

Programmering Eksamen

Af Mads T.F. Knudsen

[Forblad: 3](#_Toc38056717)

[Abstract: 3](#_Toc38056718)

[Problemformulering: 3](#_Toc38056719)

[Funktionsbeskrivelse: 3](#_Toc38056720)

[Implantation: 4](#_Toc38056721)

[Tale Til Tekst: 4](#_Toc38056722)

[Database: 4](#_Toc38056723)

[Åbne hjemmeside: 5](#_Toc38056724)

[Start program: 5](#_Toc38056725)

[Kommandoer: 5](#_Toc38056726)

[Handlinger: 7](#_Toc38056727)

[Open URL: 7](#_Toc38056728)

[Hide / Show AI Log: 7](#_Toc38056729)

[Open External Application: 7](#_Toc38056730)

[Mute / Unmute: 7](#_Toc38056731)

[Main / Command Screen: 7](#_Toc38056732)

[Flowchart: 8](#_Toc38056733)

[Test: 9](#_Toc38056734)

[Tale til tekst: 9](#_Toc38056735)

[Handle med eksterne programmer: 9](#_Toc38056736)

[Eksport til WallpaperEngine: 9](#_Toc38056737)

[Konklusion: 9](#_Toc38056738)

# Forblad:

Mit programs navn, har jeg valgt at kalde det Hall, som er en reference til computerintelligensen fra ”A Space Odyssey”, som er kendt for at sige ”Im afraid I cant do that Dave” og være et sort rektangel med et rødt øje.

Dette projekt er lavet af Mads T.F. Knudsen.

Dato: 17 / 4 / 20

GitHub link:

<https://github.com/Mfknudsen/ProgrammeringsEksamen>

# Abstract:

Jeg har til min programmerings eksamme valgt at arbejde alene om at lave en simpel computer assistent, som skal kunne forstå brugerens tale og kunne påføre forskellige handlinger, som f.eks. at åbne en hjemmeside og starte et program. Igennem projektet har jeg lavet et Unity program ud fra min ide om en computer assistent, som bruger C# til kode og forskellige funktioner i Unity, samt skrevet en rapport ud fra dette program.

# Problemformulering:

Mit produkt, Hall, er lavet med den hensigt, at den skal kunne tage forskellige stemme kommandoer og påvirker forskellige handlinger.

Den skulle starte op før login til ens desktop, så man kunne evt. logge ind gennem den, eller lige efter login til desktop. Den vil derefter begynde at lytte gennem en tilgængelig mikrofon efter kommandoer. Når den har fået et input fra mikrofonen, skal den tjekke efter, hvorvidt det er en af dens kommandoer, som den skal have gemt på computeren. Den skal derefter påføre en handling ud fra den kommando der blev givet. Den skal vide hvor mange kommandoer der er i gang med at påføre, samt hvornår den er færdig med en kommando. Til sidst skal brugeren kunne lave nye kommandoer, som bliver gemt når programmet slutter og bliver automatisk opsat igen efter at den starter op. Programmet skal gøre dette som et baggrundsprogram og gennem Wallpaper Engine, vil brugeren kunne påvirke produktet uden at den skal være i fokus, samt så den ikke bliver vist som et åbent program i brugerens taskbar.

# Funktionsbeskrivelse:

Programmet kan starte alle de forskellige dele af dens systemer, uden fejl der stopper dens funktioner. Den kan modtage input fra en mikrofon og tjekke for kommandoer i det resulterende tekst og påføre handlinger ud fra kommandoerne, samt skabe nye kommandoer og gemme disse nye kommandoer og loade dem igen når den starter op igen.

# Implantation:

## Tale Til Tekst:

Når man koder med Unity, får man flere pakker af kode, som indeholder forskellige funktioner og giver et bedre workflow og går det nemmere at lave kode som påvirker et Unity program. Unity har flere forskellige pakker som indeholder forskellige funktioner, der gør det muligt for Unity, at genkende forskellige ting. For tale til tekst, som jeg bruger i mit produkt, bruger jeg pakken kaldet ”Speech”, som man kan hente til ens script ved at sætte ”using UnityEngine.Windows.Speech”. Denne pakke kan kun bruges med Windows, samt at Windows indstillinger er sat til at understøtte tale til tekst, samt at Unity programmet er indstillinger til at kunne bruge en mikrofon.

Når programmet bliver startet op, vil den opsætte en ny ”DictationRecognizer”, som har fire forskellige komponenter, som kan give et output.

Den første der bliver sat, ”.DictationResult”, er selve resultatet, som den har fundet ud fra inputtet fra mikrofonen, samt når den tror at den har fået hele sætningen. Den er outputtet fra denne, som bliver sendt til kommando kode, som vil tjekke for om der er givet en kommando.

Den anden der bliver sat op, er ”.DictationHypothesis” som et output der viser hvad den tror der er blevet sagt ud fra hvad den har fået. Forskellen mellem denne og resultatdelen, er at den giver et output før den tror at sætning er færdig. Outputtet fra denne del, kan bruges til at få en forståelse af, hvordan computeren opsætter en sætning.

Den tredje er ”.DictationComplete”, som ikke giver en tekst ud fra et input fra mikrofonen, men i stedet giver et output, som beskriver hvorfor ” DictationRecognizer” er stoppet. Den reagerer både en besked om at den skal stoppe, samt hvis der er en fejl, som får den til at stoppe.

Den fjerde er ”.DictationError”, og vil give et output ud fra en fejl som den kan støde på mens ”DictationRecognizer” prøver at tjekke inputtet fra mikrofonen.

Unity vil sætte så mange ting på pause som den kan, mens programmet ikke er i fokus på computeren, og det vil stoppe ”DictationRecognizer” med et resultat af en fejl. Mit produkt starter derfor en ny når det sker, som Unity ikke stopper indtil man igen stopper med at have fokus på Unity programmet. Mit produkt vil derfor stadig kunne reagere på tale, også selvom det ikke er i fokus.

## Database:

Eftersom Hall starter med forskellige kommandoer, som er nødvendig og påvirker forskellige dele af dens systemer, men det er dog muligt for brugeren, at tilføje flere kommandoer. Disse kommandoer er gemt i en liste af forskellige ”dictionary”, hvor man sætter forskellige værdier til en nøgle. Jeg convertere så de forskellige nøgler og laver dem om til et ”array” af ”string”. Hver værdi til de forskellige nøgler er et ”array” af ”string”. Jeg omdanner de forskellige ”strings” til en enkelt ”string”, og når jeg har gjort dette for alle, bliver de gemt i et nyt ”array” og begge vil blive gemt i et object. Dette vil blive gjort for hver kommando.

Bruger så Unitys indbyggede funktionen kaldet ”JsonUtility.ToJson(’object’)”, som omdanner objekterne til Json tekst, som så bliver gemt i en tekstfil.

Når den skal læse hvad der er blevet gemt, så bliver funktionen ”JsonUtility.FromJson<’object’>(’inputString’)” brugt til at omdanne teksten som den aflæser fra tekstfilen, til de samme typer objecter, som da de blev gemt.

Når den så starter på ny, vil den lave en ny liste af kommandoer ud fra de aflæste objekters værdier.

Objektiverne som bliver til at gemme, er alle ”Serializable”, hvor Serialization er en måde, hvorpå Unity kan automatisk gemme og rekonstruere. Data kan normalt ikke gemmes direkte i Unity og objekterne er derfor nødvendige for at gemme og indlæse data.

## Åbne hjemmeside:

Unity har indbyggede funktioner og metoder, hvorpå man kan åbne og kommunikere med forskellige hjemme sider.

Hall kan åbne hjemmesider ved at starte en ”Corotine” ud fra en ”IEnumerator”, som er en funktion der kan sættes på pause baseret på forskellige typer ”yield”, som f.eks. an være en ”bool”. Når Hall skal åbne en hjemmeside, så bliver den sat på pause indtil den får et resultat tilbage fra hjemmeside.

## Start program:

For at åbne et eksternt program, har Unity igen indbyggede funktioner og metoder, som kan bruges.

Efter en ny proces bliver skabt, bliver den givet et filnavn, som vil inkludere filtypen, så f.eks. ”Discord.exe”. Starter derefter en ny proces ud fra den skabte proces.

Den kan også åbne spil som er downloade fra Steam. Når et spil eller et andet typeprogram bliver downloades fra Steam og vil derved åbne gennem Steam, så kan man se gennem de forskellige genveje som bliver oprettet på skrivebordet, ikke har en .exe fil, men i stedet viser den i stedet f.eks. ”steam://rungameid/440”. For f.eks. Origin, Ubisoft og Blizzard, bruger de stadig .exe. Mit produkt kan åbne begge typer.

## Kommandoer:

Kommandoerne som mit produkt bruger er bygget op af tre dele.

”command” er den første del, og indeholder en tekst, som den tjekker overens med den givet tekst fra tale til tekst og hvis de to matcher, så vil kode bruge delende til at finde din givet handling der hører til, samt al den information handlingerne skal bruge til at fungere.

”commandType” indeholder to tekster, hvor den første fortæller hvilken handlinger der hører til den givet kommando, og den anden fortæller hvorvidt bruger kan lave om på kommandoen. Hvis der er skrevet ”false”, så kan bruger ikke påvirke den, mens bruger kan påvirke dem med ”true”, og alle de kommandoer som brugeren selv har skabt, bliver sat som ”true”.

”commandInput” indeholder f.eks. hjemmesiden som den skal åbne eller det program den skal åbne.

## Handlinger:

Alle handlinger burger den samme metode til at holde oversigt. De modtager hver deres input og skaber et objekt og tilføjer et ”Action” component, som holder oversigt med dens tilsvarende handling, samt en component som er specielt lavet til hver kommando.

### Open URL:

Modtagere den givet kommando for at tjekke hvilken hjemmeside der skal åbnes, samt inputtet som selve kommandoen holder.

Den special lavede component som denne handling giver objektet det skaber, er koden ”Action\_OpenURL”, som bliver beskrevet under Synopsis – Implantation – Åbne hjemmeside.

### Hide / Show AI Log:

Modtagere den givet kommando, samt det input den skal bruge for at fungere.

Den special lavede component som denne handling giver objektet det skaber, er koden ”Action\_HideShowAILog”, og modtager en ”bool” der bestemmer hvorvidt objektet som holder loggen skal være aktiv eller ej og bruger kan derved se eller ikke se den ud fra dette.

### Open External Application:

Modtagere den givet kommando, samt det input den skal bruge for at fungere.

Den special lavede component som denne handling giver objektet det skaber, er koden ”Action\_OpenExternalApp”, som bliver beskrevet under Synopsis – Implantation – Start Program.

### Mute / Unmute:

Modtagere den givet kommando, samt det input den skal bruge for at fungere.

Den har ikke sit eget handlings kode, men ændre en ”bool” i kode, som bestemmer hvorvidt den skal reagere på teksten den for fra tale til tekst. Den vil dog stadig reagere hvis kommandoen er det modsatte af hvad ”bool” værdien er.

### Main / Command Screen:

Modtagere den givet kommando, samt det input den skal bruge for at fungere.

Den har ikke sit eget handlings kode, men sender i stedet en ”bool” til UI kode holderen, som vil skifte mellem hvilken hvad der er aktivt og derved, hvad brugeren kan se.

## Flowchart:

Et billede, der indeholder tastatur

Automatisk genereret beskrivelse

# Test:

## Tale til tekst:

Denne del var det første jeg fik lavet i mit produkt og har senere hen, været en stor del i arbejdes processen, eftersom jeg ikke lavede en metode hvorpå jeg kunne indtaste en tekst for at teste kommandoerne.

Jeg fandt nemt en metode hvorpå man kunne implementere tale til tekst i et Unity program, da dem bag Unity har skrevet hvordan på deres dokumentations hjemmeside.

Jeg støtte på to store problemer og et mindre problem, som ikke ødelægger produktet. Det første store problem var, at den ville stoppe automatisk efter x antal sekunder hvor den ikke fik noget nyt input fra mikrofonen. Hvor lang tid det var, kunne man ændre via ”DictationRecognizer.InitialSilenceTimeoutSeconds”, som jeg satte til at være -1, som gør at den aldrig vil stoppe, hvis den ikke for noget input. Det andet store problem kom ved at den ville stoppe når programmet ikke var er fokus, som jeg også beskrev tidligere i Synopsis – Implantation – Tale Til Tekst.

Det mindre problem er hvor god den er, og den har vist flere gange at den har svært at forstå en via en headset mikrofon, når der blev spillet lyd igennem det.

Udover det, er det dog en succes efter som man kan aktivere handlingerne ud fra tale til tekst.

## Handle med eksterne programmer:

Eftersom mit produkt kan åbne et eksternt program uanset hvilke type fil det er, men ikke kan påvirke dem, vil jeg sige at denne del også er en succes, dog kun lige.

## Eksport til WallpaperEngine:

At eksportere programmet og have det køre i Wallpaper Engine, viste sig ikke at være svært at gøre, eftersom programmet understøtter Unity programmer til at køre gennem det. Jeg støtte dog på et problem med at eksportere mit produkt, som gør at det ikke lykkedes for mig at køre mit produkt gemme Wallpaper Engine, så denne del fejlede.

# Konklusion:

Jeg har lavet et produkt som kan tage input fra en mikrofon, omdanne det til en tekst og starte en handling ud fra denne tekst, samt en rapport ud fra dette produkt.