan se. Her er der tale om den kode og de fysiske regler som, som udvikleren af spillet har sat. Den anden kategori er Dynamics som handler om hvad der sker når spilleren udfører en handling, f.eks. giver et input på "Space" så i mange spil så gør karakteren i spillet

---

<sup>1</sup> Anders V. Pedersen (ukendt)

<sup>2</sup> Marco (Ukendt)

noget. Denne kategori kan betragtes som et samspil mellem bruger og udvikler. Den tredje er Aesthetics, som handler om, hvordan spilleren ser verden omkring sig inde i spillet. Dette gøres ved brug af grafik eller lyd, til at give spilleren et specifikt indtryk eller -følelse. Dette gælder også historie og quests. Vi har valgt at gøre brug af MDA i vores produktion, da det giver os en bedre fornemmelse af, hvordan spillet opleves for brugeren, og hvad vi kan gøre for at tage højde for dette perspektiv. Det gør det også nemmere at arbejde sammen om spillet som hold, da alle udviklere er på samme page, omkring hvad der skal arbejdes på. Vi har i spillet haft fokus på, at Dymacis og Aesthetics skal føles godt og se godt ud. Disse to elementer vægter højt for vores spil, da vi prøver at ramme en følelse af nostalgi.

Use Cases bliver brugt til at kunne beskrive en specifik del af spillet. Hvis der er ting, som deler samme funktion, går det ind under samme use case. Grunden til vi har brugt use case, er for nemmere at kunne opbygge, hvordan de forskellige elementer spiller sammen. Derudover skaber det visuelt overblik at have diagrammet at falde tilbage på.

Funktionsliste er en tabel over alle de forskellige funktioner, man har afklaret om hvordan spillets hændelser og interaktioner, som skal spille sammen. Her giver en funktionsliste et overblik over koden, og skaber rækkefølgen af hvilke dele der har førsteprioritet. Vi har gjort brug af dette til vores spil af samme grund.

## **Idégenerings fase - Saxe og Ulrik**

Spillet skal kunne blive lavet på relativ kort tid. Hvor der maks er fire uger til at blive færdig med et spil, som helst skal være spilbart. Derfor er der nogle ting som vi kan udelukke fra idégeringen, ting som multiplayer og et spil med 3D elementer. Grunden til disse ting er blevet valgt fra er fordi koncepterne ville være for svære at lave på mængde af tid der er blevet givet til os.

Vi laver en product vurderingen, hvor vi har brainstormet nogle idéer og så skrevet dem ned. Her vil vi vurdere alle idéerne på fem kriteriere. Alle kriterier har et nummer der står i parentes, de nummer går fra et til fem alt efter hvor vigtig vi synes den er. Nummeret bliver så ganget den

vurderet score som gå fra et til ti. Denne score bliver så givet for hver en enkelt og til sidst ganget med det nummer som kriteriet har og så for vi summen af det hele. Det spil med den højeste sum bliver spillet som vi laver.

Den første ting vi vurdere på er mængde af genbrugeligt mæssigt kode fra andre projekter. Vi giver genbrug mæssig kode 5 points, for vi ønsker ikke at lave basal kode igen og igen fra hvert projekt, så det er bedre at mindske tiden forbruget hvis vi kan tage noget vi har lavet tidligere og så bare lave lidt om på koden så den passer i konteksten.

Den anden ting vi vurdere er. Er der nok kode i spillet for os at kode? som for en point mængde af 4, da der skal en større mængde af kode til som vi er tre personer. Her vil man næsten altid kunne finde på nye dele af spillet som laves. Men det er vigtig at der er mulighed for at man kan lave udvidelsere til spillet.

Den tredje ting vi vurdere for en point mængde af 5, da tungere kode som vi både lære nyt af og lære bedre at forklare er en stor del af formålet af projektet. Hvis koden skulle sælges eller lignende, så ville dette punkt være meget mindre givende.

Den fjerde ting vi vurdere for en point mængde af 3, da produktet ikke skal være færdig uden bugs til afleveringen dato, men vi skal selvfølgelig stadig have noget at fremstille, dermed for dette del af skemaet, ikke full points.

Produkt-ide	Mængde af Genbrug Mæssigt kode fra andre projekter (5)	Er der nok indhold i spillet for os at kode (4)	Har spillet tungere kode som kan lære os nyt (5)	Vil vi kunne nå at lave spillet færdig inden for tid (3)	Sum
Wave Defence	7	6	7	7	129
Platform Fighter	2	7	8	6	96
Miner Game	10	6	9	8	143

Factorio lignende	6	10	9	6	133
Race game	2	5	6	9	87

Igennem resultatet så er vi kommet frem til at spillet skal handle om en type Miner Game

### **Gameplay - Ulrik og Bjarke**

Spilleren kommer til at kunne bevæge sig mellem to forskellige dele af samme verden. Første del af verden er over jorden, hvor man kan opgradere sig selv eller komme af med alle de ressourcer som har samlet op på vejen. Den anden del er verden er under jorden, hvor man kan grave rundt efter ressourcer. Her er det meningen at spillet skal blive loaded ind i chunks, så man kan få en større verden efter behov. Når man er under jorden begynder man at miste brændstof, hvilket resulterer i at man kun har en vis mængde tid, før man løber tør og taber spillet. Når man miner igennem jorden, vil man på et tidspunkt støde på forskellige ressourcer, som man kan samle op. Efter man har samlet nok op, kan man bruge ressourcerne til at opgradere sin maskine. Dette loop bliver kørt igen og igen indtil man har den sidste opgradering.

### **Udformning**

- Progression
  - Spilleren starter i toppen og slut målet er at nå ned til de dybder som giver de mere værdifulde ressourcer.
  - Spilleren kan opgradere sig selv ved hjælp af disse ressourcer.
  - Større solcelle, kraftigere bor, bedre batteri er de forskellige opgraderinger som kan tilkøbes, udover den afsluttende AI udvikling, som vinder spillet.
- Liv
  - Mister man alt sit energi i batteriet dør man automatisk og skal starte forfra.
  - Man kan få energi tilbage ved at være i solen på overfladen.

### **Historien - Sakse**

Årstallet er 2089 og du er robot nr. 9989 der er lavet af MAERSK der skal ned til jordens kerne og mine noget super duper metal (også kendt som SDM) til deres nye rumskibe. Det øverste lag af jorden er blevet totalt tømt af menneskene mange år deres ukontrollerbar forbrug af jordens metaller. MAERSK personalet der har designet dig uden at havde taget højde for at det bliver svære og svære at grave jo længere man bevæger sig nedad. Så du har brug for at anskaffe mange forskellige metaller , så du selv kan udstyre dig med bedre udstyr så du kan nå ned til kernen og anskaffe SDM. Robotten når dog at opdage inden, med sine voksende kundskaber, at der ikke længere er nogen mennesker tilbage at grave for.

### **Business plan - Bjarke og Ulrik**

Scrap Worthy vil primært have fokus på at lave indé games. Her vil der bliver taget fat i gamle indie spil fra omkring 2000 - 2010 og give dem nyt liv. Dette vil vi gøre ved at holde fast i kerne værdien af spillet, så man oplever samme følelse og stemning når man prøver den nye version, som ved det gamle. Ved at implementere nye elementer og grafik til spillene, kan vi sørge for ikke at lave en kopi af spillet, som mange har spillet før, men give spilleren en nye oplevelse ,udover det som de har prøvet før.

Dette er et stykke arbejde vi har valgt, fordi vi elsker den nostalgi man får, når man tænker på- eller spiller et spil, som man prøvede dengang man var barn. Vi vil som start lave spillet til computer, hvorefter det vil blive udgivet på telefonmarkedet, da teknologien nu er avanceret nok til nemt at kunne køre ældre spil, som før var for tunge.

Scrap Worthy vil arbejde som freelancer firma, for at tjene penge til senere at kunne lave spil på fuld tid. Når vi har penge nok til at stå alene, vil vi gå efter at få grants og en publisher som et sikkerhedsnet.

### **Platform - Ulrik og Bjarke**





Maersk Miner vil til at starte med kun blive udgivet på computer. Her vil spillet blive udgivet på Steam og Humble Bundle som primær platform. På Steam skal man betale for at få sit spil på platformen for 100 USD. På Humble Bundle skal man udfylde en form, hvorefter de vurderer om spillet er godt nok til at blive lagt op. Derudover vil man kunne finde en demo version af spillet på Itch.io så brugeren kan spille spillet før det endeligt køber. Dette giver også mulighed for at modtage feedback før spillet bliver udgivet. Hvis spillet herefter klarer sig godt på platformen, vil vi lave en port til telefon-markedet, da der her er et større marked for casual games.


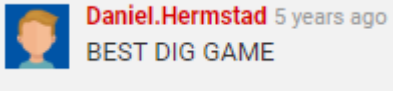

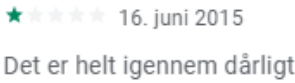


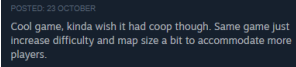
### Forretningsmodellen - Ulrik

Maersk Miner skal være en paid business model, hvor spillet vil komme til at koste 4.99 euro, 37 kr. og 5 USD. Denne pris er sat i den lavere ende af spektret, da spillet så passer perfekt ind i den kategori(1) hvor brugeren ikke skal gøre sig nogen større tanker om hvorvidt de har råd til spillet. Så passer det med at når streamers spiller så kan seerne også spille med.

### Our Competitors: - Bjarke & Ulrik

Vores konkurrenter består af casual minespil af tre forskellige generationer og tre forskellige prismodeller.

Competitors	Motherload	Miner	Dome Keeper
Capsule			
URL	<a href="https://www.y8.com/games/motherload">https://www.y8.com/games/motherload</a>	<a href="https://play.google.com/store/apps/details?id=nl.palatamedia">https://play.google.com/store/apps/details?id=nl.palatamedia</a>	<a href="https://store.steampowered.com/app/1637320/Dome_Keeper/">https://store.steampowered.com/app/1637320/Dome_Keeper/</a>

		<a href="#">a.minerfree&amp;hl=da&amp;gl=US&amp;pli=1</a>	
Short Description	Minespillet som de fleste husker fra “Y8 tiden”. En robot skal grave ned under jorden for at samle ressourcer. Ressourcerne kan bruges til at købe brændstof og opgraderinger, som kan få robotten dybere ned.	Nyere miner spil, sammenlignet med Motherload. Saml ressourcer under jorden og brug dem til at købe opgraderinger til din gravemaskine.	Spilleren skal beskytte en kuppel på overfladen mod fjender, mens han skal samle ressourcer under jorden, mellem angrebene.
Tags	Single player, Flash, Upgrade, Mining, Collecting, Free.	Hygge, Singleplayer, Stiliseret, Pixeleret, Offline.	Action, Indie, Roguelike, Alien survival, Sci-Fi.
Price	Free to play	Free to play med in app køb 0,99\$ - 5,99\$	17,99£ for hele spillet.
Reviews		2.8 stjerne ud af 5 totalt af 16.6 tusind anmeldelser 6.476 har givet 1 stjerne 5.454 har givet 5 stjerner	RECENT REVIEWS: <b>Very Positive</b> (996) ALL REVIEWS: <b>Very Positive</b> (3,192)
Release Date	29 Sep. 2004	3. Jul. 2013	27 Sep, 2022
What fans say	 	 	 
Est Wishlists (10x)	N/A	N/A	42.584

followers)			
Est. Gross revenue	N/A	N/A	\$1.1 mil

Grunden til at vi har est. Whishlists og est. Gross revenue skrevet N/A ved Motherload og Miner, er fordi det ikke er blevet opgivet nogen tal som vi kan tage udgangspunkt i. Motherload er et gratis spil som man kan spille på f.eks. Y8 eller Crazygames, hvor man ikke skal betale for at spille det. Miner er gratis at spille, men man kan købe ting i spillet, men der står ikke nogen steder hvor mange køb der er blevet lavet indgame. Selvom vi ikke kan finde ud af hvor mange penge Motherload og Miner har tjent kan vi nogenlunde se på hvor mange der har spillet det. Miner på Play store har 1+ million downloads, man har kunne finde spillet på Apple appstore engang men det virker til de har taget det ned. Det samme gælder med Amazon appstore der kan man heller ikke finde tal på noget.

Motherload har været lagt ud på alle de hjemme sider der har gratis web baseret spil. Kigger på hjemme siden af dem som har lavet Motherload så har den omkring 19 million total spiller. På Y8 er der 276.378 total spiller, så det betyder ud fra alt det vi har tal på så er Motherload bliver spillet 19.276.378 gange. Det vides ikke om det nummer er forskellige spiller eller bare være gangspillet er blevet spillet.

Så alt i alt har begge spil været meget populære med millioner af bruger hver.

Hvis vi skal se på konkurrenten Miner, så har den over 1.000.000+ downloads på google store, og samtidigt utroligt dårlige anmeldelser. Spillet fremstår ufærdigt, og alligevel vælger brugerne at prøve det.

Dette kunne stærkt tyde på at der er en tom plads på markedet inden for denne type spil. Hvis Miner er det bedste alternativ til Motherload, tyder det godt for Maersk Miner, da brugerne tydeligvis, baseret på Miner's downloads, er interesserede i spil-genren.

### **Pre release: 1.000 følgere**

Vi går efter at Maersk Miner minimum skal have 1.000 følgere, men flere er selvfølgelig foretrukket. Grunden til at vi har valgt 1.000 specifikt, bunder i at når man rammer de 1.000 følgere eller flere så er der en større chance for at komme på Steams “Popular Upcoming Games”<sup>3</sup>. Hvis vi kommer med på denne liste, vil spillet være på forsiden af Steam som gør at det vil blive vist til mange flere mennesker end det ellers ville. Følgerne kommer ofte i bølger, fremprovokeret af andre faktorer end Steam. Dette kunne fx være at en YouTuber eller streamer der tester vores spil, eller at vi nyligt har lavet et succesfuldt opslag vedrørende produktionen. For at opnå dette mål, skriver vi rundt til content creators angående fremvisning af spillet i håb om at skabe omtale. Her vil vi ikke henvende os til de største Twitch streamers, men derimod nogle mindre kanaler med omkring 100 seere. Der er også nogle få YouTube kanaler, som fremviser små indé games, de personligt mener har potentiale. Her kunne man, hvis økonomien indbyrdes i virksomheden i forvejen er stabil, betale forskellige content creators for at fremvise spillet. En helt anden mere proaktiv mulighed kunne også være at tage hen til forskellige spil-eksposé og fremvise spillet for potentielle investorer og købere.

### **Post release: dag 1-7 retention på 80%**

Det er vigtigt for alle spil at de beholder en stor del af den første bølge spillere. Hvis man mister for mange spillere inden for kort tid, kan det tolkes af andre potentielle kunder, som om spillet er dårligt og ikke er værd at købe. Derfor vil vi gerne arbejde med produktet på en måde, som giver brugeren lyst til at følge med i projektet og finder det sjovt at komme tilbage til udviklingen. I den første uge af spillets udgivelse vil vi gerne forsøge at beholde cirka 80%. Der ligger ingen tal på nettet baseret på hverken Motherload eller Miner, ud over hvor mange spillere de har i deres livstid. Kigger man derimod på Dome Keeper, faldt denne spillerbase fra 6.883 til 4.557 den<sup>4</sup> første uge. Dette giver en retention rate på 66.21. Vi har med vilje håbet at kunne ramme højere, da Maersk Miner ligger tæt op ad stilen i Motherload og Miner, som spillerne allerede er bekendt med. For at sikre os at vi beholder vores kunder, vil vi søge for at udgive spillet med så få fejl

---

<sup>3</sup> Zukalous(2022)

<sup>4</sup>SteamDB(2022) Dome Keeper

som umuligt. Hvis der dog alligevel skulle komme en fejl op, som vi havde overset, så vil vi sende hotfixes ud så hurtigt som muligt.

### **Post release: dag 7-14 retention på 60%**

Den anden uge af spillets udgivelse er generelt der, der er det største tab af spillerbasen. Her vil vi gerne beholde en retention på cirka 60%. Kigger vi på Dome Keeper, så er det ikke et umuligt tal at ramme. Deres spillerbase gik fra 4.557 til 3.210 spillere, hvilket giver en retention rate på 70.44%. Her skal de sidste fejl pensles ud, så man kan begynde at tænke på, om der skal tilføjes mere indhold i fremtiden. Dette kan både give flere spillere, men også gøre at man mister dem man har, hvis det enten tager for langtid med nyt materiale eller det man udgiver, ikke er interessant nok. Dette kræver også at spillerbasen er stor nok i første omgang, til at det kan betale sig at finansiere udvidelsen, så den kan skabe økonomisk overskud.

### **Post release: 1.000 Monthly Active Users(MAU)**

Det er vigtigt for os at finde ud af, hvor mange der stadig spiller spillet. Efter spillet er kommet ud af sin fase som et nyt spil, vil vi gerne have en stabil spillerbase på omkring 1.000 MAU. Hvis vi kan opnå dette, vil det sørge for at spillet får en længere levetid, og at vi har råd til at lave en port af spillet til telefonmarkedet. Siden Dome Keeper er blevet udgivet d. 27. september 22, ligger den nu med en spillerbase på alt fra omkring 700 til 1000 spillere om måneden. Jo længere tid et spil bliver vist af større streamers og YouTubers, jo større er chancen for, at der vil komme flere spillere til løbende. Så ved brug af reklame, vil vi månedligt forhåbentlig kunne få samme mængde nye spillere ind, som der kommer ud.

### **Motherload som brugeroplevelse - Bjarke**

Hvis vi skal gå ind i hvordan Maersk Miner skal bygges op, er vi nødt til at se på hvordan Motherload virker som produkt.

Spilleren bliver smidt på jordoverfladen og får at vide at man er en miner på mars. Man skal derefter grave mineraler for at tjene penge til brændstof, reparationer og bomber. Man kører med W, A, S,D, og graver derefter, hvis man kolliderer med væg eller gulv. Hele



Figure 1 Motherload Inventory

bevægelsessystemet er fast paced og maskinen tilter i luften, som hvis den var bundet af ægte fysik. Hele styresystemet er generelt godt og sjovt at bruge. Maskinen er dog, når propellen ikke trækker den opad, tung, og maskinen falder hurtigt til jorden. For store sammenstød skader maskinen, og skal hurtigst muligt repareres. Dette element giver en fornemmelse af maskinens skrøbelighed, og gør det mere spændende at bevæge sig rundt i undergrunden. Maskinen kan kun grave nedad og til siderne, hvilket gør at man skal lægge en strategi for hvordan tunnellerne skal graves. Dette giver også et element af spænding når man skal finde op til overfladen igen ved lavt brændstof.

Lydene ved fx 'lavt brændstof' er god og sætter gang i adrenalinen, selv inden niveauet er kritisk. Musikken er dog meget repetitiv.

Man trykker på 'I' for inventory, hvor man kan se sit equipment, sine upgrades og alle indsamlede ressourcer. Maskinen har en last begrænsning, som gør at man må smide noget, for at samle andet op, imellem salg.



Figure 2 Motherload Gameplay

Ting at tage med fra Motherload, er at der i Maersk Miner skal være en god balance mellem brændstof og dybde. Controls' skal være underholdende, og man skal have en god fornemmelse af maskinens holdbarhed. Disse "flyvske" bevægelser bliver dog nok svære at genskabe i Monogame. Inventory systemet virker godt, og spillet er generelt tilpas udfordrende. Hvis man skulle ændre på noget, skulle det være i lyd, og det generelt visuelle. Animationerne er næsten ikke eksisterende og tegningerne består ikke af ret meget. Spillet er dog lyst og farverigt, hvilket klart er værd at tage med. Her kunne man klart lave forbedringer som målgruppen ville sætte pris på.

## Content Analyse

### MDA -Saxe og Bjarke

#### **Mechanics:** - Saxe

vores kode laver chunks af terræn som er random i dets indhold, chunksne blive lavet anderledes med dets indhold jo dybere du kommer. Chunksne bliver gemt som txt filer bag spillet, da informationen skal gemmes for at mindke forbrug af ens ram. Dermed vil det maksimale af tegnet chunks være 4 adgangen. Mining af terrænet vil ændre informationen af den chunk og gemmes i txt file efter chunken bliver forladt.

#### **Dynamics:** - Bjarke

Vores gameplay er en balance mellem at grave så mange chunks af ressourcer som muligt, uden at løbe tør for energi. Denne energi får man ved at tanke op på ved opladeren på overfladen. Spilleren bruger piletasterne til at køre/grave til siderne, grave ned, eller flyve op, for at kunne bevæge sig gennem terrænet. I jorden findes materialerne som kan bruges til at bygge nye gadgets og udviklinger, som kan bruges til at nå endnu flere klaustrofobiske og omspændende dybder. Disse finder spilleren i en workshop på toppen, sammen med batteri-opladeren, som kan aktiveres ved kollision plus "E" tasten. Spillet slutter ved at robotten bliver så avanceret i sin udvikling, at den kan undslippe sin evige graveopgave. Dette vil ske når spilleren køber den sidste AI opgradering til robotten gennem workshoppen.

#### **Aesthetics:** - Bjarke

I vores spil har vi selv tegnet alle sprites for bedst selv at kunne sætte stemningen. Denne tager mest udgangspunkt i en lys og farverig palet. Spilleren bliver præsenteret for universet som underlagt er præget af vores dystopiske baggrundshistorie. Man får ikke meget forklaret, ud over den gennemgående dynamiske opgave 'Grav'. Da man er en robot designet til at grave, er dette

den eneste opgave som betyder noget, og det eneste information den behøver. Dette er dog ikke alt som spilleren oplever, hvilket her forhåbentlig skaber en stemning af mystik og flere spørgsmål end svar, jo dybere i jorden man kommer. Mystiske objekter og fortidslevn fra tidligere krige dukker frem af jorden, og kan inspiceres af robotten inden destruktion. Som robotten bliver mere og mere avanceret, vil der droppes hints, som vil efterlade spilleren med en følelse af om hvorvidt robotten er santient eller ej. Til sidst vil robotten stille spørgsmål ved sin rolle i universet, og til hvorfor og hvem der har givet den denne evige graveopgave i første omgang. Dette fører til spillets slutning hvor robotten undslipper sit formål.

## Objekt Orienteret Analyse

### Systemafgrænsning - Saxe

- Genre
  - Platformer, overlevelse, Miner.
- Core mechanics
  - Hardcore survival, hvis man løber tør for brændstof dør man, .
  - Flyve og rulle i sin robot for at få nødvendige ressourcer og bedre udstyr.
  - Man har kun et liv.
  - Grave, man graver sig tunneller for at finde sig frem til de ønskede materialer
  - Udforsk dybet til man finder det ønskede SDM.
- Kunde segment
  - Folk der kan lide at spille hardcore survival spil.
  - Folk som kan lide slow paced spil.
- Platform
  - Pc (senere port til mobile)



## Use cases - Saxe

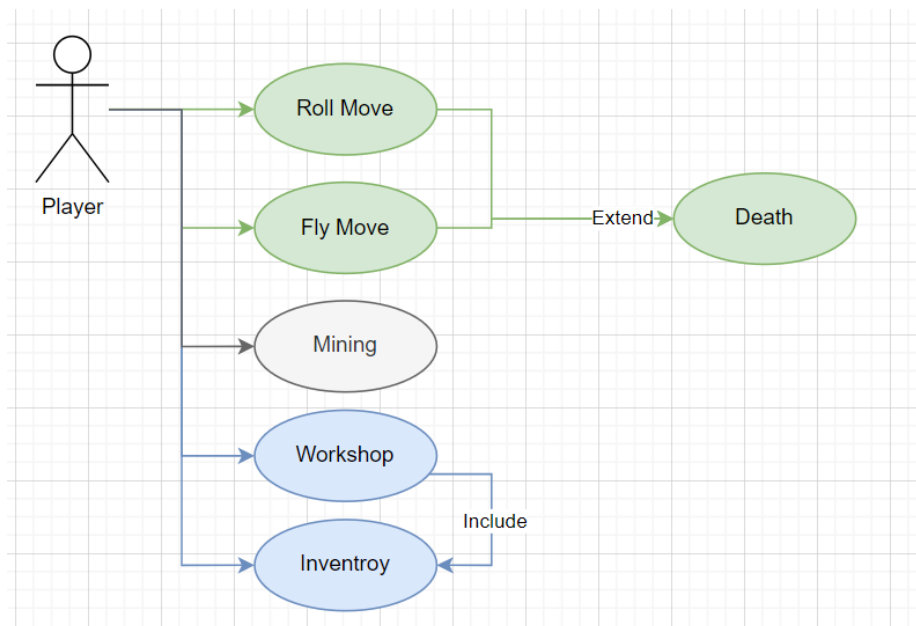


Figure 3 Use Case Diagram

Vi har 6 use cases, da vi har tænkt os frem til at dette er hvad spilleren skal kunne benytte sig med i spillet. Eftersom vores spil ingen enemies har og et relativt simple game loop, er vi endt med 6 der har direkte relation med spilleren.

- Roll Movement
- Flying movement
- Mining
- Selling
- Buying
- Inventory

Use Case Name	Roll Movement
Description	Kan få spilleren til at flytte sig på jorden
Actors	Spilleren
Preconditions	Spilleren er på jorden og har brændstof
Main Success scenario	Spilleren Flytter sig i ønskes retning.

<b>Alternatives scenarios</b>	Hvis spilleren løber tør for brændstof, dør man. Hvis spilleren stopper med at være på jorden
-------------------------------	--

<b>Use Case Name</b>	<b>Flying Movement</b>
<b>Description</b>	Spilleren kan få robotten til at flyve i ønsket retning (hurtigere end andet movement)
<b>Actors</b>	Spilleren
<b>Preconditions</b>	Mining action er ikke igang
<b>Main Success scenario</b>	Spilleren Flyver i ønsket retning, og mister brændstof
<b>Alternatives scenarios</b>	Hvis robotten løber tør for brændstof, så vil spillet ende.

<b>Use Case Name</b>	<b>Mining</b>
<b>Description</b>	Spilleren kan Fjerne terrænet, og skaffe ressourcerne fra det
<b>Actors</b>	Spilleren
<b>Preconditions</b>	Spilleren skal være foran ønsket terræn der skal fjernes
<b>Main Success scenario</b>	Spilleren Miner terræn og terrænet bliver fjernet.
<b>Alternatives scenarios</b>	

<b>Use Case Name</b>	<b>Selling</b>
<b>Description</b>	Spilleren kan sælge alt i inventory ved at flytte sig hen til shoppen
<b>Actors</b>	Spilleren
<b>Preconditions</b>	Spilleren skal være foran shoppen, og have noget i inventory

<b>Main Success scenario</b>	Spilleren mister alt i inventory og for penge til dets pris.
<b>Alternatives scenarios</b>	

<b>Use Case Name</b>	<b>Buying</b>
<b>Description</b>	Spilleren kan købe nye dele eller brændstof
<b>Actors</b>	Spilleren
<b>Preconditions</b>	Spilleren skal være foran shoppen
<b>Main Success scenario</b>	Spilleren kan åbne shoppen ved at klikke på den, og købe ønskede opgraderinger
<b>Alternatives scenarios</b>	

<b>Use Case Name</b>	<b>Inventory</b>
<b>Description</b>	Spilleren kan åbne inventory og se hvad der er i, og smide det ud.
<b>Actors</b>	Spilleren
<b>Preconditions</b>	
<b>Main Success scenario</b>	Spilleren åbner sin inventory og smider dårligere ressourcer ud.
<b>Alternatives scenarios</b>	

## Funktionsliste

Herunder har vi lavet en tabel over hvordan vi har tænkt os at spillet skal hænge samme. Tabellen er sat op på den måde så vi ved hvad vi skal have lavet først og hvad vi kan vente med at lave.

Funktionsnavn	Prioritet	Involveret klasser	Involveret actors	Involveret use case	Foreslag til metode navn
Death	Høj	Player	Player	Death	
Mining	Høj	Player	Player	Mining	
GroundCollision	Høj	Player	Player	Roll Movement	Collision
Change Terrain	Høj	Terrain			Change
Which Terrain	Høj	Terrain			Which
Starting Terrain	Høj	Terrain			Start_Terrain
Player Ground Movement	Høj	Player	Player	Roll Movement	PGMove
Player Flying Movement	Høj	Player	Player	Flying Movement	PSMove
Player Falling Movement	Lav	Player	Player		BCollision
InventoryManagement	høj	Inventory		Inventory	Manage
Inventory Updater	høj	Inventory		Inventory	Update
Starting Inventory	Lav	Inventory		Inventory	Start_Inv
Inventory Mouse Collision	mellem	Inventory		Inventory	Box_Collis
Workshop Update	Mellem	Workshop		Building	Update
Works. Mouse Collision	Mellem	Workshop		Building	Box_Collis
Workshop Manager	Høj	Workshop		Building	Manage
PlayerResourceLoss	Lav	Player	Player	Death	Fuel_Loss
Chunk Starter	Lav	Terrain			C_Start
Chunk Maker	Lav	Terrain			C_Maker
Chunk Reader	Lav	Terrain			C_Read
Chunk Writer	Lav	Terrain			C_Write
Inspect Artefact	lav	inventory		inventory	ShowArtefactDescription

## Klassediagram - Saxe

Det røde er vores klasser der omhandler Spilleren og dens mechanics, det lilla er vores klasser der opsætter en inventory og Workshop for spilleren til at opgradere ens robot. Den grønne terrain styre alt terræn. GameWorld er vores Hoved klasse der kontrollere koden, dermed er det den koden der henter kode fra resten.

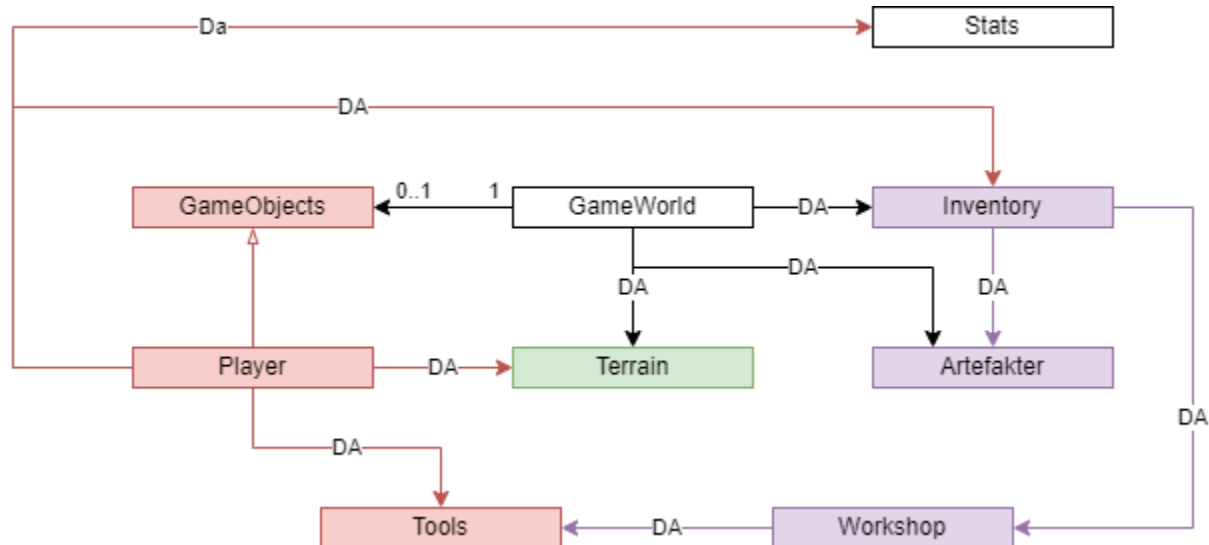


Figure 4 Klasse Diagram

## SystemDefinition - Bjarke

Maersk Miner er et spil udviklet med Monogame i sproget C# med fokus på at skulle kunne køre på pc. Senere, når det er naturligt gennemtestet og poleret, vil det blive porteret til mobil platform. I spillet er man en lille robot hvis eneste formål er at grave. Det er hvad den er skabt til og derfor alt spillet handler om. Spilleren skal grave ressourcer under jorden, uden at løbe tør for brændstof, for så at bygge nye personlige opgraderinger oppe på overfladen og genopfylde sine depoter.

Opgraderinger, som fx større batteri, bedre solpaneler, kraftigere bor eller udvikling af dens først simple kunstige intelligens, udvikler spillerens oplevelse gennem spillet. Omkring robotten vil spilleren opdage at alt ikke er som det burde være. Jorden er i ruiner og virksomheden, som

robotten graver for, eksisterer ikke længere. Under jorden findes artefakter, som fortæller historien om hvad der skete med verden, og som robotten bliver klogere for hver AI opgradering, begynder den med tiden at stille spørgsmål ved sin eksistens. Til sidst slutter spillet ved at robotten bliver klog nok til at handle ud over sin programmering, og bryde det uendelige grave-behov.

Spillet er udviklet med udgangspunkt i spillet Motherload fra 2004, som de fleste fra Y8-generationen kender. Maersk Miner er derfor en hyldest til Motherload og til nostalgien som det tilhører. Denne generation af gamere vil derfor også være vores målgruppe. Udover elementerne fra Motherload, har vi tilføjet elementerne, som fx artefakter, historie og opgraderinger, som giver et forhåbentligt positivt twist til spillerens forventninger.

Spillet har kun et enkelt centralt objekt, nemlig grave-robotten. Resten af klasserne indgår primært i terrain eller workshop. Af denne grund har vi vurderet at projektets scope er realistisk at udvikle inde den givne tid, da vi er tre medlemmer i gruppen som hver især vil tage del i kodningen. Hvis noget bliver der nok tilføjet ekstra detaljebaserede features, for blot at give alle på holdet noget mere at arbejde på. Der skal afsættes tid til tegning af sprites, lyd og skrivning, men koden vil blive sat forrest i tidsbudgettet. Funktioner som at kunne bevæge sig flydende rundt i den tilfældigt genererede undergrund er absolut førsteprioritet. Styresystemet skal føles rart og intuitivt. Her forventer vi at spilleren skal bruge piletasterne eller WASD. Derudover skal vi skabe en nysgerrighed hos spilleren, som skal friste ham/hun til udforske. Her spiller diversitet i terræn og specielle encounters en stor rolle.

## Første UML Diagram

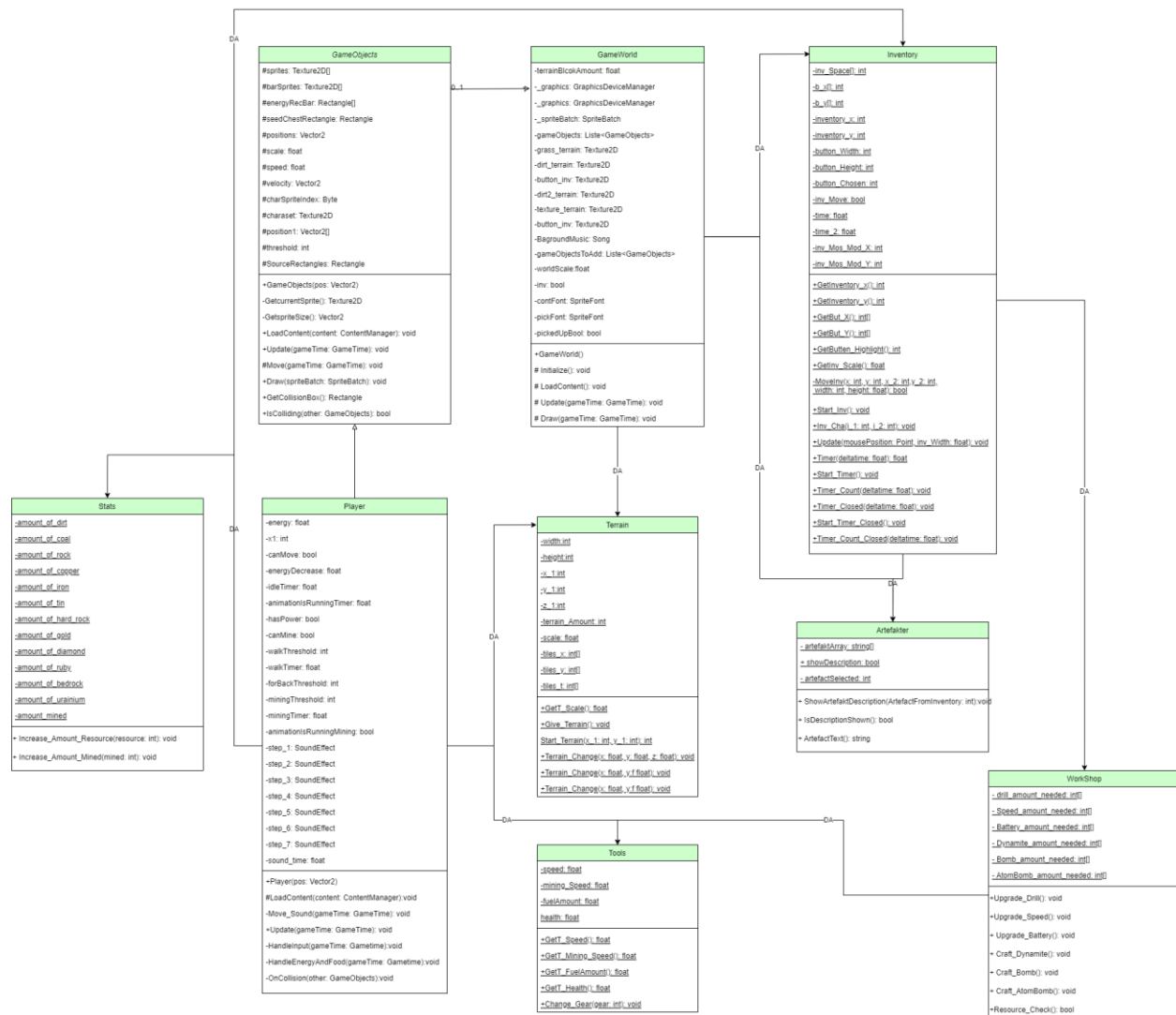


Figure 5 Første Udkast UML Klasse Diagram

```

classDiagram
    class GameWorld {
        -graphics: GraphicsDeviceManager
        -spriteBatch: SpriteBatch
        -screenSize: Vector2
        -groundSprite: Texture2D
        -textureTerrain: Texture2D
        -contFont: SpriteFont
        -contFont: SpriteFont
        -worldScale: float
        -inv: bool
        -offset_x: int
        -offset_y: int
        -current_chunk: int
        -toolsList: Liste<Tools>
        -gameObjects: Liste<GameObjects>
        -workshop: Liste<Workshop>
        +GameWorld()
        # Initialize(): void
        # LoadContent(): void
        # Update(gameTime: GameTime): void
        # Draw(gameTime: GameTime): void
    }
    class GameObjects {
        #frame: int
        #frameTimer: float
        #idleTimer: float
        #effect: SpriteEffects
        #speed: float
        +position: Vector2
        #SPRITESHEET_DRIVING: string
        #SPRITESHEET_DRILLING_SIDE: string
        #SPRITESHEET_FLYING: string
        #SPRITESHEET_DIGGING_DOWN: string
        #SPRITE_OVERLAY: string
        #_spriteSheetTexture: Texture2D
        #_spriteIdleTexture: Texture2D
        +_controlsFont: SpriteFont
        +_drillOverlay: Player
        +_drilling: bool
        +LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        +Draw(gameTime: GameTime): void
    }
    class Workshop {
        #spritePlacer: Texture2D
        #upgradeInfo: Texture2D
        #spritePlacerPos: Vector2
        #fontTitle: SpriteFont
        #uiRectangles: Rectangle[]
        #isCount: Texture2D
        #artifactSprite: Texture2D
        #artifactPlacer: Rectangle[]
        #isCraftClicked: bool
        #isUpgradeClicked: bool
        #isArtClicked: bool
        #isStatsClicked: bool
        #isUpgraded: bool
        #artifactFound: bool
        #upgradeClicked: byte
        #artifactClicked: byte
        -r1Copper: int
        -r2MilitarScrap: int
        -r3Titanium: int
        -r4Plat: int
        -r5Uranium: int
        -isInvOpen: bool
        #closeDownShopTime: float
        #menuSound: SoundEffect
        +Upgraded: bool
        +ArtFound: bool
        +isInvOpen: bool
        +R1Copp: int
        +R2Milit: int
        +R3Titl: int
        +R4Plat: int
        +R5Uran: int
        +LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        +Draw(spriteBatch: SpriteBatch): void
    }
    class UpgradeButton {
        -mouse: MouseState
        +LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        +DrawUpgrade(spriteBatch: SpriteBatch): void
        +UpgradeTab(gameTime: GameTime): void
    }
    class ArtifactsButton {
        -mouse: MouseState
        -artiRec: Rectangle[]
        +LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        +DrawArtifact(spriteBatch: SpriteBatch): void
        +ArtifactTab(gameTime: GameTime): void
    }
    class CraftButton {
        -mouse: MouseState
        +LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        +DrawCrafting(spriteBatch: SpriteBatch): void
        +CraftingTab(gameTime: GameTime): void
    }
    class StatsButton {
        -mouse: MouseState
        +LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        +DrawStats(spriteBatch: SpriteBatch): void
        +StatsTab(gameTime: GameTime): void
    }
    class Player {
        -spriteIdleTexture: Texture2D
        -drivingTexture: Texture2D
        -flyingTexture: Texture2D
        -drillingSideTexture: Texture2D
        -drillingDownTexture: Texture2D
        +Player(pos: Vector2)
        #LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
    }
    class Tools {
        #SPRITESHEET_BATTERY: string
        -batterySpritesheet: Texture2D
        +battery: float
        +drainTimer: float
        +gainTimer: float
        -batteryFrame: int
        -batteryMax: int
        +solarPanelSize: int
        +solarPanelCombined: int
        #LoadContent(content: ContentManager): void
        +Update(gameTime: GameTime): void
        # Draw(gameTime: GameTime): void
    }
    class Terrain {
        -width: int
        -height: int
        -x: 1: int
        -y: 1: int
        -x: 1: int
        -y: 1: int
        -tiles_x: int[]
        -tiles_y: int[]
        -tiles_1_c1: int[]
        -tiles_1_c2: int[]
        -tiles_1_c3: int[]
        -tiles_1_c4: int[]
        -tiles_empty: int[]
        -tiles_mined: float[]
        -loaded_chunks: list<int>
        -switch_off: byte
        -stonebreak_1: SoundEffect
        -stonebreak_2: SoundEffect
        -stonebreak_3: SoundEffect
        -stonebreak_4: SoundEffect
        -sound_timer: float
        +Give_Terrain(): void
        +Player_Collis(side: int, delatime: float): bool
        +Player_Collis_Gravity(): bool
        +Is_We_On_Top(): bool
        -Break_Sound(deltatime: float): void
        +Load_Chunks(x: int, y: int): void
        -Get_Tiles(pos: int) tile: int[]
        -Sorter_New(new_one: int): void
        -Sort_Chunk_Moving(new_one: int, x: 1: int, y: 1: int): void
        -Start_Chunk(direction: int): void
        -Chunk_Maker(direction: int): int[]
        -Chunk_Terrain(x: int, y: int): int[]
        -RandomThreshold: int[]
        -Chunk_Write(r: int, direction: int): void
        -Chunk_Check_File(position: int): bool
        -Chunk_Read(direction: int): int[]
        -Chunk_Differ(): int
        +Loaded_Chunk_Differ(): int[]
        +DirectionLoaded_one: int[]
        +Terrain_Change(x: float, y: float): void
        +Change(x: float, y: float, z: float, chunk: int): void
        +Which(x: float, y: float, chunk: int): int
        +Mining_Updater(x: float, y: float, delatime: float, direction: int): void
        -Mining_On_Tiles(i: int, delatime: float): void
    }
    GameWorld "1" -- "0..1" GameObjects
    GameWorld "1" -- "0..1" Workshop
    GameWorld "1" -- "0..1" Player
    GameWorld "1" -- "0..1" Tools
    Workshop --> UpgradeButton
    Workshop --> ArtifactsButton
    Workshop --> CraftButton
    Workshop --> StatsButton
    Player --> Tools
    Tools --> Terrain
  
```

Figure 6 Endelig UML Klasse Diagram

24



For at se se billedet i større format kan man åbne Gruppe 1 UML Klasse Diagram i bilagmappen. Vi endte med ikke at lave stats og artifact klasserne. Dog har vi tilføjet flere klasser, såsom dem for button. Vi har tilføjet mange funktioner og variabler i fields til de klasser vi har endt med at tilføje, da meget af det kode vi har lavet er kode som vi ikke har prøvet at bruge og skrive før. Derfor er det naturligt, at der har været flere ændringer end forventet, især når vi finder andre måder at lave produktet på end de først planlagte metoder.

Under Terrain var det ikke først tænkt at der skulle være lyd, collision og tyngdekraft under den klasse. Da vi først tænkte at det skulle være under player klassen. Alt med chunks er kodet vi havde meget lidt viden om hvordan skulle implementeres, så den del har fået mange flere funktioner og field variabler. Men field variabler for chunks var næsten perfekt.

Da vi lavet det første udkast til UML med WorkShop havde vi ikke prøvet at lave et inventory system før som skulle kunne indeholde så mange forskellige ting. Der var ikke nok forskellige "Texture2D" eller "Rectangle" som vi kunne gøre brug af. I første udkast tænkte vi det hele kunne være i en klasse, men efter vi begyndte at arbejde på inventory fandt vi ud af at der kommer til at stå alt for mange ting inde i den WorkShop klassen og det ville være svært at finde rundt, hvis det hele kun stod er et sted. Derefter besluttede vi os for at have en klasse til hver knap for at gøre processen meget nemmere.

Player klassen endte primært med at indeholde de forskellige visuelle animationer til playeren. De bagvedliggende reaktioner, som vi havde regnet med skulle ligge her, er udviklet i GameWorld. Dette kom af at det i spillet er terrænets offset som flytter sig ved input, ikke spilleren. Dette havde vi ikke taget højde for i det gamle diagram. Derudover arver den fra den abstrakte klasse GameObjects, som tegner spilleren fra Player.

Tools er en klasse vi senere udviklede, for at have et sted til at kunne sammensætte de forskellige upgrades. Der er også hvor Batteriet er udviklet og tegnet, da det er kraftigt bundet sammen af opgraderingerne.

## Objekt Orienteret Programmering

### Implementering

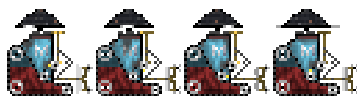
Der var nogle ting som vi ikke noget at få implanteret ind i spillet grundt af tidsnød. Her har vi ikke nået det vi ville og så var nød til at skrotte idéen eller skifte den ude med en anden idé som var nemmere at udføre.

- At kunne crafte forskellige ting til at hjælpe dig med at grave hurtigere. Her havde vi tænkt os at man skulle kunne bygge bomber der kunne hjælpe spilleren med at grave.
- At kunne se forskellige spil-statistikker inde i sin inventar. Her skulle man kunne se f.eks. hvor lang tid man har spillet, et totalt antal på alle blocks man har mined.
- Et ordenligt UI som passer sammen ind i spillet, her har vi ikke fået noget at lave et UI som for det meste ikke er andet end nogle simple firkanter med 1 farve.

Igennem dette projekt er der ikke blevet brugt nogle modler udenfra alt er blevet lavet med egen hånd. Nogle få sprites er taget fra et tidligere projekt og er blevet genbrugt til nogle terræn blocks, som vi i samme gruppe havde lavet tidligere.

### Animation - Bjarke

Spiller animationerne, som bliver styret af Player' klassen, er bygget af 4 spritesheets med fire frames på hver. Alle på 32 pixels. Den bliver kørt ved at en timer tæller op til at lægge 32 pixels til X koordinatet, og således hoppe et frame længere ned af spritesheetet.



*Figure 7 Flying*

Dette sker hver gang der er gået 50 millisekunder. Når den når ud over sidste frame, dvs 128 pixels, starter den forfra.

Hvornår de forskellige animationer afspilles er styret af spiller input som fx W,S,A,D, men bagvedliggende endnu mere af collision-bool's fra GameWorld. Hvis spilleren fx er idle i luften aktiveres propellerne, mens der på jorden vises køre-animationen. Vælger man at køre til højre med D og rammer en væg, skifter den fra at køre, til at bore etc.

```
#region - FLYING ANIMATIONS -
if (Keyboard.GetState().IsKeyDown(Keys.W))
{
    drilling = true;
    if (GameWorld.upCollision == true && GameWorld.inAir == true)
    {
        _spriteSheetTexture = _drillingUpTexture;
    }
    else
    {
        _spriteSheetTexture = _flyingTexture;
    }
    frameTimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
    if (frameTimer > 50)
    {
        frame = frame + 32;
        frameTimer = 0;
    }
    if (frame == 128)
    {
        frame = 0;
    }
}
```

Figure 8 Loading Spritesheet for flying

Hjul-positionen er ens på alle aktionerne og bliver styret af samme frame variabel, hvilket gør at der vil være en naturlig overgang mellem animationerne. Hjul-positionen bliver altid bragt videre til den næste. Selv når man fx flyver nedad, hvor animationen bliver afspillet baglæns, eller hastigheden af animationen ændres ved idle fald, er animationernes overgange sammenhængende. Dette er gjort for at skabe en fornemmelse af at maskinen bruger hjulene og de mekaniske dele til at trække sig fremad.

Derudover har den et idle frame, (figur 9), som bliver tegnet ca. hvert sekund. Denne bliver tegnet ovenpå køre-animationen, 1 pixel højere oppe, om så den kører eller ej. Dette får hovedet til at hoppe med 1 frame, separat fra hjul rotationen, og gør derfor at man føler at aktionerne er naturlige, da de er uafhængige.



Figure 9 Idle Frame

Oppe i hjørnet af skærmen findes et batteri, som er styret af Tools' klassen.

Dette kører på samme måde som spiller-animationen, men denne animation spilles både for- og baglæns, og er ikke styret af direkte spiller input. Derimod, hvis man fx er i bunden og graver, så går animationen frem og batteriet drænes.



Figure 10 Batteri

```
// - BATTERIETS VIRKNING OG ANIMATION -
draintimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
//drainTimer tæller op til batteryMax, som bliver defineret pr batteri upgrade.
if (Terrain.is_we_on_top() == false && draintimer > batteryMax && batteryFrame <= 300)
{
    batteryFrame = batteryFrame + 30;
    draintimer = 0;
}
```

Figure 11 Batteri Dræning kode

Hvis man er på overfladen, spilles den baglæns og batteriet fyldes op.

```
// - SOLPANELETS VIRKNING -
gaintimer += (float)gameTime.ElapsedGameTime.TotalMilliseconds;
/*gainTimer tæller op til solarPanelCombined,
som bliver defineret pr ovenstående upgrades på sloarSize og batteryMax.
Når den er nået til max, skifter den frame og timeren bliver resettet. */
if (Terrain.is_we_on_top() == true && batteryFrame > 0 && gaintimer > solarPanelCombined)
{
    batteryFrame -= 30;
    gaintimer = 0;
}
```

Figure 12 Batteri Oplade Kode

Hvis spriten når til sit sidste frame er spillet tabt. Tiden som bruges til at tælle frem og tilbage kan manipuleres gennem batteri- og Solpanel upgrade listerne fra workshop klassen. Max-batteri upgrades manipulerer hvor meget tælleren skal tælle op til, før den skifter frame. Denne stiger pr.

upgrade, så batteriet holder længere. Solpanelets ydeevne består af

```
solarPanelCombined = batteryMax - solarPanelSize;
```

hvilket gør at der er en balance i at det tager længere tid at fylde et større batteri op. Dog stiger solpanelets størrelse også med opgraderingerne, og mere kan derfor trækkes fra Max-batteri-tælleren når batteriet fyldes op.

```
#region - Battery upgrade -  
if (UpgradeButton.Upgrade[7] == true)  
{  
    batteryMax = 14000;  
}  
if (UpgradeButton.Upgrade[6] == true)  
{  
    batteryMax = 13000;  
}  
if (UpgradeButton.Upgrade[5] == true)  
{  
    batteryMax = 12000;  
}  
if (UpgradeButton.Upgrade[4] == true)  
{  
    batteryMax = 11000;  
}
```

Figure 13 Batteri Opgrades

### Inventory - Ulrik

For at give spilleren noget at grave efter, så er det vigtig at have et inventory system, hvor man har mulighed for at gøre brug af alle de ting man har samlet op på vejen. Her har vi udarbejdet et inventory system som har 4 kategorier i alt. Dette system skal bruges til at hjælpe spilleren, med at gøre nogle ting nemmere. Den første knap hedder Craft, som skal bruges til at spilleren kan bygge forskellige ting der kan hjælpe spilleren med at grave. Den anden knap hedder Upgrade som spilleren gør brug af til at opgradere sig selv, her kan man forbedre alt fra at hakke jord, batteri liv og gøre genopladning af ens batteri lidt hurtigere. Den tredje knap hedder Artifacts som bruges til at formidle historien om verdenen til spilleren igennem artefakter man samler op under jorden. Den fjerde knap hedder Stats som bruges til at indeholde mange generelle informationer om hvad spilleren har lavet i spillet.

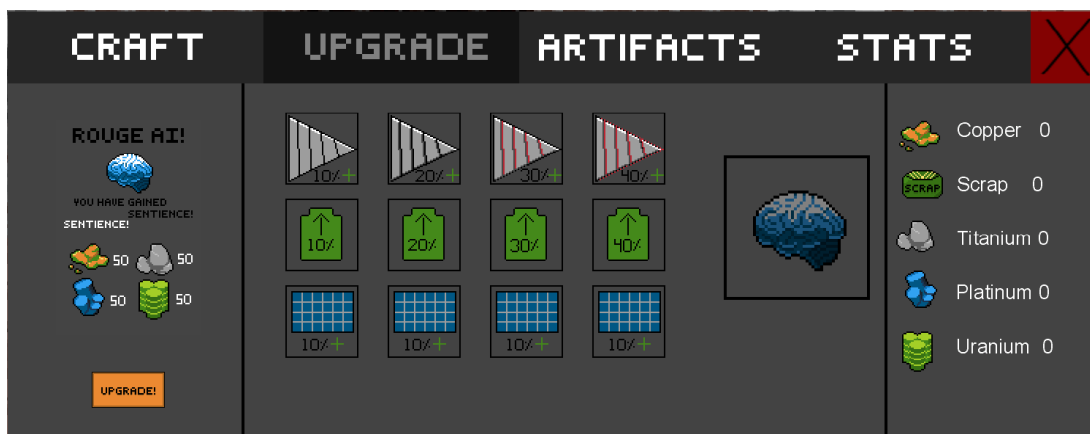


Figure 14 Inventory Upgrade knap

Hele inventory system bliver styret af en super klasse som hedder WorkShop som så nedarver til 4 andre klasser, en klasse til hver kategori. Workshopklassen er lavet til en abstractklasse for at gøre det nemmere at arbejde i med de nedarvet klasser. I stedet for at hver sub klasser har en masse variabler selv, så er det gjort nemmere ved at gemme alle variabler i WorkShop for at bedre kunne tegne hele inventory.

Med nogle af variablerne er det nødvendig at søge for at man kan tage fat i dem i andre scripts når de skal bruges. Derfor er der lavet otte property's som man kan interagere

med i andre klasser. Fem af dem er til at styre ens ressourcer. Den sjette property er et opgradering systemet, som er lagt ind i et array med opgradering så man kan se hvad for nogle opgraderinger man har købt. Den syvende property er til at se om ens inventory er åben. På den måde at koden er sat op nu, så ville der ikke være nogen grund til at have den som en static property. På grund af at man styrer åbning og lukning af inventory igennem WorkShop og de nedarvet klasserne. Hvis man fjernede den som property og bare lavet den om til protected variable så ville det have været nok. Men vi beholder den stadig i det tilfælde af at vi et tidspunkt syntes inventory skal styres af spilleren i stedet. Den sidste property er en boolsk property som man kan gøre brug af inde i andre klasser for at finde ud af om man har fundet nogle artefakter endnu.

```
protected static Texture2D[] spritePlacer = new Texture2D[30]; //A
protected static Texture2D[] upgradeInfo = new Texture2D[13]; //An
protected static Vector2[] spritePlacerPos = new Vector2[10]; //A
//U
protected static SpriteFont[] fontsTitle = new SpriteFont[10]; //A
protected static Rectangle[] uiRectangles = new Rectangle[50]; //A
protected static Texture2D[] reCount = new Texture2D[5]; //For sto
protected static Texture2D[] artifactsSprite = new Texture2D[20];
protected static Rectangle[] artifactsPlacer = new Rectangle[20];
//4 bools to see which tap your are on
protected static bool isCraftClicked = false;
protected static bool isUpgradesClicked = false;
protected static bool isStatsClicked = false;
protected static bool isArtiClicked = false;

protected static bool[] upgraded = new bool[13]; //for the upgrade
protected static bool[] artiFound = new bool[12]; //for the artifa
```

Figure 15 WorkShop Variabler

```
public static bool[] Upgraded { get { return upgraded; } set { upgraded = value; } }  
//A property so you can access the whcich artifacts you have found from another class  
0 references  
public static bool[] ArtiFound { get { return artiFound; } set { artiFound = value; } }  
//A property so you can open the inventory through another sub class  
16 references  
public bool IsInvOpen { get { return isInvOpen; } set { isInvOpen = value; } }  
//5 properties for each resource that can be accessed in anyother script  
12 references  
public static int R1Cop { get { return r1Copper; } set { r1Copper = value; } }  
9 references  
public static int R2Mili { get { return r2MilitaryScrap; } set { r2MilitaryScrap = value; } }  
20 references  
public static int R3Tit { get { return r3Titanium; } set { r3Titanium = value; } }  
12 references  
public static int R4Plat { get { return r4Plat; } set { r4Plat = value; } }  
12 references  
public static int R5Uran { get { return r5Uranium; } set { r5Uranium = value; } }
```

Figure 16 Workshop Properties

Når inventory er åben og man kigger i Draw så bliver alle knapperne tegnet hvorefter hvis man klikker på en af de fire knapper så bliver der kaldt på de forskellige nedarvet kasser. Hver klasse indeholder deres egen metoder hvor der står alt det den skal tegne. Dette bliver gjort på denne måde for at sørge for at Workshop klassen ikke blive overfyldt, som så vil gøre det svære at finde rundt i koden når man skal skrive eller læse den.

```
public void Draw(SpriteBatch spriteBatch)  
{  
    if (isInvOpen == true) //handles the drawing and the 4  
    {  
        //draws the button at the top of the inv  
        Buttons  
        //Switches between what is shown when a button is  
        if (isUpgradesClicked == true)  
        {  
            UpgradeButton.DrawUpgrade(spriteBatch);  
        }  
        if (isArtiClicked == true)  
        {  
            ArtifactsButton.DrawArtifact(spriteBatch);  
        }  
        if (isStatsClicked == true)  
        {  
            StatsButton.DrawStats(spriteBatch);  
        }  
        if (isCraftClicked == true)  
        {  
            CraftButton.DrawCrafting(spriteBatch);  
        }  
    }  
}
```

Figure 17 Workshop Draw

Inde i klassen der hedder UpgradeButton, så er det her at man har mulighed for at opgradere sig selv. Det er også inde i denne klasse hvor man kan se hvor mange ressourcer man har samlet op. Opgraderingerne er sat op på den måde at hvis man klikker på en opgradering så ville man i venstre side få en boks, som beskriver den opgradering til en, der står også hvad den koster. Opgraderingerne er sat op på den måde at den står i en rækkefølge, så man kan ikke opgradere midt i det hele uden at få den før. Hvis man vil

f.eks. opgradere ens drill og man vælge den sidste, så kan man ikke købe den, men man skal tage den fra den første opgradering og så forsæt. Hvis man ville havde den sidste opgradering så er man tvunget til at få alle de forrige opgraderinger.

Kollision mellem musen og opgraderingerne sker på denne måde.

```
if (uiRectangles[5].Contains(mouse.X, mouse.Y) && mouse.LeftButton == ButtonState.Pressed)
{
    upgradeClicked = 1;
}
```

Figure 18 Kollision check mellem mus og en boks

Ved at gøre brug af en indbygget funktion som ligger i MonoGame "MouseState" så kan vi lave et check der ser om musen er indenfor området af en boks, hvorefter vi kan spørge om den bliver trykket. Denne form for kollision der sker mellem musen og de forskellige bokse, bliver alle sammen gjort på samme måde.

Når mand klikker ind på en opgradering så for man en boks med noget information og en ny knap mand kan trykke på, koden bag den knap er så leds.

```
if (uiRectangles[37].Contains(mouse.X, mouse.Y) && mouse.LeftButton == ButtonState.Pressed)
{
    switch (upgradeClicked)
    {
        //First line
        case 1://Upgrade DrillBit 10% faster
            if (R2Mili >= 10 && R4Plat >= 5 && Upgraded[0] == false)
            {
                Upgraded[0] = true;
                R4Plat -= 5;
                R2Mili -= 10;
                menuSound.Play();
            }
            break;
    }
}
```

Figure 19 Upgrade info knap

Det er som sagt før, samme form for kollision der sker mellem musen og de forskellige bokse. Her er forskellen bare at der ikke ville ske noget medmindre man opfylder nogle specifikke krav. De krav i det her tilfælde handler kun om man har de nødvendige ressourcer og har den tidligere opgradering hvis der er en. Hvis man opfylder de krav, så kan man få lov til at købe opgraderingen. Når man køber en opgradering, så mister man de ressourcer den koster og der bliver afspillet en lyd. I tilfældet med opgraderingerne så giver de ikke en talværdi tilbage som man kan bruge, den returnere kun en sand eller falsk værdi som man så kan bruge et andet sted.



Placering af alle elementerne fra inventory er blevet fastkodet ind til det sted de har fra starten af. Grunden til dette er blevet gjort sådan er fordi det ikke er spilleren der bevæger sig, men verden rundt om spilleren som flytter sig. Så der er ikke nogen grund til at flytte rundt på User Interface (UI) med spilleren hvis den bare står stille.



*Figure 20 inventory artefakt knap*

Forsætter man og går ind i Artifacts knappen, så ville man kunne 12 forskellige billeder. Her har man mulighed for at klikke rundt på hvert enkelt et af billederne. Hvis man ikke har været under jorden og finde nogle artefakter endnu, så ville man ikke kunne se hvad for nogle artefakter der er eller læse noget omkring dem. Finder man et artefakt så ville man kunne klikke ind på den og læse information om den. Alt i koden er sat op ordentligt, det eneste der mangler, er at vi får ændret de nuværende billeder, til billeder af nogle artefakter som kan bruges. Når man klikker ind på et billede, fungerer kollisionen mellem de to på samme måde som det gjorde i inde i Upgrade knappen. Systemet med artefakter er blevet lavet på den måde at den laver et check der ser om man har fundet nogle artefakter. hvis man ikke har, så bliver den sat til falsk hvor billedet bliver gråt, hvis man har fundet et artefakt, så bliver den sat til sandt og vil man kunne klikke ind på det artefakt man har fundet

## **Lyd - Saxe**

Maersk Miner har lyd til de fleste ting man laver. Der bliver afspillet et par forskellige lyde alt efter hvad man laver. Der bliver afspillet en baggrundslyd når man bevæger sig, som startes af

bevægelse, denne lyd er lyden af en motor som burde hjælpe på ideen om at man er en maskine. Lyden bliver laveret i volume når man står stille, til at simulere en maskine i dvale.

Lyden af kollisionen mellem spiller og terræn er flere forskellige lyde der bliver afspillet tilfældigt. Hvilket bare gøres med en timer ved brug af deltatime, og når den timer rammer fuld så bruger vi random med en switch ud fra hvilket nummer der kommer (bilag 13). Vores lyd for mining af terræn er tilfældig for at vedholde variation i lydende. Hvis vi havde mange andre lyde, ville dette ikke være nødvendigt, eksempelvis forskellige for hver sten. Men nuværende er det nødvendigt at der bliver variation for ikke at irritere brugeren af programmet.

## **World Terrain - Saxe**

### **Hvordan den er skrevet op**

Ideen med terræn klassen er at vi har et koordinatsystem hvor hver position er en chunk, hver chunk har et array på 576 indexes der hver bestemmer hvilket terrain er hvor.

Spilleren er på en af disse chunks, og hver chunk fylder mere end skærmstørrelse.

Grunden til 576 er at vi bruger en skærmstørrelse af 1920 x 1080 som standard, vores sprites er 32 x 32 pixels store, 32 går ikke op med 1920 og 1080. Men vi ønsker at tegne hver sprite indtil det fylder skærmen ud, men hvis vores sprite ikke passer til skærmstørrelsen vil der blive tegnet dele af sprites i kanten af skærmen, og den går ikke. Så vi finder en anden resolution der går op i 32, men som samtidig går op i 16:9 (resolution for 1920x1080), for så kan vi gange væk forskellen. Vi fandt 1024 x 576 der går op i begge, og ved brug af denne resolution kan vi dividere den med 32, og få mængden vi skal bruge (32 på x, og 18 på y), gange dem op med forskellen på 1920x1080 og 1024 x 576 (1,875), dermed får vi et perfekt billede over hele skærmen ned til hver pixel. derefter valgte vi at spillet skulle forstørres til at ganges op med 5 (5 kan ikke bruges til at lave en perfekt kant (selvom 1920 kan divideres med 5 og derefter med 32 som så laver et helt tal, men det kan 1080 ikke)).

Men  $32 * 18$  er 576, hvilket er grunden til den arraylængde, da det er den mængde sprites til at fylde skærmen ud (indtil det blev forstørret). Indexet på arrayet bestemmer lokationen, hvilken værdi på indexet bestemmer hvilket terrain der er på lokationen.

opsætningen af spillet kunne være lavet uden y lokation (y lokation for terræn, dog kunne det også gøres andre steder), eftersom det ikke er nødvendigt på den måde vi laver terrænet (da det kun køre på index), men selvom det er lidt langsommere, laver vi det med y for bedre overskuelighed over koden.

Der bliver lavet nye chunks (og dermed nye arrays) hver gang spilleren opdager nyt terræn, disse chunks bliver så nedskrevet som txt dokumenter så informationen kan hentes næste gang spilleren ser det terræn. Dermed vil vi ikke lave masser af arrays der fylder på ens hukommelse, og vi vil nemmere kunne gemme spillet hvis vi lystede.

### **Hvordan virker chunksne**

Første gang chunks bliver brugt er når de bliver kaldt i starten af spillet for at lave den første chunk man står på.

Ellers vil chunks køre i denne rækkefølge.

Deciding:

vælger hvilken retning der bliver lavet nye chunks eller skiftet på main chunk (main chunk er den man står på (se bilag 1)).

Det gør koden ved at tage x og y positionen af verden (vi flytter verden rundt om spilleren så spilleren er altid i midten af skærmen), og checke om den position er større end givet minimum i hver retning, hvis det er, vil den retning blive givet en værdi.

vi har lavet mange forskellige versioner af denne funktion, men vi endte med at bruge 2 switches en for x og en for y, det viste sig at det mest simple var bedst (vi lavet en switch med 1 switch i hver case, som så havet en if i hver case, det blev meget uoverskueligt).

Men efter funktionen har givet en retning og checket at retningen ikke allerede er i brug, så sender den den retning videre til sortering, da der skal sortere listen af chunks der er i brug.

loaded\_chunks er en liste af int arrays der bruges til at se mængden af chunks der er i brug, dog husk at loaded\_chunks ikke har int arrays af 576, men af 2, da det er positionen den bliver brugt her.

vi sender den retning videre (grunden til at vi kalder det en retning og ikke en position er at værdierne bliver brugt til at skaffe positionen af denne retning plus position af main chunk)

### Sortering

Funktionen for sortering (se bilag 2), vi har skrevet mange forskellige måder op for at sortere listen af chunks, men vi kunne ikke få mine sorteringer til at virke, så vi endte med den simple løsning, hvilket er en switch for hver retning.

Vi skal bruge sortering for at kunne vide hvilket index hvert chunk har til senere.

Vi sorterer ved at finde den case der passer til retningen.

Derefter vil den checke om denne retning er i brug i øjeblikket, så køre den et loop der fjerner alle arrays i loaded\_chunks undtagen index 0, for den er main chunk, derefter giver den chunks til givet retning, hvis retningen er vandret eller lodret er der en der bliver tilføjet, og hvis den er skrå, er det 3. dette vil sørge for at man altid kan se en chunk, da der altid bliver lavet chunks i den retning man går.

### Tilføjelse af chunks:

For at få nye chunks bliver get\_tiles kaldt (se bilag 3), denne funktion checker for om der er en allerede eksisterende fil til den givet location (array pos) hvis der er, læser den filen, og hvis ikke vil den lave en ny fil. Begge funktioner giver et array tilbage som bliver brugt her til sidst.

for terrain er der 8 arrays der er konstant, er aktive i memory (som alle fylder 576 i længde), og blive lavet i fields.

Tiles\_t\_c med nummer er de arrays der holder på de terrain tiles der blive brugt i øjeblikket.

Grunden til 4 af dem er fordi maks mængde af chunks du ville kunne se på en gang er 4, disse arrays er givet deres value i denne funktion med den værdi der bliver returned af read file eller file maker. Når funktionen bliver kaldt, er der også en værdi i headeren kaldet tile, der bestemmer hvilket array der skal bruges.

tiles\_x og tiles\_y er positionerne af hver index, brugt når gameWorld positioner skal konverteres til terrain lokationer.

tiles\_empty bruges til at gøre tiles\_t\_c null når de ikke skal bruges.

tiles\_mined er til for at se mængden spilleren har minet på en givet tile.

Bestemmelsen af arrays:

Hertil kan vi nu lave nye chunks omkring os, men vi har ikke kigget på koden bag det endnu, så først kigger vi på chunk maker (se bilag 4), funktionens formål er at lave et int array der skal derefter sendes videre til chunk writer, int arrayet bliver derefter sendt tilbage.

Hvert index bliver givet en værdi ved hjælp af chunk\_terrain, der er en bunke af switches og ifs for at se hvor i chunk koordinatet det er og hvilket index det er, når fundet giver det et bestemt eller random int som afleveres til at indskrive i indexet.

Efter alle indexes er indskrevet bliver Chunk\_writer kaldt inden arrayet bliver tilbagesendt.

Chunk\_writer (se bilag 5) har til formål at tage et int array og nedskrive det array i et txt fil med koordinatlokationen som navn.

Først checker den for eksistens af filen (se bilag 6) (hvis filen eksisterer deleter den), og derefter begynder den på nedskrivning, den laver en string som får tilføjet hvert index af arrayet, mellem hvert index kommer der et komma, grunden kan ses senere. der bliver lavet en ny linje på index 200 og 400 da vi fik en fejl da den indskrev en string længere end længste længde i et txt document.

Vi bruger streamwriter til at skrive dokumentet, da når vi brugte file.create() lukkede programmet ikke filen efter brug, hvilket skabte skrivebeskyttet filer som programmet ikke kan åbne igen eftersom den er åben. Så når vi skulle læse filen, kunne den ikke åbne den, og skabe en fejl.

Læsning af filer sker i Chunk\_Read (se bilag 7), læsningen af filer sker i tilfælde af allerede eksisterende filer når Get\_Tiles bliver kaldt. Formålet med Chunk\_Read er at læse en txt document gennem for alt indskrevet, ændre stringen ind til et int array, og returnere det int array. Måden vi gør dette er at vi først åbner dokumentet med streamwriter (samme grund som når vi laver skriver filer), og læser alt i dokumentet som vi sætter til en string. Herefter køre vi et for loop, for stringens længde. For hver char som ikke er et komma gemmer vi det kortvarigt, og

hvis det er et komma, tager vi den kortvarige gemte del og indsætter den i int arrayet ændret til en int. På denne måde får vi både hvert index ind, men koden kan også håndtere ints der har flere cifre. Derefter bliver int arrayet returneret.

### **Get info fra Terrain klassen**

Den første gode funktion at snakke om er Which (se bilag 8), Formålet med Which er at returnere værdien af det terræn med givet lokation. Måden vi opnår dette er først at bruge den måde vi satte terrænet op på, så først i koden ændre vi de indgivet x og y værdier fra skærm positioner, til x og y positioner der passer til vores chunks, det vil +1 i x (eller y) per tile positionen er længere an.

Efter at positionen er skiftet til terræn type position, så køre vi koden igennem et for loop der checker hvert 32 y position om den er det samme som angivet y position.

Grunden til dette ses hvis vi går tilbage til basis opsætningen, så hvis vi har en x bredde på 3, og en y længde på 4, så ville vores arrays for de 2 værdier se sådan her ud:

X [ 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2]

Y [ 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3]

Hvis vi tjekker for 2 i x arrayet er der mange muligheder, det samme for y arrayet ligesom i et koordinatsystem. Men hvis vi tjekker y hver tredje gang (dermed x bredde) så vil vi kun have et muligt svar hvor 2 er i indexet, hvilket er den tredje gang. Dette er hvad sker i for loopet, og når vi når dertil gemmer vi det antal at gange loopet har kørt igennem, og breaker ud.

Herefter skal vi finde x, og nu ved vi at x skal være rækken af x i tredje gang, så vi bruger vores fundet y index til at tilføje til x når vi leder efter den, men ellers gør vi det samme. Når vi har x og y kender vi dermed indexet for det ønskede terræn, hvor vi kan så sende det tilbage. Dog først skal vi lige checke for hvilken chunk der er ønsket, dette gør vi hurtigt ved at tjekke hver chunk (ikke så slemt, der er maks 4, og det er deres position, så det er maks 4 int arrays 2 lang der tjekkes) Og når den rigtige er fundet kan vi dermed sende index af ønsket terrain tilbage.

Ændringen af terræn sker med Change (se bilag 9), Change gør det samme som Which, med ændringen når funktionen har fundet index for terrænet, så skifter change terrænet, og ender der da Change er en void funktion.

Mining af terræn sker i Mining\_Updater (se bilag 10), hvor den går det samme igen som Which, for at finde index, Når vi har fundet index så køre vi en switch for indexet, som vi har skrevet lidt om tidligere så er vores terræn lavet af arrays af ints, grunden til dette er at vi kan så have hvert index for hver tile i terrænet, vi bruger ints fordi vi bestemmer terrænet ud fra dets nummer, 0 er ingenting, 1 er baggrund osv... Så vi kan se hvilket terræn tilen er. Derfor køre vi en switch der så kan bestemme hvilket terræn for forskellige kode, grunden til flere cases er fordi vi har forskellige mining hastigheder for de forskellige terræner.

Vi checker om terrænet skal fjernes, som vi ser med et float array, float arrayet er 576 lang og er for hvert terræn i chunken, det er lavet for at kunne gemme mængden vi har mined de forskellige tiles. Hvis vi forlader chunken er det meningen at dette arrays bliver sat til nul eller null. Men hvis det index (det samme index som det for terrænet) er over vores givet nummer for hvor hårdt materialet er, så ændrer vi terrænet med Change, ellers øger vi float terrænets værdi med deltatime. Der er få ændringer igennem switchen, såsom at vi tjekker for om terrænet er et materiale, i det tilfælde øges det materiale til klassen Workshop.

### **Player Terrain Interactions**

kollisionen på spilleren bliver checket i Player\_Collis (bilag 11), Player\_Collis bliver kaldt når spilleren bevæger sig i en retning, hvor retningen medsender en int der bruges til at se hvilken side der skal tjekkes på. Funktionen er en bool returtype da vi skal have tjekket om spilleren skal bevæge sig i den retning. Vi bruger ikke standard box collision (så ikke monogame version, eller vores egen), da vores terrain ikke er sat op som objekter eller med højde og bredde, mens at det kan bare løses ved at tjekke for 2 punkter. Vores spil har ikke nogle elementer der er mindre i højde eller bredde end spilleren, så vi kan tjekke med 2 punkter ud fra spilleren om de punkter er hård terræn eller passerbar terræn. Dette går vi dog ikke for op og ned ad collision, da vi tjekker med 3 punkter der. Grunden til dette er at vores kollision er også hvad der kalder mining af terrænet. Så ved at tjekke ud fra midten miner vi først den tile der er mest under eller over os. Dette gøres dog ikke når vi miner til siderne da vi kun kan mine til siderne når man er på jorden.

Tyngdekraften på spilleren bliver lavet i GameWorld, men hvad der tjekker om den skal stoppe er Player\_Collis\_Gravity. Denne funktion er en bool og tjekker om to punkter under spiller er

hårdt terræn ligesom med spilleren, funktionen er en bool returtype da den bare skal tjekke om man rører jorden.

### **User Interface**

Vores spil skal kunne tilpasse sig mobil-markedet. Derfor er det vigtigt for vores spil, at UI'en er simpel, og typisk nem at gennemskue for nye brugere. Da den generelle bruger styrer sin mobil med fingrene, hvilket ofte bliver meget upræcist i sin præcision, er vi nødt til at tage højde for tilgængelighed i vores design. I modsætningen til, hvis det var udviklet til PC, skal vores spil have få, store knapper, som ikke er placeret for tæt på hinanden. Dette gælder fx Escape. Hvis den er placeret for tæt på Movement, vil spilleren opleve at trykke på knappen ved en fejl og derved forlade spillet uden det henseende.

### **Balancing - Bjarke**

Kravet for om et spil er underholdende, bunder ofte i om det er balanceret korrekt. Dvs. at det skal være udfordrende, uden at være opgivende svært. Sværhedsgraden skal stige sammen med spillerens kundskaber og progression. Samtidigt giver det ikke eufori at løse udfordringerne, hvis de er for nemme. I Maersk Miner har vi flere parametre at skrue på.

Der er en naturlig stigning i sværhedsgraden, da spilleren automatisk bliver tvunget til at grave dybere og dybere, med samme mængde batteri. Her er det vigtigt at vi laver upgrades som matcher dybden, så det hele tiden, kun lige akkurat, kan lade sig gøre at skaffe materialerne. Disse upgrades kræver materialer, som vi kan skrue på hvor sjældne skal være, gennem vores terræn generering. Der skal ligge en belønning i at finde dem, men samtidigt skal de ikke være så sjældne at man aldrig får købt den næste upgrade. Da alle vores upgrades bunder i at give mere tid, og tid også er kriteriet for både at vinde og tabe spillet, er tiden det absolut vigtigste sted at balancere. Derudover er der også ting som den generelle spiller-hastighed, som udgør hvor hurtigt maskinen kører, flyver og falder. Denne spiller ind i hvor nemt spilleren har ved fx at nå overfladen, efter at have gravet. Dvs at hvis den kalibreres senere, vil den sætte alle tallene i



Upgrades ud af proportioner. Dette betyder at spiller-hastigheden skal bestemmes som det første, og derefter de individuelle upgrades.

Der er også balancing i at vurdere hvor længe vores gameplay forbliver underholdende, da det i længden er ret repetitivt. Dvs hvor mange materialer skal man samle for at vinde spillet?

Enten skal vi tilføje et twist til gameplayet, eller også skal vi lade spilleren vinde mens legen er god. Et twist kunne være nye gameplay mechanics som at flygte fra NPC's, nye varierende områder eller nogle mere interessante upgrades, som bedre udvikler og motiverer gameplayet.

Alle disse muligheder kunne være realistiske, hvis vi havde mere tid til at udvikle på projektet.

## **Konklusion** - Bjarke

Motherload er fast paced, farverigt og stærkt tidsbaseret. Disse værdier har vi implementeret i Maersk Miner, og undladt elementer som fx. tilt fysik, fall damage og 3D til 2D grafik.

Derudover har vi tilføjet pixel animationer, større områder og (forberedt) collectables, i håb om at det er nok til at gøre vores produkt unikt, samtidigt med at vi kan beholde en del af nostalgien hos målgruppen. Man kunne eventuelt genskabe spillet i en fysik baseret engine, som Unity, og måske få nogle controls som var endnu bedre end Motherload's.

Hvis Maersk Miner bliver udviklet helt færdigt, er der med stor chance en plads på markedet, hvis man tager udgangspunkt i tallene fra de konkurrerende titler.

Hvis spillet bliver en succes, kunne vi gennem Scrap Worthy sagtens udvikle flere, da der klart er et marked for pixel indie spil, og er masser af gamle spil at hylde.

## Litteraturliste

(1) WWW:

Zukalous (2022) How to Market a Game

Lokaliseret d. 07 dec.

[\*One way to actually make money on your first game – How To Market A Game\*](#)

(2) WWW:

Marco (Ukendt) Game design Theory Behind Games

Lokaliseret d. 13 dec

[Mechanics Dynamics Aesthetics\(MDA\): game design theory behind games  
\(gamedevelopertips.com\)](#)

(3) WWW:

SteamDB (2022) Dome Keeper

Lokaliseret d. 13 dec

[Dome Keeper \(App 1637320\) · Steam Charts · SteamDB](#)

(4) PDF:

Anders V. Pedersen (ukendt) Slide #2 (pdf) - Market Analysis.pdf

Lokaliseret d. 13 dec

[Course: Virksomhed \(gr22das122\) \(mrooms.net\)](#)