DANMARKS TEKNISKE UNIVERSITET

02322 MACHINE ORIENTED PROGRAMMING

GRUPPE 31

Projektopgave 2

Forfattere
Mark NIELSEN
Lucas SCHOUBYE
Bertram KJÆR

Kursusansvarlig
Edward ALEXANDRU TODIRICA

Mark Nielsen
s204434

Lucas Schoubye
s215801

Bertram Kjær
s215775

April 2022

Indhold

1 Introduktion

I dette projekt fik vi givet opgaven om at udvikle spillet Yukon i C ud fra en opgavebeskrivelse. Der blev lagt fokus på at implementere en Linked Liste datastruktur samt indlæsning af input filer, I denne rapport vil vi dokumentere vores analyse, design, implementation og testning af programmet. Til sidst vil vi konkludere hvor vidt programmet lever op til krav fra opgavebeskrivelsen.

2 Spille regler

1. Hele dækket af spillekort deles ud i 7 kolonner af henholdsvis 1, 6, 7, 8, 9, 10 og 11 kort.
2. 21 af kortene har billedsiden vendt ned af og resten er vist.
3. Kort skal lægges i de fire foundations med én kulør i hver foundation og kort fra es-konge.
4. Kort kan flyttes mellem kolonner hvis det valgte kort har en værdi der er én mindre end det kort det lægges på og er en anden kulør.
5. Alle vendte kort kan flyttes, der kan dog kun flyttes til det nederste kort i en kolonne eller foundation.
6. Hvis der ikke er nogle vendte kort under et ikke vendt kort vendes dette kort.
7. Når alle kort er flyttet til foundations har spilleren vundet.
8. Et kortdæk har 52 kort

3 Analyse

3.1 Krav specifikation

Kravene er opstillet ud fra den stillede opgave. Vi havde i projektet stort fokus på at fuldføre de krav, der bragte os tættest på det ønskede produkt. Vi har derfor prioriteret vores krav efter MoSCoW principperne; "Must have", "Should have", "Could have". Vurderingen for de enkelte krav er baseret på kriteriet om at spillet bliver spilbart for brugeren.

3.1.1 Funktionelle krav

ID	Funktionelle Krav
Must have	
M01	Hver kort skal vises med to Characters, det første Character er værdien af kortet og det andet Character er kuløren af kortet
M02	Kort skal kunne flyttes fra kolonne til kolonne eller foundation
M03	Kort skal kun kunne flyttes, hvis trækket er lovligt
M04	Programmet skal understøtte «Game Moves>"i form af kommandoerne «from>"og «to>"
M05	Efter kommandoen "p"skal tidligere kommandoer ikke bruges længere
M06	Programmet skal kunne understøtte kommandoen "LD"
M07	Der skal være 7 kolonner og 4 foundations i programmet

ID	Funktionelle Krav
Should have	
S08	Hvis kortet ikke er synligt skal det vises med []
S09	Kolonner skal vises med C1-C7
S10	Foundations skal vises med F1-F4
S11	I bunden af terminalen skal 3 linjer vises med "Last command", "Message"og "Input >"
S12	Hvis intet filnavn er givet til LD kommandoen skal et ikke blandet kortspil indlæses
S13	Hvis filnavnet ikke findes skal brugeren få en fejlbesked
S14	Hvis filnavnet findes skal programmet tjekke om dækket har den rigtige mængde kort
S15	Programmet skal kunne understøtte kommandoen "SW"
S16	Hvis programmet ikke har et dæk skal det give en fejlbesked
S17	Programmet skal kunne understøtte kommandoen "SI<split>"som blander dækket på en indfletet måde
S18	Programmet skal kunne understøtte kommandoen "SR"som blander dækket tilfældigt
S19	Programmet skal kunne understøtte kommandoen "SD<filename>"
S20	Hvis intet filnavn er givet skal "SD<filename>"skal standard navnet “cards.txt” bruges
S21	Programmet skal kunne understøtte kommandoen "QQ"
S22	Programmet skal kunne understøtte kommandoen "P"
S23	Programmet skal kunne understøtte kommandoen "Q"
S24	Efter kommandoen "Q"skal programmet gemme det dæk der blev brugt
S25	Hvis kommandoen "P"bruges efter "Q"skal spillet starte med det samme dæk
S26	kommandoen «from>"skal være i formatet «from>=<column>:<card>"
S27	Hvis en kommando er gyldig skal en besked "OK"printes
S28	Hvis en kommando er ugyldig skal en fejlbesked printes

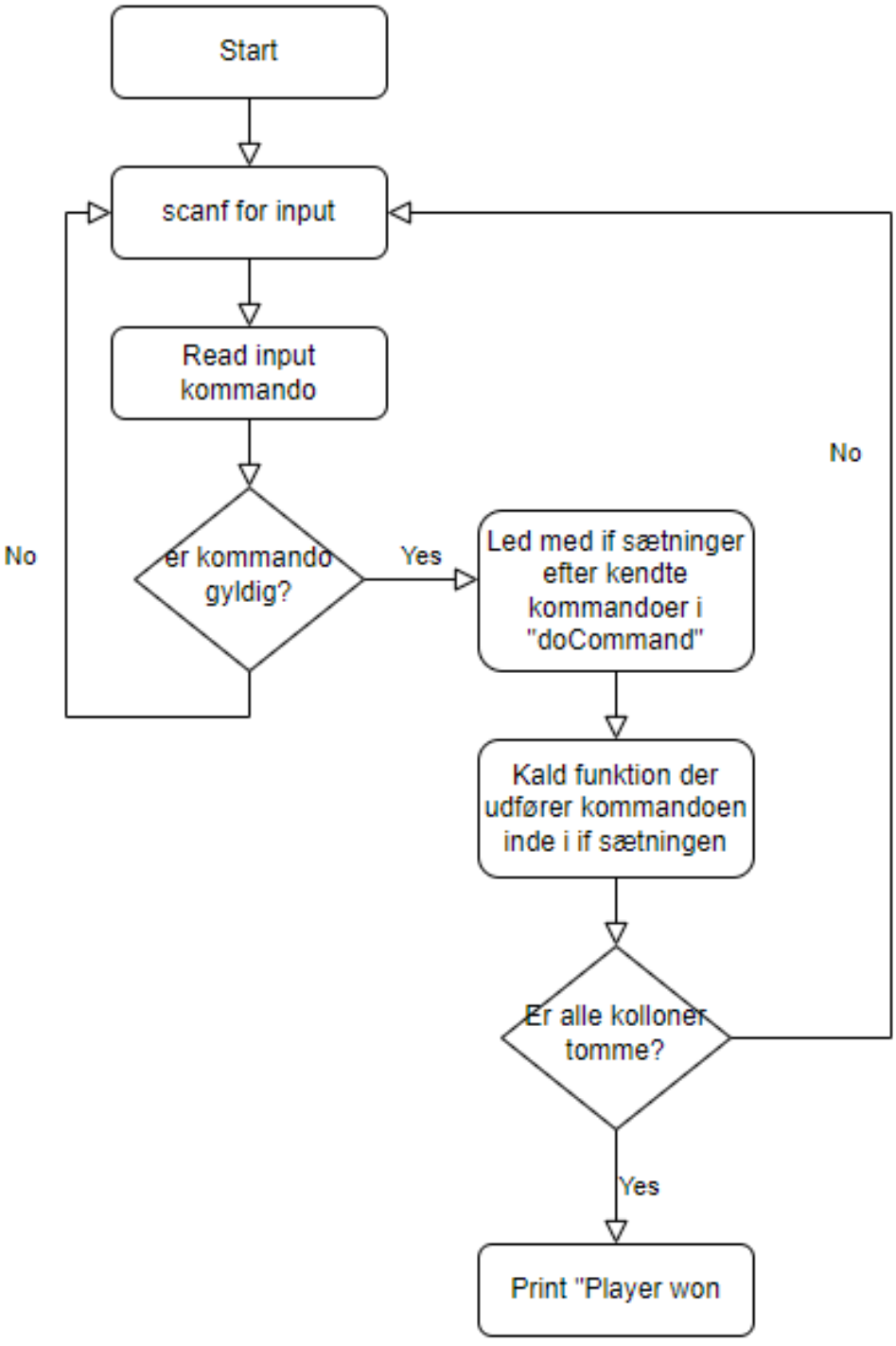
ID	Funktionelle Krav
Could have	
C29	Programmet skal tjekke om input er gyldigt
C30	Programmet skal kunne understøtte kommandoen "U"
C31	Programmet skal kunne understøtte kommandoen "R"
C32	Kommandoen "R"kan kun benyttes hvis kommandoen "U"er brugt inden
C33	Programmet skal kunne understøtte kommandoen "S <filename>"
C34	Programmet skal kunne understøtte kommandoen "L <filename>"
C35	Programmet skal have en GUI

3.1.2 Kommandoer

De forskellige kommandoer der omtales i krav specifikationen uddybes her:
LD<filename> indlæser et sorteret dæk kort ud fra filnavnet
SW viser alle kort i terminalen i den rækkefølge de ligger i dækket
SI<split> blander dækket af kort på en indflettet måde hvor parameteren split er mængden af kort i den første bunke af blandingen som til sidst bliver vores nye dæk
SR blander dækket af kort tilfældigt som bliver vores nye dæk
SD<filename> gemmer det nuværende dæk af kort i en fil af samme navn som det parameter givet
QQ lukker programmet
P starter spillet med det nuværende dæk af kort
Q lukker det nuværende spil og tager brugeren tilbage til "startup phase"men beholder det dæk af kort der blev brugt i spillet
<Game Moves> flytter kort i spillet ud fra det angivet i parameteren

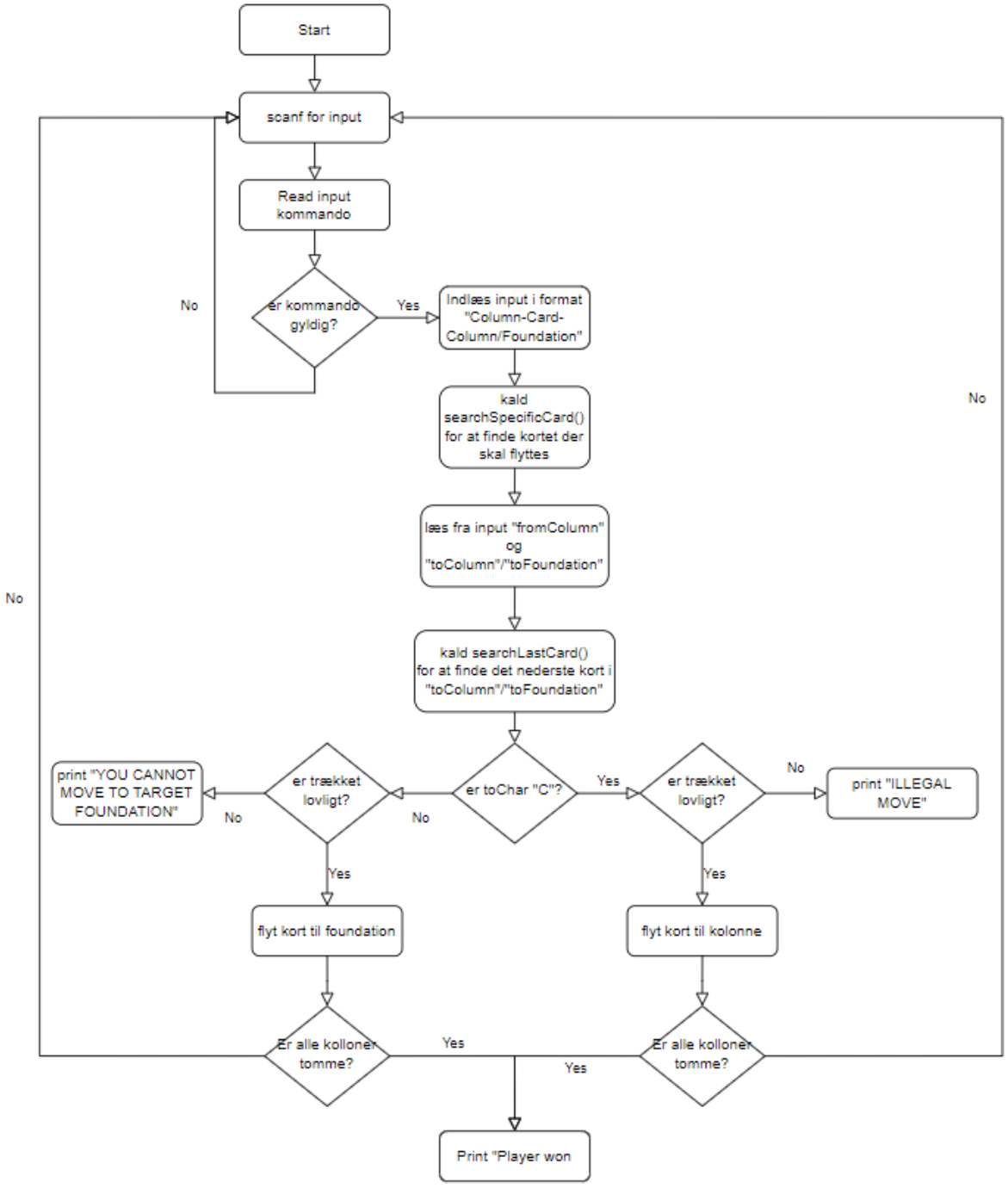
4 Design

Inden der blev implementeret er der udarbejdet flowdiagrammer over funktionaliteten af spillet yukon, der efterfølgende er implementeret ud fra. Der er valgt at lave flowdiagrammer over to vigtige enkelte funktioner samt et generelt over spil logikken. Disse ses her:



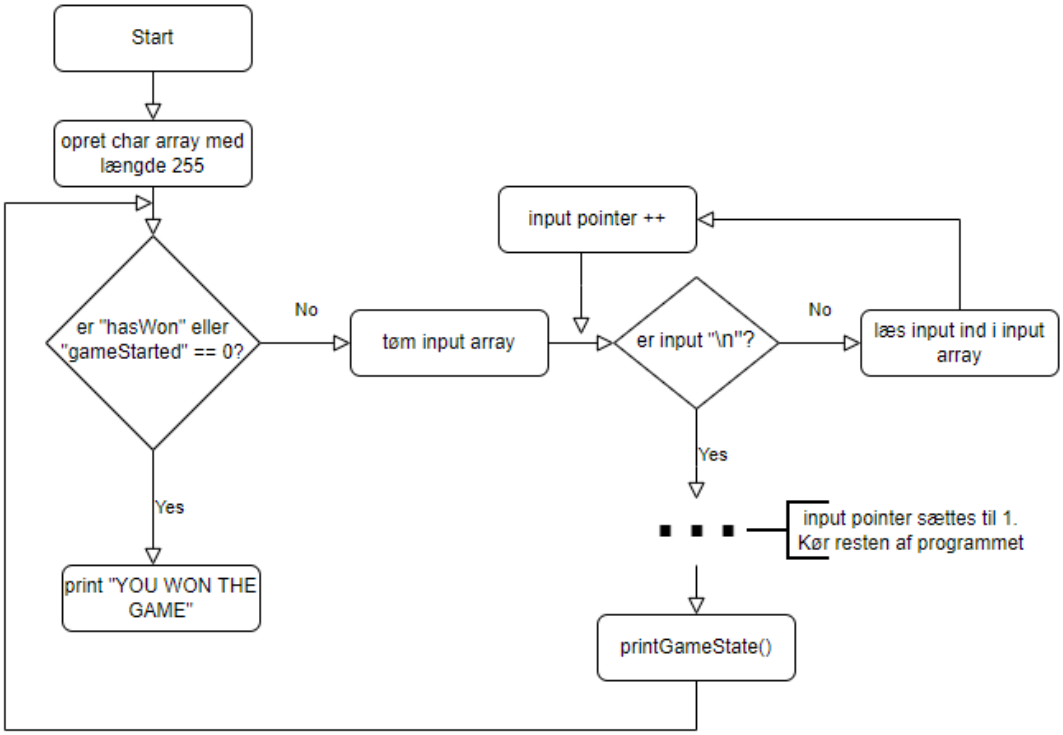
Figur 4: Game logik flowdiagram

I dette flowdiagram er det antaget at input er til at flytte et kort derfor er der en varians fra det generelle flowdiagram efter "er kommando gyldig?". Samt er der antaget at toChar er F hvis ikke C.



Figur 5: moveCard flowdiagram

I dette flowdiagram viser "..."irrelevant kode for denne logik der ligger inden næste step, med en annotation om hvad der sker under det forkortede område.



Figur 6: Input flowdiagram

5 Implementering

5.1 Card Structs & Linked List

I dette afsnit vil vi dokumentere hvordan card er blevet implementeret og hvordan de understøtter linked lister. Card Structuren indeholder 2 chars, 1 integer og 2 Card pointers. De 2 char står for kortets værdi og kortets kulør. Værdien bliver konverteret til en integer under spillogikken, men der bliver brugt en char til visning og input af 2 cifrede tal som skrevet i opgavebeskrivelsen. De 2 pointers bliver brugt til at holde styr hvilke kort der ligger i hvilken rækkefølge. 'next'-pointeren peger imod kortet nedenunder det givne kort, altså tættere på toppen af kolonnen. 'prev'-pointered peger i modsatte retning. Hvis kortet er det sidste i kolonnen er dens pointer null. Ved hjælp af disse pointers er det muligt at lave en linked list hvorpå kort kan rykkes fra en kolonne til en anden, hvorpå alle kort der er lagt ovenpå følger med. Det gør det også nemt at implementere print af kolonner.

5.2 Input

I dette afsnit vil vi dokumentere hvordan vi modtager input i vores kommando-linje og hvordan de bliver brugt senere i programmet. Vi modtager input en char af gangen ved hjælp af printf() og en while loop. Alle characters bliver indsat i et char array med længde 255 (inputLine). Der er også en integer som tæller op for hver karakter, dette gøres så man kender til det sidste index i arrayet der ikke er NULL (inputPointer). Der bliver tjekket efter om spilleren inputter et enter, hvorpå while-loopet stopper. Nu kan vi læse vores public char-array med vores input i resten af programmet og vi kan læse fra højre side af arrayet. Før while-loopet bliver der kørt en loop som sætter alle index lig NULL så vi er klar til at næste input. Som vist i figur 6.

5.3 Generelle Kommandoer

- **LD**: Load-kommandoen er implementeret vha. checkFileInput-metoden som tjekker en fil's navn og parser den til doCommand-metoden, hvor vores 'LD' kommando er placeret. For at læse filen skal stien til filen kunne indlæses, hvortil der er manipuleret med filnavnets char array, således arrayets første 3 pladser indeholder '..\', da den skal finde frem til filens sti, hvilket '..\' i starten af filnavnet sørger for. Når filens kort skal benyttes i vores deck benyttes metoden fopen som sættes til variablen FILE* fp. Herefter er der et while loop som kun kører så længe filen ikke er endt, hvilket metoden feof med FILE * fp som parameter sørger for at tjekke. Herefter læses filens characters to af gangen, hvor henholdsvis rang og kulør læses ind i et char array og tilføjes til decket, som metoden fgets sørger for, der læses en linje af gangen fra filen. Hvis filen ikke indeholder noget printes beskeden "File not found.". Herefter benyttes metoden fclose til at sørge for lukning af filen.

- **SW**: SW-kommandoen står for at vise spillets nuværende tilstand og er blevet implementeret ved et metodekald til printDeck metoden som itererer 52 gange svarende til antallet af kort i spillet, og kalder metoden printAnyCard i sig som står for at printe den korrekte kulør og rang for hvert kort. Eftersom et kort indeholder to characters har vi givet de tocifret rang (10, 11, 12 og 13) bogstaver i stedet for cifre, så de har henholdsvis T for 10, J for knægt, Q for dronning og K for konge når de bliver printet til spillet, hvilket er grunden til vi benytter en switch på et korts rang efter rang 9.

- **SR**: SR-kommandoen blander kortene, hvilket sker gennem et metodekald til mixCards-metoden, som er designet med det specifikke formål at blande hele bunken. Rand() metoden bruges her til at give indekstallene en tilfældig værdi op til 52, hvor et tilfældigt vend mellem to tal sker bagefter.

- **SI**: SI-kommandoen blander kortene gennem mixCardInterleaved() metoden, på den specifikke måde at dele bunken op i to og så "flette"dem sammen ved skiftevis at tage øverste kort fra hver bunke og lægge i en tredje bunke der bliver den nye bunke. Dette gøres ved at få en tilfældig integer topDeck fra rand() metoden der bestemmer mængden af kort i første bunke (shuffle1), herefter fyldes de to array shuffle1 og shuffle2 med for loops. Dernæst fyldes den tredje bunke i et for loop af længden topDeck ved at skifte indeks og så opdatere den tredje bunke til at have samme værdi som det indeks i shuffle bunkerne. Til

sidst hvis topDeck ikke er lig med 26 opdateres de resterende indeks af den tredje bunke til at være lig de resterende indeks fra det shuffle deck der har overskydende kort.

5.4 Spil Kommandoer

Vi har implementeret spil-kommandoer ved hjælp af vores input array. Vi indlæser input på dette format:

- C6-1H-F0

Vi inputter disse seks char ind i vores metode. Vi læser først hvilken column der skal søges kort fra. Herefter søger vi på kortet ved hjælp af vores Linked-List. Efter dette vil vi rykke kortet. Der bliver tjekket på det sidste kort i linked listen indikeret på de sidste 2 input. Hvis dette er et lovligt træk placere vi kortet i den korrekte linked-liste ved at opdatere vores pointers.

6 Tests

I dette projekt er der lavet test over vigtige aspekter af spillet, hertil er der valgt at teste kommandoerne SR, SW, P samt at lave et ulovligt træk.

Test case	Expected result	Actual result	Remarks
01	Dækket er ikke samme rækkefølge	Dækket er ikke samme rækkefølge	der er et kort som gentages 6S

Her ses testen i terminalen, samt koden for at blande.

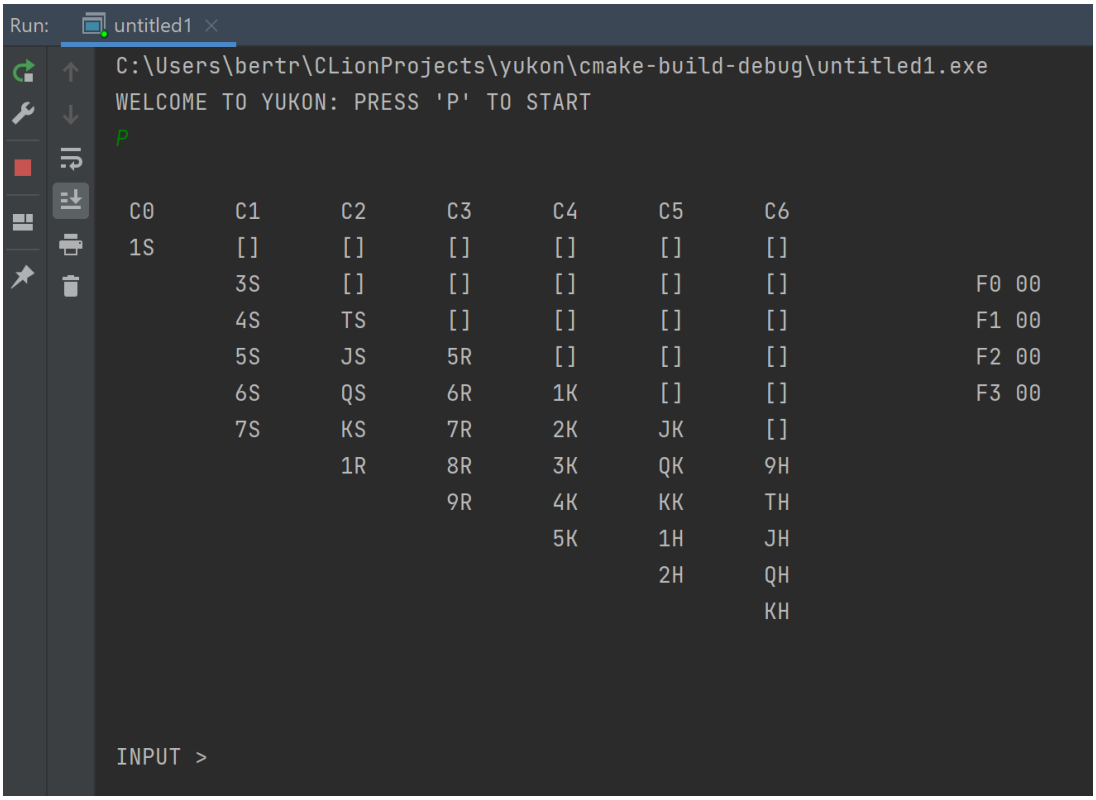
Figur 7: SR Test

Test case	Expected result	Actual result	Remarks
02	Dækket vises i terminalen	Dækket blev vist i terminalen	Dækket er vist på forkert format

Som der ses i figur 7 er SW kommandoen fungerende til at vise det indlæste dæk

Test case	Expected result	Actual result	Remarks
03	Spillet startes i terminalen	Spillet blev vist i terminalen	

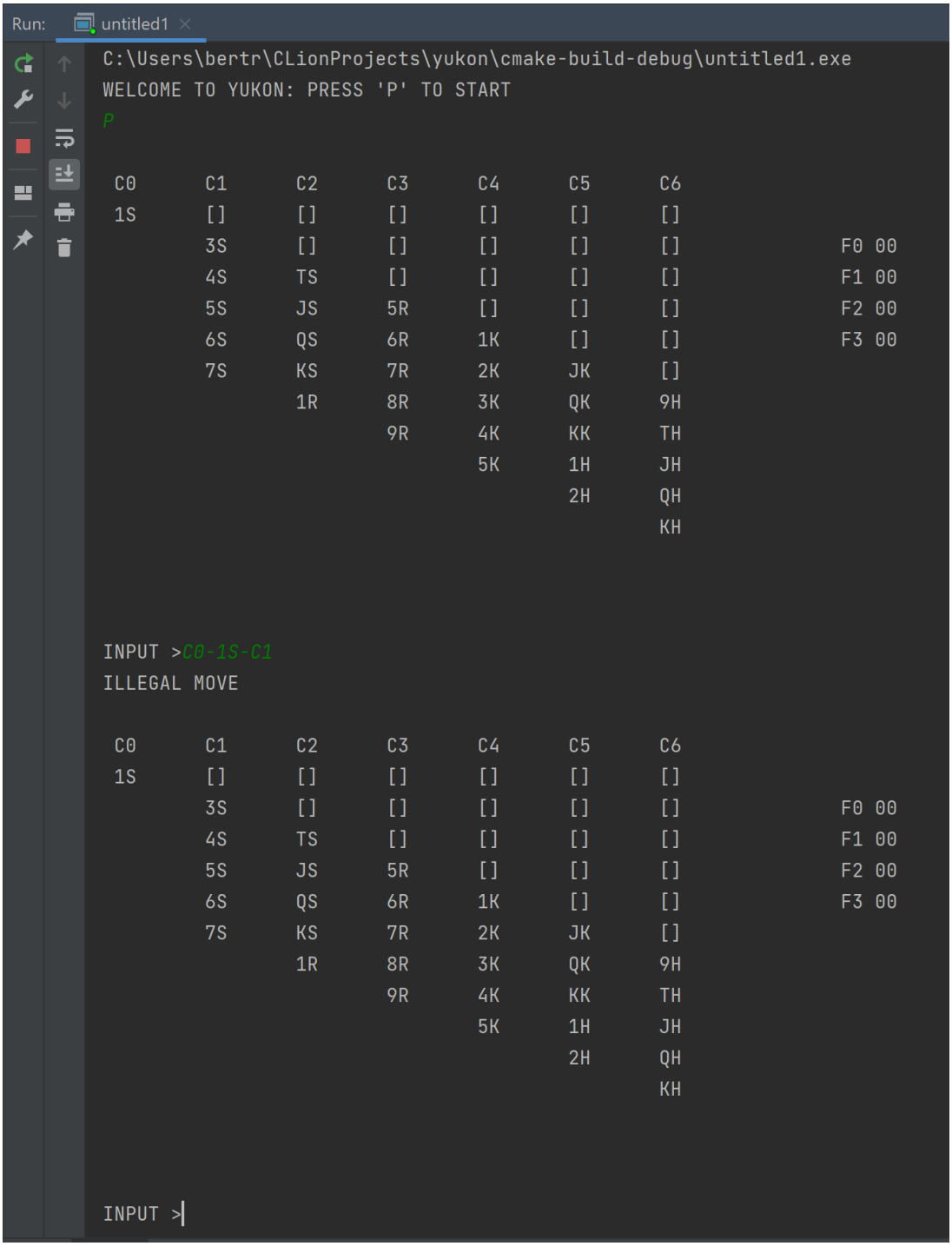
Her ses terminalen med spillet:



Figur 8: P Test

Test case	Expected result	Actual result	Remarks
04	Ulovligt træk ikke foretaget + fejlbesked	trækket blev ikke foretaget samt en fejlbesked givet	

Her ses terminalen med spillet:



Figur 9: Move Test

7 Konklusion

Vi kan konkludere at alle Must-have krav er opfyldt. Krav M05 er opfyldt, dog med visse front-end mangler. Visse should-have krav mangler at blive opfyldt. S09-S12, S14, S16, S19, S23 og S27 er delvist løste. Ingen could-have krav er opfyldt. Vi konkludere at vi implementerede en funktionel linked liste data struktur. Vi konkludere at en Struct til selve Linked Listen kunne være fordelagtig og mere dynamisk i fremtidige projekter. Vi konkludere at gruppen brugte flow-charts og diagrammer til at implementere krav effektivt. Vi konkludere at der er blevet produceret et funktionelt program som simulere spillet Yukon til en acceptabel standard.

8 Gruppe indsats

I dette afsnit vil vi gennemgå aktiv deltagelse af diverse gruppemedlemmer. Der vil blive gennemgået generelle gruppeopgaver samt beskrivelse af mængden af tid gruppemedlemmet kunne tildele deres arbejdsopgaver.

8.1 Mark s204434

Mark brugte størstedelen af sine timer på programmering. Han implementerede indlæsningen af filer, Linked Lister og arbejdede i kollaboration på diverse metoder og spil logik. Mark har også skrevet rapport, hovedsageligt implementationsafsnit. Han har et over gennemsnitligt antal timer på projektet, dvs. arbejdede overtid visse dage.

8.2 Bertram s215775

Bertram brugte størstedelen af sine timer på rapportskrivning. Han implementerede split-shuffle metoden. Bertram skrev blandt andet Analyse, Design, Konklusion og Test. Visse dage havde Bertram ikke mulighed for at være tilstedeværende fuldt ud.

8.3 Lucas s215801

Lucas brugte størstedelen af sine timer på programmering. Han implementerede Linked Lister, formattering og spil logik. Han arbejdede i kollaboration på diverse metoder. Rapportmæssigt har han været inde over implementationsafsnittet. Han arbejdede ikke overtid.