Fremgangsmåde

Navn	Stefan Andrei Tumurug
Projektopstart	Projektet blev startet med oprettelse af et nyt GitHub repository via GitHub Desktop. Repositoryet blev initialiseret med en README, som grundlag for løbende dokumentation af projektet. Herefter blev projektet åbnet i Visual Studio i samme mappe som GitHub-repositoryet. Selve projektet blev oprettet som en Blank Solution i C# frem for et direkte Console App-projekt. Dette valg blev truffet for at kunne styre arkitekturen fra begyndelsen og sikre en klar adskillelse mellem spil-logik, brugerinterface og tests. Denne tilgang understøtter bedre struktur, testbarhed og fremtidig udvidelse (fx GUI). Projektopstarten og de første arkitekturvalg blev registreret i tidsregistreringsdokumentet.
Versionsstyring GitHub	GitHub anvendes til versionsstyring via GitHub Desktop. Alle ændringer committes løbende med beskrivende commit-beskeder. Dette gør det muligt at følge udviklingen trin for trin og giver mulighed for rollback, hvis der opstår fejl, som ikke har simple løsninger. Der blev desuden oprettet en .gitignore-fil for at udelukke Visual Studio-specifikke filer og build-artifacts såsom .vs, bin, obj og TestResults. Dette sikrer et rent repository, hvor kun kildekode og relevante projektfiler versionsstyres.
Solution- og Projektstruktur	Solutionen er opdelt i tre separate projekter: Blackjack.Core: Indeholder al spil-logik og domænemodeller såsom Card, Deck, Hand og GameEngine. Dette projekt har ingen afhængighed til Console eller UI og fungerer som projektets kerne.

	Blackjack.Cli **Console-applikationen, som håndterer brugerinput og visning af spillets tilstand. Dette projekt refererer til Blackjack.Core og fungerer som eksekverbar indgang til programmet.** **Blackjack.Tests** **MSTest-projekt til unit tests. Dette projekt refererer ligeledes til Blackjack.Core og anvendes til at teste kernefunktionalitet såsom håndværdi, dealer-logik og vinderberegning.** **Referencer er sat således:** **Blackjack.Cli → Blackjack.Core** **Blackjack.Tests → Blackjack.Core** **Blackjack.Core har ingen afhængigheder til de øvrige projekter.** **Denne struktur sikrer tydelig separation af ansvar og gør det muligt at teste spil-logikken uafhængigt af UI.**
Arkitektur og mappestruktur	**For at understøtte god kodepraksis er projekterne yderligere opdelt i mapper:** **Blackjack.Core:** **Domain (Card, Deck, Hand)** **Game (GameEngine og spilflow)** **Abstractions (interfaces, fx til randomisering)** **Blackjack.Cli:** **Ui (console rendering og input)** **Blackjack.Tests:** **Domain (tests af domænemodeller)** **Game (tests af game logic)**

	Denne struktur følger princippet om Single Responsibility og gør koden mere overskuelig og testbar.
Klargøring af kørbart projekt	Da Blackjack.Core er et Class Library, kan det ikke køres direkte. Derfor blev Blackjack.Cli sat som startup project i Visual Studio. En testkørsel blev foretaget, hvor standard "Hello World" blev vist i konsollen, hvilket bekræftede at solution, projektreferencer og startup-konfiguration fungerer korrekt. Projektet er herefter klar til implementering af domænemodellen og den videre spil-logik.
Implementering af domænemodeller	Domænemodellen blev implementeret i Blackjack.Core og omfatter Card, Deck og Hand. Card repræsenterer et kort med kulør og rang samt standard Blackjack-værdi. Deck opretter et standard 52-korts deck, blander kortene med Fisher-Yates algoritmen og understøtter træk af kort. Deck kan initialiseres med en seed for at gøre kort-rækkefølgen deterministisk i tests. Hand repræsenterer en hånd og beregner håndværdi med korrekt es-logik, hvor es først tælles som 11 og nedjusteres til 1 ved behov for at undgå bust.
Opsætning af unit tests og test coverage	Afsnittet om test er opdelt i separate underafsnit for unit tests og test coverage for tydeligt at adskille validering af funktionalitet fra dokumentation af testdækning. Unit tests anvendes til at verificere korrekt adfærd i spillets kernefunktioner, mens test coverage anvendes til at dokumentere, hvor stor en del af koden der faktisk er dækket af automatiske tests. Denne opdeling gør det lettere løbende at udvide tests i takt med at nye dele af spillet implementeres, samtidig med at testdækningen kan opdateres og dokumenteres uafhængigt. Strukturen understøtter en iterativ udviklingsproces, hvor tests og coverage forbedres løbende frem for at blive tilføjet samlet til sidst.
Unit tests	For at validere den mest kritiske regel tidligt blev der oprettet unit tests i Blackjack.Tests under Domain.

Domænemodel	**HandValueTests tester håndværdi-beregning, korrekt håndtering af flere esser samt bust-logik.** **Testene fungerer som sikkerhedsnet ved videre udvikling, så ændringer i game engine ikke bryder grundreglerne.**
Test coverage rapport	**For at opfylde kravet om test coverage blev Coverlet collector anvendt til at generere coverage-data.** **Coverage-rapporten visualiseres i HTML-format via ReportGenerator, så testdækning kan gennemgås pr. klasse og dokumenteres som en del af projektet.** **Der blev desuden opsat et script, som kan generere en opdateret coverage-rapport automatisk.**
Implementering og test af game engine	**Efter domænemodellen var valideret med unit tests blev GameEngine implementeret til at styre selve spilflowet.** **Dette omfatter startdeal, spillerens hit-funktion, dealerens automatiske træk indtil mindst 17 samt afgørelse af rundens resultat (win, lose eller push).** **For at gøre game engine testbar blev et IDeck-interface introduceret, så der i tests kan anvendes et kontrolleret FakeDeck i stedet for et tilfældigt deck.** **Dette gør det muligt at teste specifikke scenarier deterministisk, såsom dealerens trækregel og forskellige udfald af en runde.** **Der blev herefter oprettet unit tests for game engine, som verificerer at dealeren trækker korrekt til mindst 17, at spilleren taber ved bust, samt at samme håndværdi resulterer i uafgjort (push).** **Disse tests sikrer, at centrale Blackjack-regler fungerer korrekt, før videre udvikling af CLI og brugerinteraktion.** **Game engine tests blev inkluderet i coverage-rapporten for at dokumentere testdækning af spillets kernefunktionalitet.** **Denne tilgang sikrer, at spillets forretningslogik er fuldt valideret før UI-implementering, hvilket reducerer risikoen for regressions.**

CLI UI **Grundstruktur og Helper-Metoder**	**Som forberedelse til CLI-interfacet blev der oprettet en UI-struktur i Blackjack.Cli, hvor input og rendering holdes adskilt fra spil-logikken. ConsoleInput-klassen håndterer robust inputvalidering (fx menuvalg), så ugyldige inputs ikke kan crashe programmet.** **ConsoleRenderer-klassen samler al udskrift til konsollen, herunder visning af spillerens og dealerens hænder samt simple status- og resultatbeskeder.** **Denne opdeling gør Program.cs til et tyndt kontrol-flow, forbedrer læsbarheden og gør UI-laget lettere at udvide senere.**
CLI UI **Implementering af CLI game loop**	**Efter opsætning af UI-helper-klasserne blev selve spil-loopet implementeret i Program.cs. CLI fungerer som composition root, hvor et konkret Deck oprettes og injiceres i GameEngine.** **Program.cs indeholder kun kontrol-flow, mens inputvalidering og konsolvisning håndteres i ConsoleInput og ConsoleRenderer.** **CLI'en understøtter nu en fuld Blackjack-runde med startdeal, spillerens hit/stand valg, dealerens automatiske tur samt visning af rundens resultat.** **Dealerens skjulte kort vises først efter spillerens tur er afsluttet. Efter hver runde får brugeren mulighed for at starte en ny runde eller afslutte programmet.** **Efter implementeringen blev hele spilflowet testet manuelt via konsollen, og alle eksisterende unit tests blev kørt igen for at sikre, at CLI-integrationen ikke introducerede regressions i spillets kernefunktionalitet.**
README opdatering	**Som afsluttende trin blev der udarbejdet en README.md i repositoryets rod, som beskriver projektets formål, arkitektur, mappestruktur, implementerede features samt hvordan applikationen og tests køres.**

	README'en fungerer som central teknisk dokumentation og giver et hurtigt overblik over løsningen. **Dokumentationen inkluderer en beskrivelse af projektets opdeling i Core, CLI og Tests, de centrale Blackjack-regler der er implementeret, samt anvendelsen af unit tests og test coverage.** **Dette sikrer, at projektet kan forstås og afvikles uden yderligere forklaring.**
