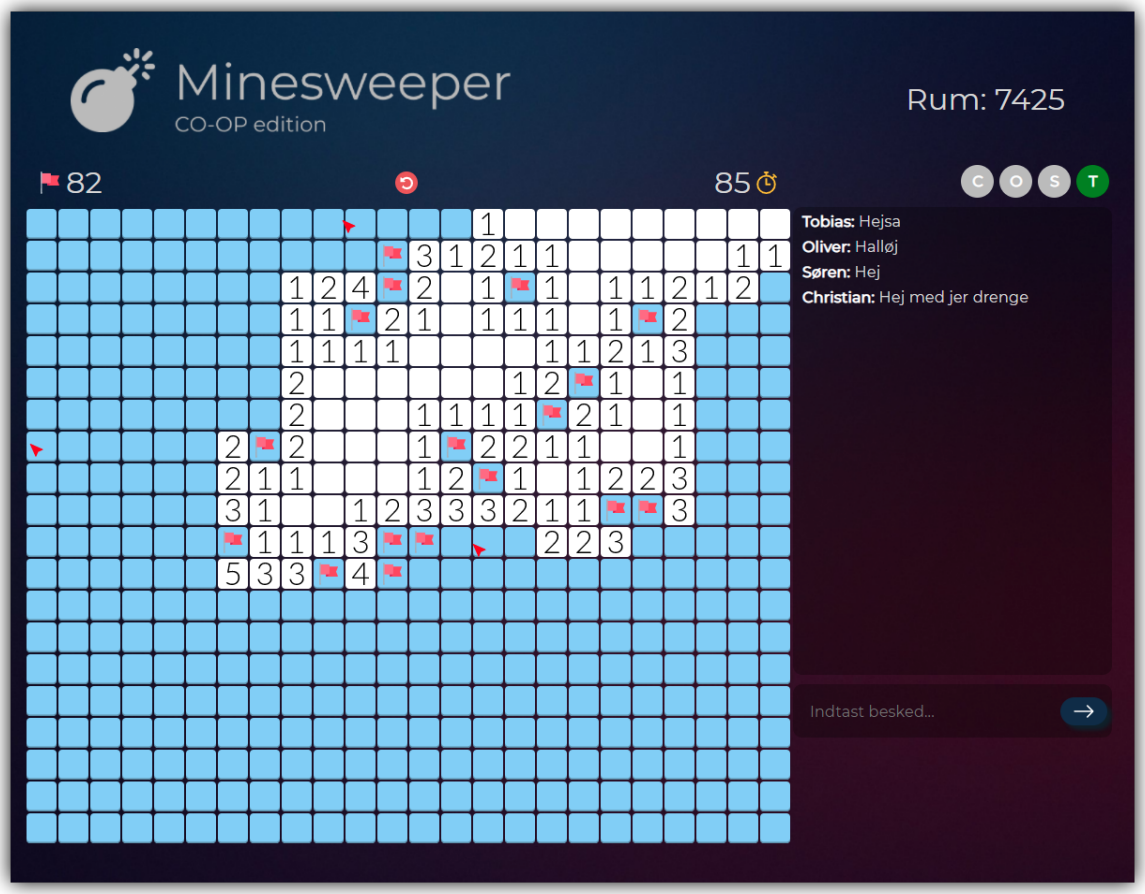
| Group 15 Rapport - CO-OP Minesweeper |

Backend development, operations and distributed Systems

62597

Afleveringsfrist: 11. maj 2021

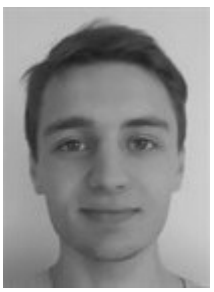


---

https://github.com/CATOS15/PythonMinesweeper, IP: http://130.225.170.81/

Gruppe 15



Christian	*Oliver*	*Søren*	*Tobias*
s184140	*s176352*	*s182881*	*s195458*

Indholdsfortegnelse

1. Indledning

I dette projekt har vi udarbejdet et kooperativt Minesweeper spil “CO-OP Minesweeper”. Backenden er blevet skrevet i Python med en tilhørende frontend udarbejdet i javascript/typescript frameworket VUE. Selve spillet er det klassiske Minesweeper med et gitter af firkanter hvortil man skal finde alle felter hvor der ikke optræder en bombe på. Vi har lavet en co-op version så flere spillere kan spille det simultant. Dette fungerer ud fra man opretter eller tilslutter et rum, hvor den score der opnås bliver gemt ud fra antallet af spillere og sværhedsgraden for det enkelte rum. Highscoren kan findes på forsiden af programmet med alle resultater fra tidligere spil.

Yderligere information omkring teknologierne Bootstrap, Github og git, NPM og design iterationer af produktet kan findes under bilag.

2. Designvalg

I dette projekt har vi skullet træffe en længere række valg af teknologier for at kunne lave et kooperative Minesweeper spil. Her vil vi gennemgå de forskellige teknologier og begrundelser bag.

2.1 Python

Backenden har vi valgt skal skrives i Python, da gruppen ikke har arbejdet med Python før, og ønsker at få erfaring med sproget og dets muligheder. Derudover har Python mange muligheder for biblioteker, herved er det nemt at tilføje yderligere funktionalitet i fremtiden.

2.1.1 Sikkerhed

Vi har taget nogle valg ift. at minimere snyd i spillet. Dette har resulteret i at vi har valgt at køre alt logikken i backend ud fra den simple ide “Don’t trust the client”. Mere præcist, hvis en klient kan tilgå logikken for spillet, vil der altid være mulighed for at snyde eller hacke spillet. Det skal dog siges at klienten stadig har mulighed for at udvikle en ‘bot’ og på den måde snyde. Dette er svært at undgå, men en mulighed ville være kun tillade menneskelige træk. Evt. hvis man mistænker for hurtige træk kunne man stoppe spillet og bede brugeren udfylde en “Am I a robot” test.

Videre muligheder for at manipulere vores program ligger i brugerens mulighed for at kunne lave scripts der modificerer frontenden/sidens udseende. Du kan godt lave et script som modificerer udseendet men som stadig sender de samme requests til serveren. Det vil altså sige at du kan ikke modificere frontend så du opnår en uretfærdig fordel.

Da vores program er opsat så hvert Minesweeper spil ligger i et rum med en række af brugere, er det vigtigt at der ikke er adgang udefra til et hvilket som helst givent rum man ikke er forbundet til. Måden dette bliver garanteret er ud fra ens “sid”, der er en unik identifikation på en bruger. “sid” også kaldt sessions id kan ikke imiteres og hvis man prøvede at ændre sit session id ville man få afvist ens kald til serveren, da serveren ikke længere genkender den session.

Sidstnævnt har vi ikke nogle chatfiltre eller andre check på hvad der bliver skrevet i chatten eller den frekvens af beskeder der bliver sendt til chatten og i resten af programmet. Dette ville kunne være et problem i form af overload af beskeder og requests til backenden indtil vores program crasher. Det er klart en forbedring der skal laves til fremtiden.

2.1.2 Session-baseret highscore, room-data

I et spil giver det mening at have en highscore, men da det er noget vi kun implementerer for det sjove aspekt, og ikke behøver at gemme vores highscore til fremtiden, gav det mening kun at have det i en session. Det vil medføre at, hver gang vi genstarter serveren, vil hele highscoren blive slettet og startet på ny.
Det vil gøre det nemmere at implementere, da vi med en sessionsbaseret liste ikke behøver opsætte og forbinde til en database.

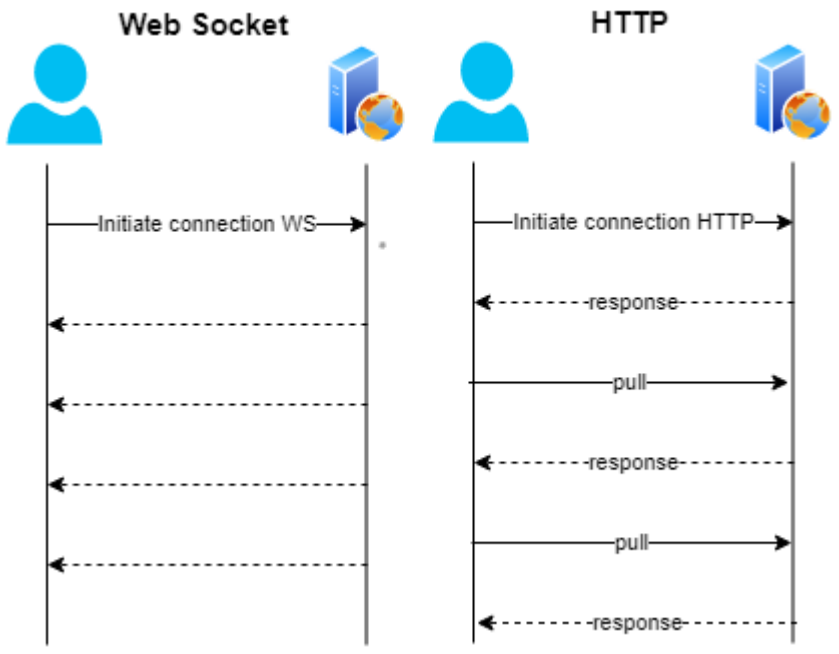
2.2 Typescript med VUE

Frontenden har vi valgt skal skrives i Typescript med VUE. Modsat normal javascript, er typescript type stærkt, hvilket gør det nemmere at holde styr på modeller. Derudover er sproget kompileret, hvilket gør det nemmere at finde compiler fejl, dog kan der stadigvæk opstå runtime fejl. Andre muligheder for javascript/typescript frameworks kunne være Angular eller React men da vi allerede har prøvet at lave et projekt ved brug af angular og ville lave en single page applikation passede VUE rigtig godt til dette projekt.

VUE fungerer reaktivt hvilket betyder at det opdaterer html elementer der er bundet til en model når modellen opdateres. Dette passer rigtigt godt til vores projekt omkring delen med at opdatere vores Minesweeper elementer og bare generelt andre elementer af siden på en simpel og ligetil måde. Videre vil det være hjælpsomt og smart at gøre brug af VUE's mulighed for at skabe genbrugelige komponenter til at opdele de forskellige dele af vores sider og specifikke elementer såsom spil og chat.

2.3 WebSocket - SocketIO

Da projektet skal være en online og live oplevelse mellem brugere ønsker vi en løsning der kan supportere dette. En måde at gøre det på er ved at lave long polling, continues polling eller benytte WebSockets. Vi har valgt at benytte WebSockets der udnytter en såkaldt socket til at oprette en tovejs forbindelse mellem klient og server. Dvs at frontenden vil være i stand til at kommunikere med backend samtidig med at backenden kan kommunikere med frontenden. Det specielle med WebSockets er at det kun kører over HTTP som altså er en TCP/IP protokol.



Billede 2.3.1 WebSocket vs continuous polling[1]

En af årsagerne til at vi har valgt at benytte WebSockets frem for en polling løsning er at det går væsentligt hurtigere end en polling løsning. Med en continues polling løsning ville vi være tvunget til at konstant at spørge backend om ændring og med en long polling løsning ville vi være tvunget til at hænge backend til et eventuelt opdatering kunne sendes tilbage. Når man sidder i et spil, Minesweeper, så skal responstiden på når en bruger laver et træk gerne være øjeblikkelig hos de andre brugere i samme spil og dette opnås bedst med WebSockets.

Vi har valgt at benytte Flask-SocketIO, da det har et velskrevet dokumentation. Desuden simplificerer det meget af arbejdet, da man blot kan skrive 4 linjer kode og så kan man kommunikere fra backend til frontend og fra frontend til backend altså en såkaldt tovejs kommunikation også kaldet fuld-duplex.

[1] https://ambassadorpatryk.com/2020/03/publish-web-socket-in-the-experience-layer/

Vi benytter Flask, som normalt benyttes for at udvikle et REST API, men i vores tilfælde benytter vi ikke den standard form for API. Alt vores kommunikation med backend sker via WebSocket altså Flask-SocketIO der er en udvidelse til Flask. Ændringen er ikke markant, da det blot resulterer i at kald til backend vil se ud som dette `@socketio.on("leftclick")` i stedet for det klassiske Flask API interface der ser ud som følgende `@app.on("leftclick")`

I frontend er der ligeså implementeret en WebSocket løsning. Her har vi via NPM, der er en pakke manager, installeret 'socket.io-client'. Da vi har valgt SocketIO i backend er det naturligvis også SocketIO der er valgt i frontend, hvilket simplificerer arbejdet og allerede har de nødvendige funktioner for at benytte WebSocket.

Vores frontend vil blive distribueret til de brugere der tilgår vores hjemmeside. I praksis vil brugerens computer også opføre sig som en slags "server/backend", da den Python backend der ligger på serveren nemlig kan sende direkte beskeder til brugeren.

2.4 Konfigurationsstyring

2.4.1 NGINX for frontend og backend

NGINX er en webserver, der kan bruges til at hoste vores frontend og backend.

For at implementere vores program i et produktionsmiljø har vi valgt at gå med NGINX. Dette valg er truffet, da NGINX blandt andet tillader reverse proxy der kan bruges til at håndtere WebSocket. Vi kunne også have valgt at bruge HAProxy der ligeså tillader reverse proxy, men da NGINX allerede er installeret på serveren og da den er veldokumenteret og meget benyttet valgte vi at gå med NGINX. NGINX benytter også load balancing hvilket gør at netværkstrafik bliver distribueret ligeligt mellem ressourcer på serveren. Det vil sige at trafikken bliver fordelt til de ressourcer der har mulighed for at løse dem.

3. Implementeringsvalg

3.1 Python

I backend har vi haft nogle ændringer, som vi først fandt ud af under udviklingsfasen. Disse har vi så diskuteret og fundet løsninger til for at udbygge vores produkt eller forbedre dele som kunne optimeres.

3.1.1 MySQL - database

Vi valgte i vores designvalg at bruge sessions, hvor grunden bl.a. var at der ikke var behov for at gemme highscoren ved server genstart. Den holdning har ændret sig, da vi ønsker nu at bibeholde highscores permanent, ved at gemme dem i en database. Der skulle så vælges hvilken database der skulle bruges, som kunne opbevare dataen. Grunden til MySQL blev valgt, var kendskab til sproget, og at det er struktureret hvorimod f.eks. MongoDB er ustruktureret og tager input i JSON format. Udover det så skal der ikke meget arbejde til for at oprette en forbindelse fra Python til en MySQL-server.

Et hurtigt interessant kodeeksempel der viser hvor simpelt det er at benytte Python til at eksekvere en MySQL kommand.

```
def get_highscores(self):
    self.cur.execute("SELECT * FROM Highscore")
    return self.cur.fetchall()
```

Billede 3.1.1 SQL kald

3.2 Rettelse til SocketIO

Under implementeringen af SocketIO gik det op for os at implementeringen faktisk ikke benyttede WebSocket i sin korrekte form. Den havde faktisk lavet et fallback til “continues polling”, og dermed var vores responstid markant øget når man foretog sig et valg i spillet. F.eks det at klikke på en brik tog over 150 ms før at resultatet blev vist på tværs af de klienter der var tilsluttet spillet.

Vi kunne reducere responstiden markant ved at få WebSocket til at virke korrekt. Vi fandt ud af at Flask-SocketIO havde behov for et bibliotek for at fungere korrekt. Biblioteket ‘eventlet’ løste dette ved at have en “concurrent networking” løsning. Denne løsning gav mulighed for at have en direkte non-blocking IO forbindelse mellem frontend og backend og dermed opstod SocketIO’s fallback til “continues polling” ikke længere.

4. Afprøvningsvalg

4.1 Load testing

Load testing af applikationen er essentiel, da hver sockets kræver andel af serverens kapacitet. Det er derfor vigtigt at vide, hvor mange samtidige brugere systemet kan supportere, hvordan applikationen håndtere fejl, om infrastrukturen kan holde til det givne load og for at have tillid til systemets ydeevne.

Som udgangspunkt fandt vi et test bibliotek Artillery.io, hvilket specifikt kunne teste SocketIO. Grundet vi har benyttet en nyere version af SocketIO, er det ikke muligt at bruge Artillery.io, da den benytter en ældre protocol. Dette problem har generelt gjort det svære at test programmets sockets. Senere vil det give mening at ændre SocketIO protocollen, for at simplificere testningen. Programmet er derfor blevet load testet manuelt, ved åbning af mange instanser i en browser, hvilket serveren klarede fint, men dette giver ikke et fuldt billede. I fremtiden skal vi altså derfor inkluderer yderligere load testing.

4.2 Sikkerhedstest

Kommunikationen mellem server og klient foregår gennem WebSockets, derfor har vi specielt fokus på den i vores sikkerhedstest. Det er specielt vigtigt for os, at de forskellige spil er isoleret for hinanden, så de ikke kunne påvirke andres spil. Her testede vi, om det var muligt at sende beskeder til rum, man ikke selv var en del af, hvilket ikke var muligt grundet validering i backenden.

4.3 Frontend Test

For at give den bedste brugeroplevelse, har vi lavet nogle test på bekendte uden nogen kendskab til programmet. De fik ip adressen til spillet, og fik ikke andet at vide end det og reglerne for spillet, som er standard reglerne.

Test person:
Jeg synes startsiden er relativ overskuelig. Der er kun de funktioner, som man har brug for og er tydeligt hvad knapperne vil gøre. Selve spil siden er også simpel, og har kun det som er nødvendigt, dog var der et problem med at se rummets nummer grundet skærm størrelsen på computeren.

5. Idrifttagnings Valg

Vi har fået tildelt Linux serveren med IP'en *130.225.170.81*

5.1 NGINX for frontend og Linux Service for backend

Som beskrevet i vores designvalg valgte vi at gå med NGINX for at implementere vores program i et produktionsmiljø. Undervejs i løbet af opsætningen af NGINX for at hoste vores sites gik det op for os at Flask faktisk er en webserver i sig selv. Dette udløste nogle problematikker i form af WebSocket der ikke fungerer optimalt gennem reverse proxy fra NGINX til hosten på Flask. Vi endte derfor med at gå med en simpel service løsning på Linux serveren for at hoste backend. Frontend forblev hostet i NGINX.

Backend og frontend er blot lagt på brugeren "/home/oliver" på Linux serveren. Optimalt set ville man nok oprette en system bruger der stod for at have produktionskoden liggende.

Frontenden produktionskode ligger i "/home/oliver/dist". Fra NGINX "/etc/nginx/sites-enabled" findes konfigurationsfilen "frontend" der netop peger på "/home/oliver/dist". Da frontend er single-page omdirigeres alt forsøgt trafik til roden af URL'en på porten 80. Dvs forsøger man at ramme *http://130.225.170.81/** , hvor stjerne betyder alt, så vil man blive omdirigeret til *http://130.225.170.81/*

Da der var problemer med opsætning af backend gennem NGINX, fordi NGINX ikke kunne navigere source porte korrekt gennem WebSockets selv med reverse proxy, resulterede dette i at backend blev lavet som en service, hvilket også passede fint da Flask faktisk er sin egen webserver.

Backend er sat op ved at benytte et virtuelt Python miljø som findes i "/home/oliver/backendenv". Dette miljø gør at Python biblioteker er afgrænset fra eventuelle andre Python miljøer. På den måde kan du have flere "sites" opsat på en server med forskellige biblioteker der ikke overlapper hinanden. Selve Python koden ligger i "/home/oliver/backend" og der er oprettet en service fil i "/etc/systemd/system/backend.service" der står for at køre backend.

Når vi er kommet frem til et stadie hvor vi vil udgive koden til serveren har vi valgt at udvikle nogle udgivningsscripts der gør dette for os.

```
cd \PythonMinesweeper\frontend
call npm run build
scp -P 22022 -r dist oliver@130.225.170.81:

cd \PythonMinesweeper\backend
scp -P 22022 -r backend oliver@130.225.170.81:
ssh -p 22022 -t oliver@130.225.170.81 "sudo systemctl restart backendpython"
pause
```

5.2 MySQL - database

Det var lige til at opsætte MySQL på serveren, hvilket også er en af årsagerne til vi har valgt det, da det blot er en ændring af en enkelt linje i konfigurationsfilen for at åbne op så databasen kunne tilgås udefra. MySQL er sat op så det blokerer alle andre IP'er end dem fra vores udviklingsmiljø og serveren selv.

5.3 Docker

I fremtiden ville det give god mening at implementere docker for let at kunne distribuere vores produktionsmiljø til en anden server. Hvis vi skulle ønske at have enten en distribueret løsning eller vi skulle rykke hele vores produktionsmiljø ville det nemt kunne gøres hvis vi havde docker opsat på serveren.

Grundet tid og andre fokusområder i projektet er dette ikke nået.

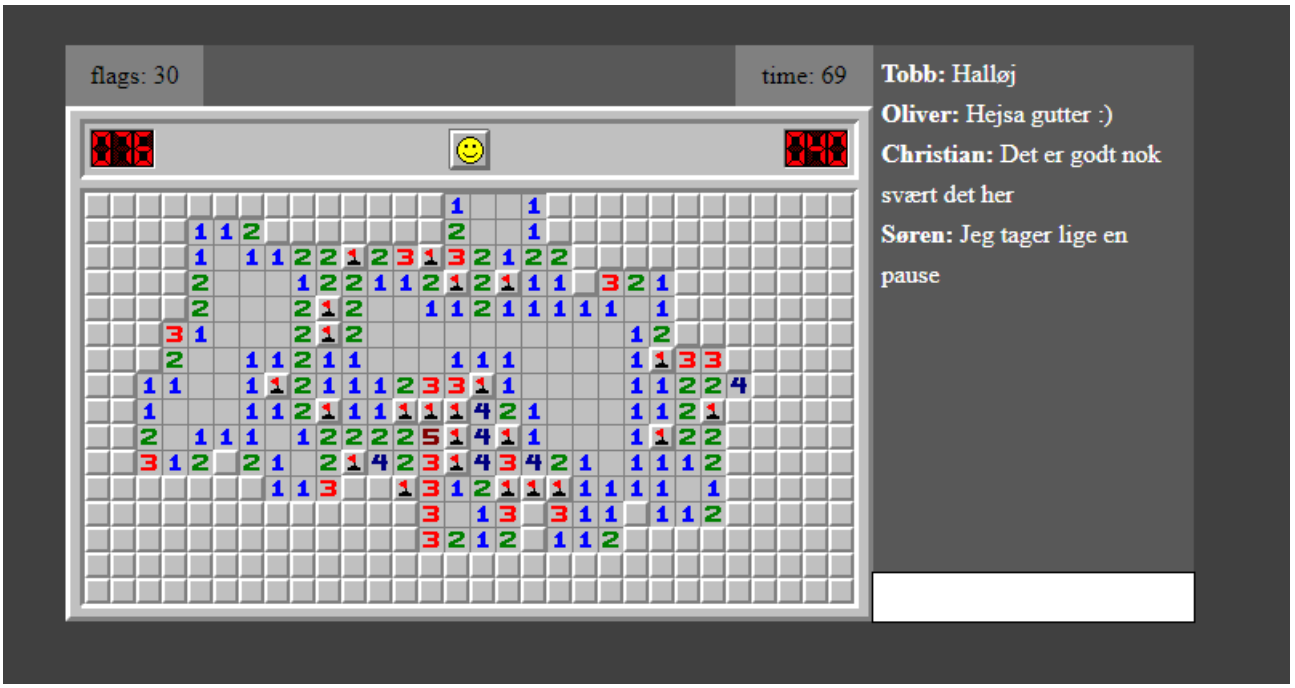
6. Konklusion

Projektet har i sin store helhed været en success og vi har fået nået de ting vi gerne ville på en meningsfuld måde. Der er stadig mange muligheder for at viderebygge vores program og spil. Dette kunne være i form af flere gamemodes, animationer, public spil, chat commands og meget andet. Dog har vi fået implementeret det vi mente var vigtigt og som vi lagde mest vægt på.

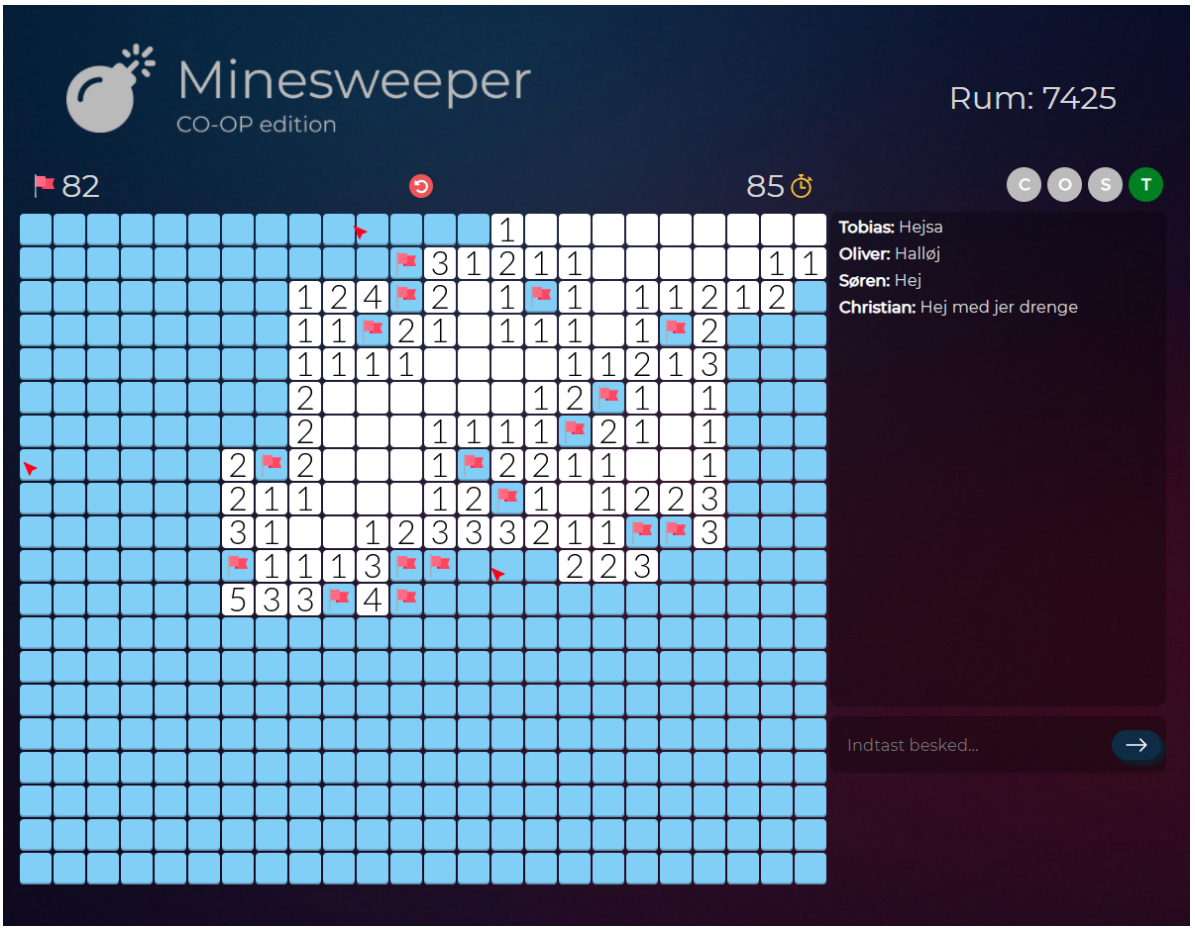
Vi har fået gjort brug af nogle nye teknologier, vi ikke havde så meget erfaring med såsom Python, VUE, Flask og meget andet. Af de teknologier vi har valgt, har de generelt passet godt til projektets scope og været velegnede til at løse de problemstillinger vores projekt omhandler. Flask-socket.IO har været godt til kommunikationen mellem brugerne og VUE har været et godt framework til vores frontend. Vi lærte Python er specielt godt til simpel funktionalitet, og grundet de mange tilgængelige biblioteker, kan man hurtig tilføje meget funktionalitet på få linjer. Dog kan man risikere at blive begrænset af biblioteket, ved at miste kontrol og dermed være nødsaget til at lave nogle mindre attraktive workarounds. Vi har også konkluderet at vi foretrækker et mere typestærkt sprog i backend såsom C# eller Java.

7. Bilag

7.1 Design iterationer



Endelige design



7.2 Information om yderligere teknologier og valg

7.2.1 Layout - Bootstrap

Der blev valgt at bruge bootstrap under implementering, da vi ville simplificere noget af det HTML og CSS kode. Bootstrap er noget vi har arbejdet med før, og havde gode oplevelser med. Det har allerede pre definerede termer, som gør det hurtigere at style end at skulle gøre det fra bunden. Ud over det, hjælper det også med at mindske cross-browser fejl.

7.2.2 NPM

NPM er en Package Manager til javascript og bliver brugt til at automatisere vores pakke styring for de dependencies vi nu engang har i vores projekt. Eksempler på disse er vores brug af VUE, Bootstrap Socket.io osv. Dette gør det også nemmere for os at installere projektet på nye maskiner, opdatere og installere nye tilføjelser til projektet som de måtte komme under udviklingen. Der er andre Package managers men NPM har meget dokumentation og passer godt med mange af de komponenter vi gør brug af.

7.2.3 Github og Git

For at kunne arbejde på projektet sammen som en gruppe har vi skullet bruge et versionsstyring værktøj samt en måde arbejde på de samme filer på tværs af medlemmer. Hertil har vi valgt github og dermed git først og fremmest på grund af vores tidligere erfaring med de to værktøjer samt deres meget gennemgående dokumentation og globale anerkendelse. Andre alternativer ville kunne være SVN, Azure portal eller Mercurial. Disse andre alternativer er dog mindre brugte og hvis sammenlignet med SVN er SVN mere simpel til simple ting men github er bedre med mere komplekse værktøjer såsom branching og merging.