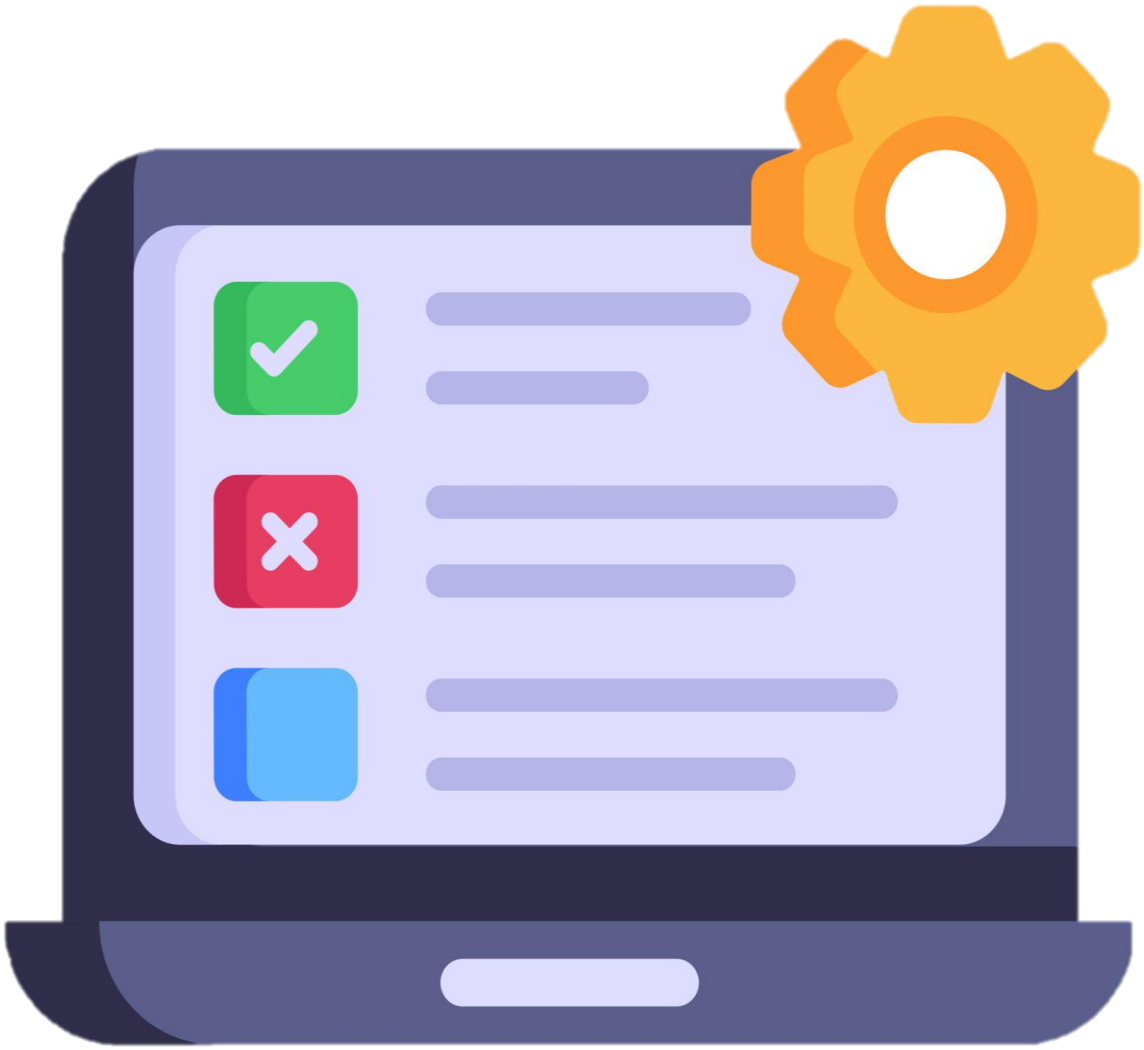
H1 – Løsningsforslag i Grundlæggende Programmering



Dette dokument er et løsningsforslag til "Afsluttende Projekt i Grundlæggende Programmering" på H1. Formålet er at beskrive en konkret, gennemførbar plan for en C# console-applikation: overblik over funktionalitet, tekniske valg, struktur, og arbejdsgang. Dokumentet fungerer som grundlag for implementering, test, versionskontrol og aflevering på GitHub.

Contents

Introduktion

Opgave Beskrivelse

Til vores grundlæggende programmering projekt, har vi fået til at opgave at programmere en console applikation ved brug af C#. Console-applikationen skal kunne tilføje opgaver med en grænse på maksimalt fem ad gangen, vise alle opgaver både færdige og ikke færdige, markere en valgt opgave som færdig ud fra et nummer samt afslutte programmet på en pæn måde. Opgaver og deres status skal gemmes i et array eller en liste, og koden skal organiseres med metoder, loops, betingelser og switch-case. Input og output foregår via konsollen med tydelige beskeder til brugeren. Projektet skal versionsstyres med Git og afleveres på et public GitHub-repository med meningsfulde commits. Som ekstra funktioner kan der implementeres mulighed for at slette opgaver, gemme og indlæse opgaver fra en tekstfil, sortere færdige opgaver nederst og bruge farver i konsollen til at vise status.

Teknisk Gennemgang

Udviklingsmiljø

Til udviklingen anvender jeg *"Visual Studio Professional Edition"*, version *"17.14.11"*.

Teknologivalg

Projektet er implementeret i *"C# .NET Core 9"*. Version 9 er ikke en LTS-udgivelse, men til mindre applikationer uden krav om langtidssupport vurderer jeg, at fordelene ved at kunne udnytte nye, eksperimentelle funktioner opvejer dette.

Versionsstyring

Kildekoden versionsstyres via "Git" og hostes på *"GitHub"*, som er en udbredt og pålidelig platform til opbevaring af repositories. Alternativt kunne *"Azure DevOps"* benyttes, som integrerer særdeles godt med Visual Studio, men i dette projekt er GitHub valgt for at sikre enkel deling og tilgængelighed.

Arkitektur og design

Applikationen er en konsolbaseret applikation, hvor brugergrænsefladen er opbygget omkring et *"side-baseret navigationssystem"*. For at understøtte dette har jeg implementeret en abstrakt basisklasse kaldet *"Page"*, som alle sider i systemet nedarver fra.
Denne arkitektur giver følgende fordele:

- Understøttelse af titler og beskrivelser pr. side
- Mulighed for tilføjelse af argumenter med visning
- Et inputsystem, der kan eksekvere metoder baseret på brugerinput
- Høj grad af konfigurerbarhed, så hver side kan tilpasses med sin egen eksekveringslogik

Estimat af softwarens levetid

Forretningsmål
Jeg vurderer min applikation til en score på 3. Denne vurdering skyldes, at applikationen er udviklet med en høj grad af modularitet, både i den generelle sideopsætning og i den hovedmenu, der er blevet udviklet. Dette gør det nemt at integrere nye teknologier og løsninger i applikationen. I hovedmenuen er der allerede flere funktioner under udvikling.

Teknologivalg
Jeg vurderer min applikation til en score på 3. Dette skyldes, at jeg udelukkende anvender C# .NET Core 9 sammen med Windows' standard WIN32-bibliotek. Denne kombination er stabil for nuværende, men ikke nødvendigvis optimal på længere sigt. Hvis jeg anvendte en LTS-version af .NET, ville jeg vurdere scoren til at være 5.

Afhængigheder
Jeg vurderer min applikation til en score på 3. Den eneste eksterne afhængighed er det native WIN32-bibliotek. Hvis dette blev fjernet, ville scoren være 5, da C# .NET Core 9 er særdeles stabilt.

Ændringer i feltet
Jeg vurderer min applikation til en score på 5. Applikationen er en simpel C#-konsolapplikation med et opgavesystem samt et klassisk pingpong-spil. Der er ingen forventede lovgivningsmæssige ændringer, som vil påvirke applikationen.

Vedligeholdelsesbudget
Jeg vurderer min applikation til en score på 5. Som nævnt tidligere er systemet udviklet modulært, og der er allerede forberedt flere værktøjer og funktioner, som gør det muligt at videreudvikle applikationen effektivt.

Teknisk kvalitet
Jeg vurderer min applikation til en score på 3. Koden er velstruktureret, og der er udarbejdet tilhørende dokumentation. Dog er der ikke gennemført omfattende enhedstest eller brugsscenarier.

Konklusion
Gennemsnittet af vurderingerne (3, 3, 5, 5, 3) giver en samlet score på 3,16.

Datamodeller

Page

Page er en abstrakt baseklasse for alle "sider" i din konsolapplikation.
Den håndterer livscyklussen for en side, herunder:

- **Aktivering og deaktivering** via Activate() og DeActivate()-metoder og tilhørende events.
- **Rendering** med kontrol over automatisk rendering (AutoRender), rydning af skærmen (ClearAtRender) og manuel eller automatisk opdatering.
- **Inputhåndtering** via event-baseret keypress (OnKeyPressed) og HandleInputTick() der styrer hvor ofte input behandles.
- **Opdateringslogik** gennem UpdateExists og HasUpdate(), som afgør om siden skal re-renders baseret på tid, eksterne signaler eller intern tilstand.
- **Layoutinformation** som vinduets bredde/højde og sideens navn.

Alle konkrete sider arver fra Page og implementerer Render(Rendere rendere) til at tegne deres eget indhold, mens Page-klassen leverer den fælles styring af opdatering, input og rendering.

LeftRightMenuPage

LeftRightMenuPage er en abstrakt sideskabelon med todelt layout: venstre interaktiv menu og højre ASCII-kunst. Den arver fra Page og leverer standard inputlogik og rendering til menubaserede skærmbilleder.

- Titel, beskrivelse og valgfri højrepanel-kunst (Title, Description, DisplayArtRight, ArtFile).
- Menuargumenter som liste af PageArgument; aktivt valg i SelectedArgument.
- Tastestyring indbygget: tal (direkte valg), piletaster (navigation), Enter (udfør). Kan slås til/fra via ListenForNumbers/Arrows/Enter.
- Event-binding i konstruktøren (OnKeyPressed += …) der håndterer valg og sætter UpdateExists ved navigation.
- Rendering opdelt i RenderLeftMenu og RenderRightMenu; venstre tegner panel, header, beskrivelse, hjælpetekst og argumenter med visuel markering. Højre centrerer ASCII-kunst.
- Hjælpefunktion TryGetDigitIndex oversætter tastkode til menuindeks for top-række og numpad.
- Render(Rendere r) opretter to sub-renderers og orkestrerer den samlede tegning.

Task

Task er en simpel, uforanderlig datamodel der repræsenterer en opgave i systemet.

Egenskaber:

- Name: Opgavens titel/identifikation.
- IsFinished: Status for om opgaven er afsluttet.

Konstruktion:

- Opretter en opgave med navn og initial status. [JsonConstructor] gør, at den kan deserialiseres direkte fra JSON.

Metoder:

- Finish(bool finished): Opdaterer om opgaven er afsluttet.
- ToString(): Returnerer en tekstrepræsentation med navn og status.

Klassen bruges til at gemme, indlæse og manipulere de opgaver brugeren opretter og kan markere som færdige.

PageArgument

PageArgument er en uforanderlig container for et menupunkt.

- Name: visningstekst for punktet.
- Action: callback der køres, når punktet vælges. Null afvises i konstruktøren.
- Brug: indgår i fx LeftRightMenuPage.Arguments, hvor UI'et viser Name og kalder Action() ved valg.

Application leve cyklys

Start og initialisering

- Applikationen starter i klassen H1BasicProgrammingFinalProject, hvor en singleton er sat op.
- I entry point kaldes singletonens Initialize() og derefter Terminate().
- Initialize() importerer først alle ASCII-arts fra resourcemappen.
- Derefter kontrollerer TaskManager, om opgavefilen findes. Hvis ikke, oprettes den. Ellers indlæses filen, og alle Task importeres.
- PageManager initialiseres og opretter de Page-instanser, der bruges i applikationen.
- MainMenuPage sættes som aktiv side, og PageManager-instansen startes.

Run-loop i PageManager

- Den primære applikationslogik kører i PageManager's run-loop.
- Hver side kan specificere ønsket zoomniveau for konsollen og om den skal være fuldskærm.
- Hvis kravene ikke er opfyldt, vises en side, der informerer brugeren om at resize og aktivere fuldskærm.
- Hvis kravene er opfyldt, fortsætter run-loop'en og kører RenderPage() på den aktive side.

Rendering og buffer

- RenderPage() udfører tjek for autoopdatering og eventuel rydning af konsollen.
- Derefter kaldes den abstrakte Render(Rendere)-metode på siden.
- Rendere er den grafiske mellemmand mellem siden og konsollen. Der kan oprettes under-renderers, men alle skriver til den samme buffer, som til sidst visualiseres i render-loop'et.

Raw Input

Raw Input

Vi tegner direkte i konsollens buffer og kan derfor ikke bruge .NET's standard-key events. I stedet læses tastaturdata via Win32 (ReadConsoleInput) på STD_INPUT_HANDLE, hvor KEY_EVENT håndteres manuelt. Det giver deterministisk kontrol, lav latenstid og egen repeat-logik.

Design

- Poll() tømmer input-bufferen og behandler kun KEY_EVENT.
- _held : HashSet<ushort> sporer nedtrykte taster. Første bKeyDown registreres som *Click*, autorepeat ignoreres. bKeyDown=false registreres som *Up*.
- Events gemmes i _events : ConcurrentQueue<KEY_EVENT> med VirtualKey, UnicodeChar, ScanCode, ControlKeyState. Konsumeres via TryDequeue().
- IsHeld(vk) returnerer om en tast er nede.

Task Manager

Formål

Central styring af opgaver: opret, fjern, markér som færdig og vis. Maksimalt 5 samtidige opgaver.

Data og struktur

- Singleton (TaskManager.Instance).
- Intern liste af DataModels.Task.
- Konstanter: MaxTasks = 5, FilePath til persistens.

Kernefunktioner

- AddTask(Task): Tilføjer opgave. Returnerer ErrorCode ved succes, for mange elementer eller ugyldigt input.
- RemoveTask(Task) / RemoveTask(string name): Fjerner efter instans eller navn.
- RetrieveTask(int index) / RetrieveTask(string name): Henter opgave efter indeks eller navn.
- Exists(string name): Tjekker om opgave findes.
- GetList(): Returnerer en kopi af alle opgaver.
- GetEnumerator(): Understøtter iteration.

Persistens

- SaveTasks(): Serialiserer til JSON og skriver til FilePath (opretter mappe ved behov).
- LoadTasks(): Læser fra fil og deserialiserer til liste.

UI-integration

- Siden TaskMenuPage bygger på LeftRightMenuPage og tilbyder menuvalg: tilføj, fjern, markér som færdig, vis alle, eller tilbage til hovedmenuen.
- Ved valg overskrives venstremenuen med relevant GUI: input til ny opgave, liste med klik-for-slet/markér, eller fuld oversigt.

Ping Pong

Formål og placering

Under hovedmenuen findes et ekstra menupunkt, der åbner PingPongPage. Siden leverer et simpelt to-spiller spil direkte i konsollen. Fokus er responsiv input, stabil fysik og minimal rendering. Funktionaliteten er isoleret på siden og påvirker ikke den øvrige applikation.

Styring og inputkilder

Spiller 1 styres med W og S. Spiller 2 styres med piletasterne op og ned. Tastatur er altid aktivt som fallback. Derudover kan siden lytte på en Arduino via USB på COM5 ved 9600 baud. Arduino sender linjer i formatet p1,p2\r\n, som fortolkes til P1Joystick og P2Joystick. Begge inputkilder sammenlæses i hvert input-tick, så joystick og tastatur kan bruges samtidigt uden konflikt.

Livscyklus

Ved aktivering (Activated) starter en baggrundsopgave, der åbner serielporten med DtrEnable og RtsEnable, venter på enheden og rydder indbufferen for opstartsstøj. Derefter kaldes StartGame(), som læser konsolstørrelse, skjuler cursor, rydder skærm, tegner ramme og paddles, viser score, starter RawInput.Poll og initialiserer boldens position og retning. Variabler for tidstagning og grænser sættes, og hovedløkken startes.

Game-loop og fysik

RunGameLoop() kører kontinuerligt. Paddles clamps til gyldigt område. HandleInputTick sampler tastetryk og joystick med et bevægelsesinterval, så bevægelse sker med kontrolleret kadence. Fysikken bruger fast tidsstep (1/240 s) med akkumulator og frame-clamp for stabilitet. Kollisionsdetektion håndterer reflekser mod top/bund og mod paddles. Træfpunktet på paddlen justerer den vertikale komponent, hvorefter hastighed vektor-normaliseres. Når bolden passerer venstre eller højre mål, opdateres scoren, bolden resettes til midten, og der serves i modsat retning.

Rendering

Rendering sker med direkte Console-skrivning. Rammen tegnes med #, paddles med |, og bolden med O. Der vises en kort trail for bevægelsesfornemmelse, og scorelinjen centreres øverst.
Render(Rendere) er bevidst tom her, da siden selv styrer tegning og timing inde i sin egen løkke for at minimere overhead og sikre jævn opdatering. Dette kan dog i fremtiden bliver opdateret til at bruge vores nye Rendere klasse.

Robusthed

Serielkommunikation er indkapslet i try/catch, så fravær eller frakobling af Arduino ikke stopper spillet. Ved fejl fortsætter spillet med tastaturstyring alene. Input behandles kun på “first-down” events for at undgå auto-repeat, og tidsbaseret gating forhindrer for hurtig gentagelse. Scoreopdatering og skærmtegn fjernes/tegnes selektivt for at reducere flicker.