NTI Gymnasiet - Nacka Strand
Teknikprogrammet-Teknikvetenskap
Gymnasiearbete 100p
HT 2023 - VT 2024

Framtagande av en potentiell kortleksblandare

Proof of Concept

Namn: Eduards Abisevs
E-post: eduards.abisevs@elev.ga.ntig.se
Namn: Leo Altebro
E-post: leo.altebro@elev.ga.ntig.se
Handledare: Elias
E-post: @
Examinator: <WHO Will it be?>

Typsatt med LaTeX.
Kompilerad 9 februari 2024.

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Innehåll

Ordlista

p-värde Den uppmäta sannolikheten att nollhypotesen är sann

ApEn Approximate entropy

bibliotek Samling av färdigt byggt kod för programmerings språk Python

crate Samling av färdigt byggt kod för programmerings språk Rust

df Frihetsgrader (Degrees of freedom)

GSR Gilberts-Shanons-Reeds

matris Data i två dimensioner som en tabell

MP Medelvärde av Position

ns Nanosekunder

PK Position av Kort

PRNG Pseudoslumptalsgenerator

SD Standardavvikelse (eng. Standard deviation)

Signifikansnivå Sannolikheten att, vid hypotesprövning, förkasta nollhypotesen trots att den är sann

SOC Shuffle-O-MatiC

STDMean Standardavviklse-och medelvärde (eng. Standard deviation and Mean)

TRNG Äkta slumptalsgenerator

1 Inledning

1.1 Introduktion till ämnet

Spelbranschen är en industri som omsätter miljardbelopp och precis som alla industrier utvecklas den med resten av världen. Industrier över hela världen har som mål att hitta nya sätt att effektivisera och sänka kostnad på det arbete som utförs för att bedriva vinst, och spelbranschen har följt denna långvariga trend med till exempel online casinon. Inom spelbranschen är kortspel vanligt förekommande, att försöka automatisera dem är därför ett logiskt steg att ta, men för en stor industri vill man vara extra säker på att allt utförs på bästa sätt. Alla sätt att blanda kort är nämligen inte lika bra, vilken sorteringsmetod som används kan ha påverkan på resultatet. Teknologins inflytande inom spelbranschen ökar kraftigt, de största delarna av branschen bedrivs mer och mer av maskiner och algoritmer som får stor potentiellt inflytande på spelresultat, därför är det bäst att börja tänka på möjliga problem så snart som möjligt. Redan nu finns det blandningsmaskiner för kortlekar som vi inte vet någonting om; varken hur de funkar eller hur bra de är. Allt som vi vet är de är certifierade av tredjepartsföretag. Faktumet att de kan kosta upp till 100 000 kr betyder att inte vem som helst kan ha tillgång till en kortblandningsmaskin.

1.2 Syfte

Syftet med denna undersökning är att ta fram den hypotetiskt bästa kortblandaren utifrån två faktorer: 1) hur slumpmässigt den algoritm som den byggs efter kan blanda kort; 2) till vilken grad den potentiella maskinen byggd efter algoritmen skulle fungera i verkligheten. Med resultaten förväntas variationen i slumpmässighet för olika blandningsmetoder kunna visas upp samt kunna använda de resultaten för att komma fram till den hypotetiskt definitiva maskinen, något som är viktigt då spelbranschen handlar mycket om chans. Det är därför viktigt att se till att allt funkar på bästa sätt. I den här vetenskapliga rapporten kommer en jämförelse av hur effektivt framtagna blandningsmetoder kan fungera i en hypotetisk blandningsmaskin att utföras. Samtidigt som man kan utforska hur en dator kan göra slumpmässiga sekvenser på ett effektivt sätt med tanken att den ska tillämpas till en potentiell kortblandare. Detta är viktigt för att spelbranschen är en stor industri som handlar mycket om tur, att se till att de slumpmässiga resultaten är framtagna på bästa möjliga sätt är en essentiell del. Vi söker med hjälp av data svaret på frågan:

Utifrån slumpmässighet och effektivitet, vilken kortblandningsmetod är bäst för en potentiell spelkortsblandare som uppfyller följande:

- Fysiskt tillämpning: Hur pass väl den kan framställas i verkligheten
- Effektivitet: Utifrån mjukvaras och hårdvaras perspektiv
- Kortblandnings slumpmässighet

2 Teori

2.1 Slumpmässighet

Beräkningsteori är en del av matematik vars syfte är grundat i hur och om problem kan bli lösta på olika beräkningssätt. Beräkningsteori har flera olika grenar. En gren handlar om vad som går att bevisa inom matematik angående om nummer och funktioner är beräkneliga eller inte. Med beräknelig menas något som kan beräknas, det vill säga värderas, uppfattas eller förutses, när det kommer till matematik syftar det på att bestämma något via matematiska modeller eller processer, alltså att kalkylera.

Slumpmässighet är en egenskap som anger att något sker utan klart mönster. När något är helt slumpmässigt är det i princip omöjligt att förutse. Slumpmässighet inom matematik är byggd beräkningsteori och den delas upp i två beroende på om det gäller slumpmässighet av en bestämd mängd eller en oändlig mängd av objekt (Terwijn, 2016, s. 49–66).

2.2 Slumptalsgeneratorer

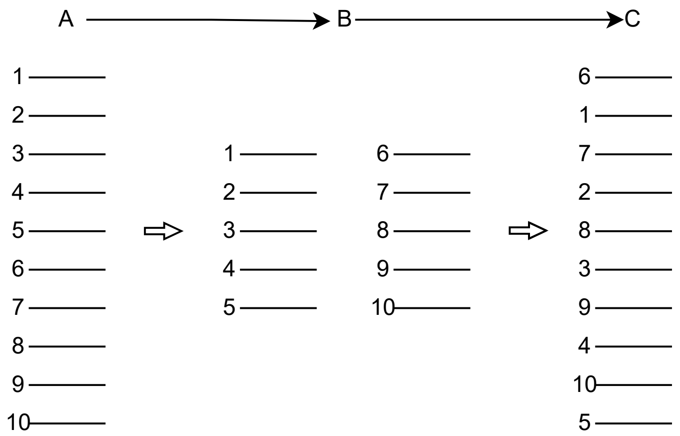
Som det nämndes i sektion 2.1 finns det olika typer av slump. Följaktligen finns det olika typer av slumptalsgeneratorer. Dem kan indelas i två typer icke-deterministiska och deterministiska så kallade *äkta slumptalsgeneratorer* (true random number generators- TRNG) respektive *pseudo slumptalsgeneratorer* (psuedo random number generators- PRNG). Icke-deterministiska processer använder sig av naturliga fenomen e.g. termisk natur, kosmisk strålning eller radioktivt sönderfall. Därför denna metod behöver en speciellt utrustining. Å andra sidan deterministiska processer är när en slumptal algoritm simulerar en slumpmässigt händelse. Men den behöver ett start data som sedan utför ett kedja av operationer. Därför deterministiska slumptalgenerator behöver ett frö (seed) att utgå ifrån. På så sätt kan exakt samma sekvens av slumpmässiga sekvenser återskapas om fröet är känd (Hannes, 2011).

Det finns olika områden när TRNG eller PRNG är lämligast att använda. Enligt *random.org* sammanfattar dem dessa till följande PRNG vid simulationer och TRNG vid casino spel (Random.org, 2024).

2.3 Bakgrund av blandings metoder

2.3.1 Riffle shuffle

Riffle Shuffle är den mest kända kortleksblandningsmetoden. Riffle shuffle har undersökts över långt tidsperiod. Den matematiska modellen utvecklades av Gilbert och Shannon 1955 och senare oberoende av Reeds 1981 (Diaconis, 2003, s. 77–79).

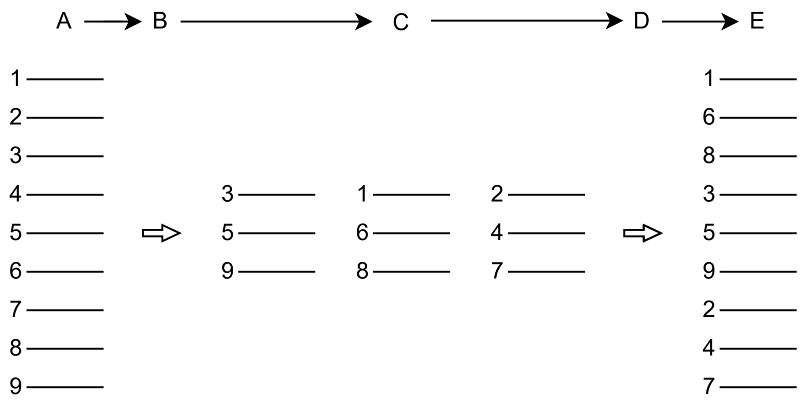


Figur 1: Steg i processen för Riffle Shuffle. Illustrationen visar de iterativa stegen från den ursprungliga högen (A), uppdelning i två högar (B), infoga dem två högarna med varandra för att bilda en ny hög. Pilar indikerar riktningen för blandningsprocessen.

2.3.2 Pile shuffle

Pile shuffle är en kortleksblandingsmetod som genomförs med fysiska kort. Processen utgår från att ett kort från kortleken läggs i en av flera olika högar. Processen försätter tills alla kort från kortleken har flyttats till en av högarna. Sedan läggs alla högar ihop i en ny kortlek. Flöde av en sådan metod visas i Figur 2.

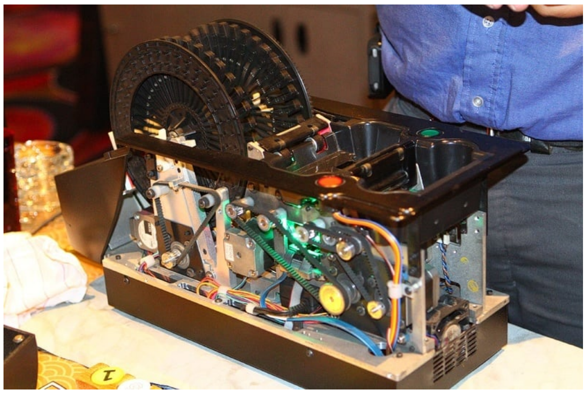
Att hitta en kortleksblandare som hade implementerat en Pile Shuffle metoden var inte enkelt uppdrag. Men det hittades en kortleksblandare som lyckades med det. Källan hittades på YouTube och var skapad av 3DprintedLife (2021). Blandingsmaskinen kallas till Shuffle-o-matic (SOC). I denna YouTube videon visas det upp hur en sådan maskin fungerar och hur kan denna 3D printas. Där meddelas det att kortmatningsmekanismen var svårt att implementera p.g.a. hur tunna spelkort är.



Figur 2: Steg i processen för Pile Shuffle. Illustrationen visar de iterativa stegen från den ursprungliga högen (A), genom uppdelning i högar (B), temporara högar (C), omarrangering av temporara högar (D), och tillbaka till en enda hög (E). Pilar indikerar riktningen för blandningsprocessen.

2.3.3 Inblick i en professionell kortlkesblandare

Information om professionella kortleksblandare är begränsat, speciellt när det gäller vilka algoritmer som dessa använder. Det har uppfattats att detta beror på säkerhetskall d.v.s. att den ska inte vara tillgänglig till publiken. Därför att på detta sättet säkerställer och minimerar utvecklarna av dessa maskiner risken att obehöriga parter hackar denna. Men under literärforskings fasen hittades det en gammalt YouTube video. Videon presenteras hur en professionell blandingsmaskin ser ut och fungerar från insidan (Yan, 2016). Såsom en bild av en likadan blandingsmaskin hittades, se Figur 3. Ett tydligt mönstren som båda maskinerna har. Utvecklarna valde att ha fullt kontroll av varenda kort som blandas. Såsom det betyder att denna blandingsmaskin skulle teoretiskt sätt använda vilket som helst algoritmn, samt genererar en riggad kortlek. Med riggad kortlek menas att ordningen är definerad i förväg. Motiveringar till denna skulle inkludera skall av vinst men detta är inte bevisat på ett eller ett annat sätt.



Figur 3: Inblick in en isärtagen kortleksblandare

2.4 Den Klassiska poker testet

Ett klassiskt poker test används att avgöra slumpmässighet i numeriska sekvenser, oftast för att testa slumptalsgeneratorer. Testet utförs genom att 3 till 5 nummer väljs ut ur en sekvens och placeras i en av sju kategorier beroende på mönstret som talen har. Mönstren är baserade på händer i poker vilket är varför testet kallas poker test (Abdel-Rehim, Ismail och Morsy, 2014). De olika mönster som letas efter i talen visas i Tabell 1. Antalet mönster i varje kategori räknas för att få en distribution, som då jämförs med en distribution som stämmer överens med sannolikheten att få de olika mönstren. Om det total antalet mönster är n och sanolikheten för en pokerhand i är p_i, därför borde antalet mönster i den kategori vara

Tabell 1: Vanliga kategorier till poker test

Pokerhand	Mönster
Femtal	AAAAA
Fyrtal	AAAAB
Kåk	AAABB
Tretal	AAABC
Tvåpar	AABBC
Par	AABCD
Ingen mönster	ABCDE

$$e_i = p_i * n$$

Där o_i är antalet mönster som borde matcha pokerhand i. För att bestäma om resultaten kan komma från en rättvis kortlek jämnförs de resultat som fåtts av att plocka ut nummer mot de resultat som fås av sannolikheten via ett chi-två-test.

2.4.1 Chi-två-test

Inom statistik används ett *goodness-of-fit* test för att mäta hur pass väl fördelningen för den data som observerats stämmer överens med fördelningen som data förväntas ha utifrån en model. Karl Pearsons chi-två-test kan användas som *goodness-of-fit* test. Det använder chitvåfördeling. I testet jämnförs ett värde χ^2 med ett kritiskt värde för att bestäma om den observerade följer den förväntade fördelningen. χ^2 beräknas enligt formeln

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

Där O_i är observerade frekvensen för en kategori av data i och E_i är förväntade frekvensen för kategorin i. Det kritiska värdet fås av antalet frihetsgrader(kategorier - parametrar) och signifikansnivån som bestämts för testet. Såsom ett viktigt aspekt i chi-två-test är att data mängderna som testas inte kan vara för små. Det rekommenderas att O_i inte är mindre en 5 (National Institute of Standards and Technology, 2023).

2.5 Programmerings verktyg

Inom ramen för denna studie är användningen av programmeringsverktyg som Python och Rust av stor betydelse. Detta sektion inkluderar beskrivnigar av alla verktyg som ska användas. Tyngden betonas på den tidigare forsknings yttersts utmärkta arbete inom datanalys verktyg som publicerats till öppen källkod för alla att använda. För att utan dessa verktyg skulle denna studie inte vara möjlig att utföra.

Python är en objektorienterad, dynamiskt typad programmerings språk med fokus på läsbarhet och enkel syntax (Van Rossum och Drake, 2009). Den har etablerat sig som ett fundamentalt språk inom datavetenskap p.g.a. sitt stora ekosystem av kraftfulla bibliotek.

NumPy utvecklat av Travis Oliphant och ett globalt team av bidragsgivare är standardbiblioteket för vektoriserad aritmetisk beräkning i Python. Dess effektiva hanterings metoder på stora datamängder och matematiska operationer gör det till första hands val för dataanalys vid vetenskaplig forskning (Harris m. fl., 2020).

SciPy, ett stort samling av öppen källkod-programvara för matematik och vetenskap. Det är resultatet av samarbete mellan hundratals forskare och bidrar till att göra komplexa matematiska beräkningar tillgängliga och effektiva (Virtanen m. fl., 2020). E.g. funktioner för chi-två-tests utförande.

Numba är ett kompilatorbibliotek som översätter Python-kod till snabb, maskinkod som körs på egen tråd. Detta projekt, som drivs av Anaconda, Inc., möjliggör betydande hastighetsförbättringar för dataintensiva implementationer förutsatt att det används fundementala datatyper och operationer (Lam, Pitrou och Seibert, 2015).

Matplotlib, skapat av John D. Hunter och nu underhållet av en stor utvecklargemenskap, är det ledande biblioteket för datavisualisering i Python. Det gör det möjligt att enkelt skapa en mängd olika grafik och plotter, vilket är kritiskt för analys och presentation av data Hunter (2007).

Rust är ett kompilerad statisk programmeringsspråk som fokuserar på minnessäkerhet, samt minneseffektivitet och parallellism. Rust är känt för

sina avancerade funktioner som ägarskapssystemet (ownership), vilket hjälper till att förhindra minnesläckor och tillåter säker minneshantering utan en skräpsamlare (garbage collector) (N. D. Matsakis och Klock II, 2014). P.g.a. dessa egenskaper har det ökats popularitet av användning av Rust i inbyggda system som ett motkandidat till programmeringsspråk C, detta undersöktes på djupet av Sharma m. fl. (2023).

Rand crate är för pseoduslumtal generering med enkelt användning via *Rng* trait. Samt säker och snabb behandling av fröet med *thread_rng*, den använder algoritm ChaCha20 (The Rand Project Developers, 2022).

Rayon crate är för att konvertera sekvensiella iterationer i Rust till parallellt och säker exekvering (N. Matsakis och Stone, 2022).

Genom att använda dessa verktyg har denna studie kunnat utföra omfattande simuleringar och dataanalyser på ett effektivt sätt. Deras tillgänglighet och prestanda har varit avgörande för studiens framgång.

3 Metod

Metoden för denna studie kan indelas i tre huvudområden: (i) Framtagande av blandningsalgoritmerna; (ii) Simulering av kortblandningsprocesser; (iii) Statistisk analys av insamlad data.

Inledningsvis kommer framtagande av blandningsalgoritmerna fokusera på utvecklingen av specifika kortblandningsmetoder. Detta inkluderar både etablerade metoder som Riffle Shuffle (2.3.1), samt nyare, innovativa tillvägagångssätt som Pile Shuffle (2.3.2). Såsom anpassning av design av en professionell kortleksblandare

(2.3.3). Vilka valts ut baserat på deras potential i en potentiell kortleksblandare. Dessa algoritmer är kritiska för bedömningen av kortblandningsmotodens slumpmässighet och effektivitet, vilket är kärnan i studiens frågeställning.

Målsättningen är att djupgånade undersöka kortblandnings fysiskt tillämplighet, dess effektivitet och slumpmässighet. Att utföra simulation valdes därmed att blandningsalgoritmerna (processmässigt) skulle vara snarlika till dess potentiella kortleksblandare. Därmed skulle det genereras mer tillförlitliga testresultat. Därför simuleringsdelen innefattar en kvantitativ metodik för att reducera den inneboendes slumpvariation i de simulerade blandningsalgoritmerna. Genoma att generera ett bestämt mängd av kortleksblandningar, där datamängder uppfyler kravet ifrån statistiska metoder. Dessa statistiska analysmetoder omfattar det klassiska pokertestet (2.4), med resulat given av chi-två-testet (2.4.1). Samt standardavvikelse- och medelvärdestest (STDMean), vilket ger ett närmare inblick i korts variation i specifika positioner i kortleken. Tillsammans dessa tester ger en tillfredsställande indikation på blandningsmetodens slumpmässighet.

Genom att välja denna metodik syftar studien till att utföra en omfattande kvantitativ analys som inte enbart bedömer algoritmernas teoretiska effektiviteten men även dess praktiska tillämplighet i en potentiell kortleksblandare. Detta tillvägagångssättet möjliggör en detaljerad utvärdering av varje algoritm, dess implementering och slutliga prestande, vilket är avgörande för att uppnå studiens mål. Mer detaljerad beskrivning av specifika algoritmerna, simuleringar av kortblandningar och statistiska utvärderingar presenteras i kommande underrubriker.

För intresserade parter finns det källkod för algoritmernas implemention, programmet som användes för simulation, samt implementionen av analysmetoderna på Github se Bilaga A.

3.1 Testmiljö

I valet av testmiljö prioriterades operativsystemets kompatabilitet av de utvecklingsverktyg som användes. Linux valdes på grund av dess robusta stöd för programmeringsmiljöer och breda stöd för mjukvaruutvecklingsverktyg. Dessutom erbjuder operativsystem Linux bättre kontroll över systemsresurer, vilket är avgörande för att uppnå följdriktiga och tillförlitliga testresulat. Därför utfördes undersökningen på en dator med Linux som operativsystem. Se Tabell 2 för testmiljös specifikationerna.

Typ	Specifikation
Processor	AMD Ryzen 5 3600 6 cores / 12 threads 3.6 GHz
RAM	15.93 GB
Hårddisk	KINGSTON SA400S3, 447 GB
Operativsystem	`Arch Linux x86_64,` `Linux kernel 6.6.9-arch1-1`

Tabell 2: Testmiljö med Linux som operativsystem.

3.2 Framtagande av blandningsalgoritmerna

I detta avsnitt presenteras processen av framtagande och detaljerna kring varje algoritms kod och dess möjliga implementering i en potentiell kortleksblandare.

Algoritmerna implementerades i programmerings språk Rust version 1.73.0. Rust valdes p.g.a dess höga abstraktioner med närmare tillgång till systemresurer (för detaljer se avsnitt 2.5). Därmed att i ett fysiskt maskin skulle blandningsalgoritmeren köras i ett inbyggd miljö det vill säga i system med begränsade resurser. Dessutom på så sätt implicit skapas det en mer likvärdig till en potentiell kortblandingsmaskin miljö för efterföljande simulationer.

Kompromisen som togs i implementation av algoritmerna är att pseudoslumptalsgenerator (PRNG) som används i denna studie kommer ifrån Rand crate som har inbyggt implemention av ChaCha20 (se sektion 2.2 för detaljer). Som tidigare nämnts behöver ChaCha20 mer systemresurer och därför är inte tillgängligt i ett inbyggt system. Men valet av att använda denna togs för att i simulerings miljö där PRNG behövs successivt i små tids intervaller kommer den att generera bättre pseudoslumptal än en PRNG som är anpassad till inbyggda system. Dessutom kommer den att generera mer trövärdiga kortblandingar som är mindre influenserade av dess underliggande PRNG implemenation.

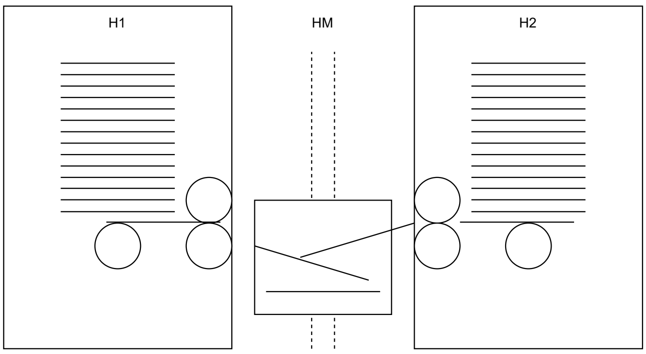
3.2.1 Anpassning av Riffle Shuffle

Första blandningsmetoden som anpassades var den matematiska Gilbert-Shannon–Reeds modell till en praktisk Rust kod (för djupgående bakgrund om modellen se sektion 2.3.1). För enkelhetens skull ska denna metod betacknas GSR Riffle Shuffle eller enbart GSR. Motiveringen av att utforska denna var p.g.a. dess lockande egenskaper i att hur matematiska modellen avspeglar människas kortblandningsprocess. Därför anpassades modellen för att utforska om den kan vara ett lämligt kandidat till den potentiella kortleksblandaren.

Matematiska formeln för GSR distrubution är anpassad på rad 6 se Algoritm 1. För att inte distrahera med tekniska utmaningar implementation i Rust finns i Bilaga B.

Algoritm 1 GSR Riffle Shuffle pseudokod

1: $H1 \leftarrow OD$ ▷ Alla kort från OD (Original Deck) är placerade i $H1$
2: $H2 \leftarrow \frac{H1}{2}$ ▷ $H2$ får hälften av kort (i sekvensiell ordining)
3: $HM \leftarrow tomlista$ ▷ Mittersa högen
4: **while** $H1$ eller $H2$ inte är tomma **do**
5:     $r \leftarrow$ slumpmässigt tal mellan 0.0 och 1.0 ▷ PRNG för att få ett slumpmässigt tal
6:     **if** $r \leq$ (längden av $H1$)/(längden av $H1 +$ längden av $H2$) **then**
7:         flytta ett kort från $H1$ till HM
8:     **else**
9:         flytta ett kort från $H2$ till HM
10:     **end if**
11: **end while**
12: flytta alla kort från HM tillbaka till $H1$



Figur 4: Ett ilustrativt blandingsprocess diagram av Riffle Shuffle fundamentala mekaniska komponenter till den potentiella kortleksblandaren. Med 3 stycken högar $H1$, HM respektive $H2$. Där sträckor representerar kort, cirkelformen representerar en roterande kortmatningsmekanism och mittersta högen är en höjbar mekanism i y-led.

Potentielt fysikt implementation: Anpasnings metodiken av GSR inledes med att det skisades upp ett potentiell fysikt implementation av Riffle Shuffle utifrån algoritmen, se Figur 4. Detta är ett simplifierad version. För att t.ex. det borde finnas ett mekanism att flytta kortet från HM till $H1$ och $H2$. Kompliceringar i detta design kan inkludera timing problem d.v.s. att kortet från $H1$ träffar kortet från $H2$ på vägen till HM.

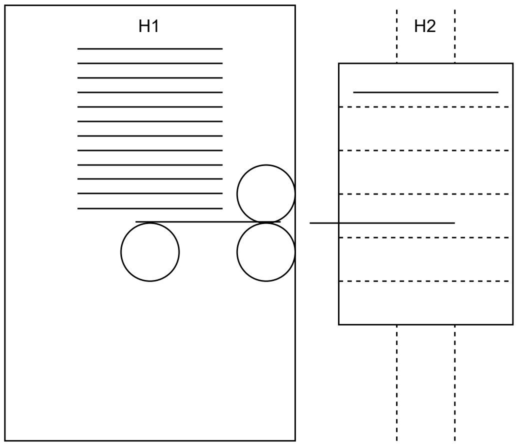
3.2.2 Anpassning av Pile Shuffle

Andra blandningsmetoden som anpassades var Pile Shuffle. För det första är den metoden design mässig annorlunda till Riffle Shuffle. Den har endast en roterande kortmatningsmekansim samt enbart två stora delar, se Figur 5. Insparation av det potentiella designen kom ifrån YouTube video 3DprintedLife (2021). I videon visas det en 3D printat kortleksblandare med Pile Shuffle algoritmen. Förkortningsvis denna metod ska betecknas till SOC Pile Shuffle eller enbart SOC. Denna metod har två variarande inställningar som inkluderar 1) antal fack n och 2) maximalt antal kort per fack M. I denna studie utforskades både SOC samt två till variationer av denna Six Pile Shuffle respektive Ten Pile Shuffle. Med följande inställningar: **SOC Pile Shuffle** med $n = 8$ och $M = 10$. Sedan utforskades hur påverkas slumpmässighet med färre fack därmed fysiska designen kan vara mindre. Detta blev till algoritmen som betcknas till **Six Pile Shuffle** med $n = 6$ och $M = 10$. Sist undersöktes om vad sker om inställningar blir till $n = 10$ och $M = 10$ denna algoritm kallas till **Ten Pile Shuffle**.

Algoritm 2 Pile Shuffle pseudokod

Indata: En kortlek $H1$
Utdata: Blandad kortlek $H1$

$n \leftarrow num$ ▷ Antal fack
$M \leftarrow num$ ▷ Max antal kort per facket
$H2 \leftarrow$ lista av n tomma listor
for $kort$ i $H1$ **do**
    loop
        $sf \leftarrow$ slumpmässigt fack index mellan 0 och $(n-1)$ ▷ slumpmässigt fack (sf)
        if längden av $H2[sf] < M$ **then**
            Lägg $kort$ i $H2[sf]$
            break
        end if
    end loop
end for
$H1 \leftarrow$ sammanfoga alla facken i $H2$ ▷ Samla ihop alla fack till en kortlek



Figur 5: Ett ilustrativt diagram av kortblandningsprocesen med Pile Shuffle metoden. Fundamentala mekaniska komponenter till den potentiella kortleksblandaren, där sträckor är kort, cirkelformen är roterande kortmatningsmekanismen. Med två högar, $H1$ och $H2$. Där $H1$ har plats för en kortlek medans $H2$ har n antal fack med maximalt M antal kort per fack.

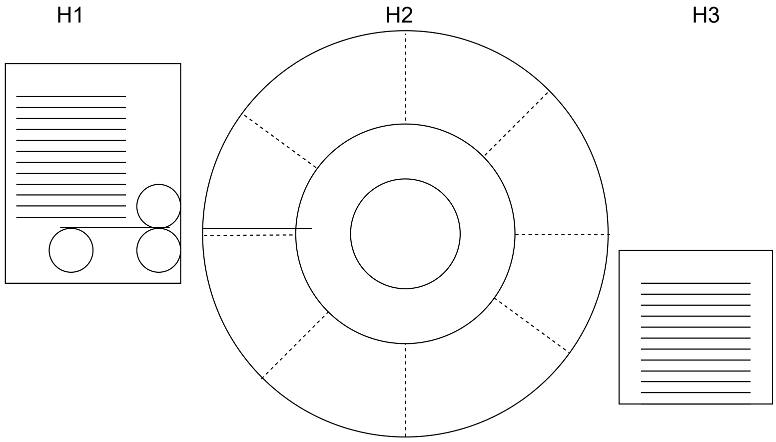
3.2.3 Anpassning av design av professionell kortleksblandare

Tredje blandningsmetoden som skapades var insprerad från en professionell kortleksblandare. P.g.a. denna design har den fack för varje kort därmed anpassades det Fisher-Yates Shuffle. D.v.s. först blandades det en lista med indexes med Fisher-Yates algoritmen och sedan kortet från $H1$ flyttas till $H2$ baserat på slumpmässigt index se Figur 6. Detta metoden teoretiskt sätt skulle kräva endast en iteration för att blanda en kortlek.

Algoritm 3 Wheel Fisher-Yates Shuffle pseudokod

Indata: En kortlek $H1$
Utdata: Blandad kortlek $H3$

$I \leftarrow$ lista med index 0 till 51 ▷ Skapa en lista med index
FisherYatesShuffle(I) ▷ Blanda indexen slumpmässigt
$H2 \leftarrow$ ny lista med 52 platser ▷ Skapa 'wheel' med plats för varje kort
for $i \leftarrow 0$ till längden av $D-1$ **do**
    $idx \leftarrow I[i]$ ▷ Välj ett slumpmässigt index från I
    $H2[idx] \leftarrow D[i]$ ▷ Placera kortet i 'wheel' baserat på slumpindex
end for
$H3 \leftarrow H2$ ▷ Uppdatera $H3$ med den nya ordningen från 'wheel'



Figur 6: Ett ilustrativt diagram av kortblandningsprocesen med Fisher-Yates metoden. Fundamentala mekaniska komponenter till den potentiella kortleksblandaren, där sträck är kort, cirkelformen är roterande kortmatningsmekanismen. Med två högar, $H1$ och $H3$. Där $H2$ har ett fack för varje kort, d.v.s. 52 stycken. Och $H3$ är plats till en blandad kortlek.

3.3 Simulering av kortblandningsprocesser

Simulations- och blandningsalgoritmerna implementerades i programmerings språk Rust version 1.73.0. Rust valdes p.g.a dess robusta stöd för abstraktioner utan bekostnad.

Simulationen avspeglar hur ett fysiskt kortleksblandare skulle fungera. Därför togs det valet att alla simulationer ska ha ett och samma utgångspunkt. Därmed i analysen kan det jämföras olika metoder beroende på deras iterationer. D.v.s. ett verkligt scenarium simulerades vart ett helt ny oöppnad kortlek skulle placeras i den potentiella kortleksblandare. Detta ordningen kallas för ett fabriksordning för en standard 52-kortlek. Där korterna är ordnade efter sitt färg i sekvensiell ordning från två till ess (utan jokrarna). Matematiskt kan detta ordningen beskrivas som mängden $\{x \in \mathbb{N}, 0 \leq x \leq 51\}$, där x representerar en enskild kort och vart ordningen spelar roll.

För att bestäma datamängden d.v.s. antal kortlekar per en simulation. Användes det rekomendationen för chi-två-testet. Där NIST rådger att minsta förekommande kategorin för pokertest borde inte vara mindre än 5 stycken (se sektion 2.4.1). I poker är den mest sällsynta pokerhanden den kungliga färgstegen (Royal Flush) och det finns 4 stycken av dessa i en standardkortlek. Den teoretiska sannolikheten att få 5 Royal Flushes per m kortdelnigar kan aproximeras på följande sätt:

$$P(\text{Kunglig färgstege}) = \frac{\binom{4}{1}}{\binom{52}{5}} = \frac{4}{2\,598\,960} \approx \frac{1}{649\,740}$$

Sedan denna resultatet skalas med faktor av 5: $m = P(\text{Kunglig färgstege})^{-1} \times 5 = 3\,248\,700$. Denna resulterande värden på m används som den absoluta längden av datamängden i simulationen. För att visualisera detta, låt D vara en matris med 52 kolumner (antal kort i standardkortleken) och m antal rader (längden av datamänden), och där x är en av talen ur den tidigare definerade mängden.

$$D = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,51} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,51} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,51} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{m,0} & x_{m,1} & x_{m,2} & \cdots & x_{m,51} \end{bmatrix}$$

Via expermintell metodik valdes det att utföra 15 iterationer per algoritm. En iteration defineras till att kortleken blandas successiv på föregånde blandning. Detta gjordes för att senare kunna jämföra hur algoritmens slumpmässighet påverkas av ökande antal iterationer. Dessutom kan detta valet motiveras av att GSRs matematiska modellen visar sig att vara mest effektiv vid 7 och 11 iterationer (se sektion 2.3.1). Detta antal iterationer kunde ske p.g.a att simulationsprocessen effektiviserades med Rayon crate. Med denna verktyg utfördes programmet i parallellt exekvering.

För att förutspå hur mycket RAM och hårddisk minne skulle eventuellt användas vid lagring och datanalys. Räknades det ut tynged av en datamängd ska väga $161MB$ vid användning av 8bit datatypen.

$$\frac{\text{totala bytes}}{\text{megabyte (MB)}} = \frac{52 \times m}{1024 \times 1024} = \frac{52 \times 3\,248\,700}{1024 \times 1024} = \frac{168\,932\,400}{1\,048\,576} \approx 161MB$$

Detta steget behövdes för att ta ett informativ beslut senare. D.v.s. hur data ska analysers och eventuella anpassas.

Som det tidagare nämndes utfördes simulationen parallellt där varje algoritm exekverades på egen tråd, oberoende av varandra. Simulationen utfördes i följande steg: 1) Algoritmenerna sparades i en lista; 2) En lista av kortlekar, motsvarande matrisen D med m antal kortlekar, initierades till fabriksordningen; 3) Algoritmerna itererades i en nästlad loop från $i = 1$ till $i = 15$; 4) Ledtidsmätaren sätts igång; 5) Denna iteratinsvariabel i användes för att iterera över listan av kortlekar och utföra blandningen i gånger; 6) Efter varje iteration i adderades ledtiden; 7) Efter avklarad iteration i medelvärde av ledtiden kalkylerades och sparades i en csv fil; 8) Resulterande blandade kortlekarna utplattedes till en endimensionell lista; 9) Denna endimensionella lista sparades som en binärfil för statistiskanalys.

3.4 Statistisk analys av insamlad data.

Detta sektion handlar om hur statistiska metoder som Poker test och STDMean test implementerades för att analyser slumpmässighet av valda och simulerade algoritmerna. Statistiska tester utfördes med Python version 3.11. Python valdes p.g.a dens breda använding vid statistisk analys (se sektion 2.5).

Efter simulationen, insamlad data är laddat in och återställt tillbaka till formen av matrisen D från sektion 3.3 med hjälp av NumPy för vidare bearbetning. Analysmetoderna delar inladdat data p.g.a. optimeringskall som e.g. spara minnen och minimisera initial exekveringstid.

På grund av begränsad kunskap och expertis inom avancerad statistisk matematik, även om metoder såsom Aproximate Entropi (ApEn) skulle kunna erbjuda ytterligare insikter i slumpmässighetsanalys (Delgado-Bonal och Marshak, 2019), valdes det att inte använda dem i denna studie. Detta beslut baserades på bristen på fördjupade kunskaper inom detta område samt tidsbegränsingen. Istället fokus ligger på en annan branch av slumpmässighet baserad på sannolikhetslära.

3.4.1 Implemention av den klassiska poker testet

Syfte med denna metod är att utföra preliminär galring av algoritmernas iterationer som har betyande avvikelser från dem förväntade värdena av en slumpmässigt blandningsalgoritm.

Den klassiska poker test utgår ifrån att man karaktäriserar typer av mönster till pokerhänder och kalkylerar den absolut värde av χ^2 i detalj beskrevs teorin bakom chi-två-test i sektion 2.4. Den klassiska poker testet i denna studie baseras på faktumet att simuleringar utfördes med en standard kortlek. Därför valdes det att anpassa chi-två-testet att använda alla poker händer. Matematiken, speciellt kombinatoriken att räkna ut sannolikheter och frekvensen av alla poker händer är relativt svår uppdrag därför addopterdes det uträkningar ifrån studie av Armstrong (2006), se tabell 3. Detta metodiken medföljde med mer komplex karakterisering av pokerhänder än när man använder chi-två-testet till att e.g. testa slumptalsgeneratorer. Motiveringen till komplixiteten var att i denna studie utforskas särkilt kortlekar till spel vart kombinationer av kort ofta spelar betyande roll och därför var det särkilt viktigt att anpassa slumpmässighets analysmetoderna till hur den potentiella kortleksblandare skulle fungera i drift.

Tabell 3: Namn på alla pokerhänder och dess relativ mönster. Antal_1 är teoretisk framkommande kombinationer med en standard kortlek. Antal_2 är faktoriserade kombinationer d.v.s $\text{Antal}_2 = \text{Antal}_1 \times 1.25$

Pokerhand	Example mönstren	Antal_1	Antal_2
Royal Flush	$A\spadesuit\, K\spadesuit\, Q\spadesuit\, J\spadesuit\, 10\spadesuit$	4	5
Färgstege	$K\spadesuit\, Q\spadesuit\, J\spadesuit\, 10\spadesuit\, 9\spadesuit$	36	45
Fyrtal	$A\spadesuit\, A\heartsuit\, A\diamondsuit\, A\clubsuit\, K\spadesuit$	624	780
Kåk	$A\spadesuit\, A\heartsuit\, A\diamondsuit\, K\clubsuit\, K\spadesuit$	3 744	4 680
Färg	$K\spadesuit\, Q\spadesuit\, J\spadesuit, 10\spadesuit\, 8\spadesuit$	5 108	6 385
Stege	$K\spadesuit\, Q\heartsuit\, J\diamondsuit\, 10\clubsuit\, 9\spadesuit$	10 200	12 750
Triss	$A\spadesuit\, A\heartsuit\, A\diamondsuit\, K\clubsuit\, Q\spadesuit$	54 912	68 640
Två par	$A\spadesuit\, A\heartsuit\, K\diamondsuit\, K\clubsuit\, Q\spadesuit$	123 552	154 440
Ett par	$A\spadesuit\, A\heartsuit, K\diamondsuit\, Q\clubsuit\, J\spadesuit$	1 098 240	1 372 800
Högt kort	$A\spadesuit\, Q\heartsuit\, J\diamondsuit\, 5\clubsuit\, 4\spadesuit$	1 302 540	1 628 175
	Summan:	2 598 960	3 248 700

Kategeriseringsprocessen till att få ett hand till ett pokerhand börjades med att det uttnytjades NumPy vektoriserad aritmetik funktionalitet. Med denna funktionen på ett effektivt sätt valdes ut 5 kort i.e en hand och kördes igenom en katigoriserings funktion om denna senare. Dem 5 kort valdes ut på ett speciellt sätt för att göra denna likvärdig verkligheten. D.v.s. pokerspel med 2 spelare. Där valdes det kort med index 0, 2, 5, 6 och 7 (spelare-ett: 2 kort i handen, spelare-2 kort förkastas och 3 sista kort är flop) med NumPy funktionalitet kördes katigoriserings funktionen rad mässigt.

katigoriserings funktion: I standardkortlek finns det 52 kort med fyra olika färger (Spader, Hjärter, Ruter och klöver) och 13 valörer (2, 3, 4, 5, 6, 7, 8, 9, 10, Knekt, Dam, Kung, Ess). Konvertering av tidigare definerade mängden av kort x, i.e. (0, 1, 2, ..., 51) till katgeriserbart kort utfördes. För att optimera kategoriseringsproccessen användes Numba för att skapa en ny tråd, detta kunde enbart göras p.g.a. att det användes den fundamentala 8bit datatypen, som definerades tidigare. Efter att ett ny tråd skapades proccessen försattes på följande sätt:

$$\text{valör}(x) = x \mod 13$$

$$\text{färg}(x) = \left\lfloor \frac{x}{13} \right\rfloor$$

där valör(x) ger värde från 0-12 och färg(x) ger värde från 0-3. För dem intresserade av specifika kod detaljerna se Bilaga A. Försättningsvis denna funktion gav tillbaka ett värde från 0-9 som indikerar ett pokerhand ifrån Tabell 3, där 0 är Högt kort och 9 Royal Flush. För att denna funktionen kördes kolumn mässigt över hela datamängded var denna resulterande lista med antal observerade pokerhänder från varje rad, låt denna lista betecknas till O (Observerat frekvens). I bästa fall skulle denna lista fått likadana värdena som kolumnen Antal$_2$ i Tabell 3.

Chi-två-test: Inställningar till testet var $\alpha = 0.05$ (Signifikansnivå), df $= 10 - 1 = 9$ (frihetsgrader), där 10 är den antal pokerhänder givna av Tabell 3. Sedan beräknades gränsvärde med SciPy biblioteks funktionen $Chi2.ppf(1-\alpha, df)$. Sedan används det *chisquare* funktionen ifrån SciPy för att beräkna p-värde och χ^2 värde. För att kalkylera denna användes tidigare genererade listan O och dem förväntade värden given av kolumnen Antal$_2$ i Tabell 3. Resulterande värdena på p-värde och χ^2 sparades i en csv fil för redovisningen i Resultat sektionen.

3.4.2 Implemention av STDMean testet

Syfte med denna analysmetod är att utforska hur enskild kort rör sig genom kortleken när den blandas. Därmed avgöra om algoritmen har eller inte har ett tendens att blanda specifika delar av kortleken mer än dem andra. Detta i sin tur är särkilt viktigt för att i kortspel, där positioner på första korterna spelar betynade roll i kortspels rätvisshet. Snedvridningen kan alltså potentiell indikera logiska fell i implementionen av en algoritm. I initiala tester användes STDMean testet för att upptäcka logiska fel. Detta hände i den första versionen av implementation av GSR Riffle Shuffle, där det på fell sätt användes indexering i iterationen, felet upptäcktes med hjälp av denna analysmetoden.

För att förstå metoden är det relevant att återbesöka konceptet med matrisen D i sektion 3.3. Därför att det uttnytjade NumPy vektoriserad aritmetik i simbios med dens inbyggda funktioner som medelvärde och standardavvikelse. Både funktionera utfärdes kolumn vis på m antal rader. Denna resultatet användes sedan till att rita punktdiagram av Medelvärde av Position (MP) på y-led och Position av Kort (PK) på x-led med hjälp av matplotlib. Standardavviklse av PK visas som vertikal symetrisk linje ifrån PK punkten. Den teoretiska medelvärdet i en slumpmässigt blanding borde ligga runt $51 \div 2 = 25.5$ och den experimentellt testade SD ifrån PK ≈ 15.

4 Resultat

Totalt 50 simulationer utfördes. 15 simulationer per algoritm för att jämföra samma algoritm med olika antal iterationer, varje algoritm har alltså resultat som motsvarar utförelse med iterationer 1-15. Dem mest relevanta resultat av simulationerna plockades ur och presenteras här. Resultat av poker test samt medelvärde för algoritmers ledtid presenteras i tabellen nedan. Resultaten av STDMean testet visas i Bilaga E. Vilket kort som en punkt i diagrammet representerar ges av punktes position på x-axeln, medelvärde är avläst från punktens position på y-axeln och standardavvikelse är givet av den symetriska vertikala linjen som går genom punkten.

GSR Riffle Shuffle: Pokertest visade en variarande slumpmässighet med första giltiga resultat vid iteration 7. Efter det slumpmässighet försämrades. Speciellt iteration 9 som överstig gränsvärden på 16.92 men efter denna iteration resultat förbätrades igen. Ledtiden visade en stabil värde men efter iteration 9 ledtiden minskade med ca 15% relativt till iteration 3 (se Tabell 4a). Resulat av STDMean testet visade en stigande trend av medelvärde av position (MP) över positon av kort (PK) samt dess standardavvikelse (SD) var jämnt för all PK. Iteration 3 visade ett stigande trend (se Bilaga E 8) men vid iteration 7 normaliserades denna till den teoretiska beräknade värden av MP = 25.5 (se Bilaga E Tabell 10).

Six Pile Shuffle: Iteration 1 och 2, pokertestet visade att χ^2 värdena låg över gränsvärden. Iteration 3 visade det första giltiga resultat. Vid nästkommande iterationer låg gränsvärden inom normen, med ingen tydlig trend. Vid ökade iterationer ledtidenerna verkar varit konstanta med minimal variation på ca +8% mellan iteration 1 och 7 (se Tabell 4b). STDMean testet vid iteration 1 visade värdena av stor betydlse, var PK 0-2 hade låg SD värde ca 5.4 vid PK 1 och ca SD 9.0 vid PK 2 d.v.s. värden ökade i denna intervall med SD ca 2.0 (se Bilaga E Tabell 11). Sedan dess värdena varit variarande som liknade sinusfunktion. Men vid PK 49-51 visades det exakt likadan mönster som i PK 0-2. Vid iteration 2 uttpladades värdena men vart tionde PK hade avvikelser i MP (Bilaga E Tabell 12). Sedan vid iteration 3, detta mönstren minskades (Bilaga E Tabell 13). Sedan vid iteration 4 uttplatades MP värdena helt och var vid den teoretiska värden av MP (Bilaga E Tabell 14).

SOC Pile Shuffle: Från pokertestet, första giltiga resultat visades vid iteration 2. Medans i iteration 3 överstig gränsvärden medans dem succsesiva iterationer låg χ^2 värdena inom den kalkylerade gränsvärden. Ledtiden från iteration 7 visade likadan variation på +8% relativt till iteration 1 (se Tabell 4c). Resultatet av STDMean testet iteration 1 visade samma mönster som i Six Pile Shuffle. För PK 0-2 och 49-51 men SD av PK 0 och 51 var med ca 2.0 större än i Six Pile Shuffle (Bilaga E Tabell 15). Medans PK från 15-36 hade MP som låg runt 25.4. Iteration 2 hade synlig avvikelse av MP för PK 0 och 51. Annars enbart PK 10 och 41 hade synlig avvikelse från den teoretiska MP (Bilaga E Tabell 16). Iteration 3 hade endast första och sista PK som hade synlig avvikelse från MP (Bilaga E Tabell 17). Medans vid iteration 4 låg alla MP på den teoretiska värden (Bilaga E Tabell 18).

Ten Pile Shuffle: Iteration 1 enligt resultatet av pokertestet visade ett icke slumpmässigt iteration. Med första giltiga resultat var vid iteration 2. Dessutom iteration 2 till 5 visade stabila χ^2 värden runt 5.28 (medelvärde av dess 4 iterationer). Ledtiden hade likadan trend som föregående Pile Shuffles d.v.s. ökat värde på ca 8% från iteration 1 till 7 (se Tabell 4d). STDMean testet vid iteration 1 visade likadan SD mönster för PK men vid högre MP värdena än SOC Pile Shuffle (Bilaga E Tabell 19). Samt iteration 2 hade likadan mönster (Bilaga E Tabell 20). Medans iteration 3 hade väldigt liten avvikelse i PK 0 och 51 (Bilaga E Tabell 17). Vid iteration 4 var alla MP värdena vid den teoretiska värden (Bilaga E Tabell 18).

Wheel Fisher-Yates Shuffle: Alla iteration var inom gränsvärden d.v.s. hade giltiga result enligt pokertestet. Men vid iteration 3 och 6 χ^2 värdena var relativt lägre än dem andra. Ledtiden från iteration 1 till 7 hade ett ökat värde på ca 9% (se Tabell 4e). STDMean testet visade att dess MP låg vid den beräknade värde redan vid första iteration såsom vid samtilga successiva iterationer (se Bilaga E Tabell 23).

Tabell 4: Resultatet från den klassiska pokertestet: Gränsvärde räknat ut till 16.92 vid en signifikansnivå (α) på 0.05 och med 9 frihetsgrader (df). Vart ledtiden representerar medelvärdet för en kortleksblandning per iteration.

(a) GSR Riffle Shuffle

Iteration	χ^2	p-värde	Ledtid [ns]
3	233941.28	0	1681
4	7077.30	0	1687
5	449.67	3.38	1660
6	34.46	7.42	1667
7	6.03	0.74	1654
8	8.50	0.48	1642
9	19.64	0.020	1582
10	8.88	0.45	1495
11	16.19	0.063	1432

(b) Six Pile Shuffle

Iteration	χ^2	p-värde	Ledtid [ns]
1	3854.97	0	1116
2	18.36	0.031	1124
3	12.27	0.20	1131
4	5.94	0.75	1184
5	12.34	0.19	1167
6	8.89	0.45	1168
7	4.33	0.89	1211

(c) SOC Pile Shuffle

Iteration	χ^2	p-värde	Ledtid [ns]
1	3349.24	0	1450
2	7.02	0.64	1465
3	19.11	0.024	1518
4	7.53	0.58	1441
5	8.50	0.48	1501
6	12.34	0.19	1536
7	8.36	0.50	1577

(d) Ten Pile Shuffle

Iteration	χ^2	p-värde	Ledtid [ns]
1	747.30	4.65^{-155}	1587
2	5.83	0.76	1654
3	4.83	0.85	1716
4	5.38	0.80	1625
5	5.09	0.82	1697
6	8.80	0.46	1698
7	14.46	0.11	1710

(e) Wheel Fisher-Yates Shuffle

Iteration	χ^2	p-värde	Ledtid [ns]
1	10.71	0.30	715
2	13.46	0.14	720
3	6.66	0.67	728
4	9.05	0.43	733
5	8.95	0.44	772
6	6.10	0.73	794
7	9.41	0.40	778

5 Diskussion

Diskussion om resultaten av simulationerna är uppdelat i flera subsektioner i syfte att göra det enklare att följa tankeproccesen lättare att följa. Den första subsektionen 5.1 handlar om hur den mest passade blandningsmetoden togs fram, den är uppdelad i två delar. Den preliminära gallringen 5.1.1 är där dem blandningsmetoder som är självklart opassande gallras bort från undersökingen. I jämförelsen av blandningsmetoderna 5.1.2 kommer vi närmare undersöka dem metoder som återstår efter gallringen för att komma fram till vilken av dem som passar bäst inpå dem krav som ställdes i syftet 1.2. Felkällor och källkritik 5.2 nämns efter att den mest passande blandningsmetoden tagit fram för att att diskutera deras påverkan på dem tidigare slutsatserna i diskussionen. I slutsatsen 5.3 återpresenteras den blandningsmetod som undersökingen kom from till vad det innebär och vilka förbättringar som skulle kunna göras i potentiell framtida forskning.

5.1 Den mest passande blandningsmetoden

5.1.1 Preliminär gallring

Dem algoritmer som gallras är dem som helt säkert inte kommer vara bättre en dem andra i dem färdigheterna som blandningsmetoderna jämförs på. Alla blandningsmetoder som var signifikanta enligt det klassiska poker testet 3.4.1 kan gallras då dem inte uppnår ett av kraven för blandningsmetoder 1.2, dem blandar inte på ett helt slumpmässigt sätt. Effektivitet är också viktigt för den mest passande blandningsmetoden som undersökingen syftar på att ta fram 1.2, därför är det logiskt att preliminärt gallra blandningsmetoder vars effektivitet är sämre till en grad som inte kan kompenseras för av dess andra egenskaper. Blandningsmetoderna som kan konstateras innehava en underlägsen effektivitet utan noggrannare analys är alla metoder med mer än tio iterationer. Metoder med 11 till och med 15 iterationer är alltså inte inkluderade i den närmare analysen och jämförelsen av blandningsmetoder.

5.1.2 Jämförelse av blandningsmetoderna

5.2 Felkällor och Källkritik

5.3 Slutsats

Bibliografi

3DprintedLife (2021). *Rigged Card Sorting Machine - ALWAYS Get The Hand You Want!* URL: `https://www.youtube.com/watch?v=eMTXyl7tPEk` (hämtad 2023-08-15).

Abdel-Rehim, Wael MF, Ismail A Ismail och Ehab Morsy (2014). ”Implementing the classical poker approach for Testing Randomness”. I: *International Journal* 4.8. URL: `https://www.researchgate.net/profile/Wael-Fawaz-2/publication/281178831_Implementing_the_Classical_Poker_Approach_for_Testing_Randomness/links/55da3a9608aec156b9ae74a7/Implementing-the-Classical-Poker-Approach-for-Testing-Randomness.pdf` (hämtad 2023-12-07).

Armstrong, Drew (2006). ”Probability of Poker Hands”. I: URL: `https://www-users.cse.umn.edu/~reiner/Classes/Poker.pdf` (hämtad 2023-12-09).

Delgado-Bonal, Alfonso och Alexander Marshak (2019). ”Approximate Entropy and Sample Entropy: A Comprehensive Tutorial”. I: *Entropy* 21.6. ISSN: 1099-4300. URL: `https://www.mdpi.com/1099-4300/21/6/541`.

Diaconis, Persi (2003). ”Mathematical developments from the analysis of riffle shuffling”. I: *Groups, combinatorics & geometry (Durham, 2001)*, s. 73–97.

Hannes, Salin (2011). ”Analys av pseudoslumptalsalgoritmer”. I: URL: `https://www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/rapport/salin_hannes_K11089.pdf` (hämtad 2024-02-06).

Harris, Charles R. m. fl. (2020). ”Array programming with NumPy”. I: *Nature* 585, s. 357–362. DOI: `10.1038/s41586-020-2649-2`.

Hunter, John D (2007). ”Matplotlib: A 2D graphics environment”. I: *Computing in science & engineering* 9.3, s. 90–95.

Lam, Siu Kwan, Antoine Pitrou och Stanley Seibert (2015). ”Numba: A LLVM-Based Python JIT Compiler”. I: LLVM ’15. URL: `https://doi.org/10.1145/2833157.2833162` (hämtad 2023-12-17).

Matsakis, Nicholas D och Felix S Klock II (2014). ”The rust language”. I: *ACM SIGAda Ada Letters*. Vol. 34. 3. ACM, s. 103–104.

Matsakis, Niko och Josh Stone (2022). *Rayon: Data-parallelism library for Rust*. Version 1.8.0. URL: `https://crates.io/crates/rayon`.

National Institute of Standards and Technology (2023). *Engineering Statistics Handbook - Section 1.3.5.15: Chi-Square Goodness-of-Fit Test*. URL: `https://www.itl.nist.gov/div898/handbook/eda/section3/eda35f.htm` (hämtad 2023-10-04).

Random.org (2024). *Introduction to Randomness and Random Numbers.* URL: https://www.random.org/randomness/ (hämtad 2024-02-08).

Sharma, Ayushi m. fl. (2023). ”Rust for Embedded Systems: Current State, Challenges and Open Problems”. I: *arXiv preprint arXiv:2311.05063.* URL: https://arxiv.org/pdf/2311.05063.pdf (hämtad 2023-12-09).

Terwijn, Sebastiaan A. (2016). ”The Mathematical Foundations of Randomness”. I: *The Challenge of Chance: A Multidisciplinary Approach from Science and the Humanities.* Utg. av Klaas Landsman och Ellen van Wolde. Cham: Springer International Publishing, s. 49–66. URL: https://doi.org/10.1007/978-3-319-26300-7_3 (hämtad 2023-04-10).

The Rand Project Developers (2022). *Rand: A Rust library for random number generation.* Version 0.8. URL: https://crates.io/crates/rand.

Van Rossum, Guido och Fred L. Drake (2009). *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace. ISBN: 1441412697.

Virtanen, Pauli m. fl. (2020). ”SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. I: *Nature Methods* 17, s. 261–272. DOI: 10.1038/s41592-019-0686-2.

Yan, Kevin (16 aug. 2016). *1-8 decks Casino Full-Automatic card shuffler.* URL: https://www.youtube.com/watch?v=txl3gqIfwHM (hämtad 2024-02-06).

Bilagor

A Källkod

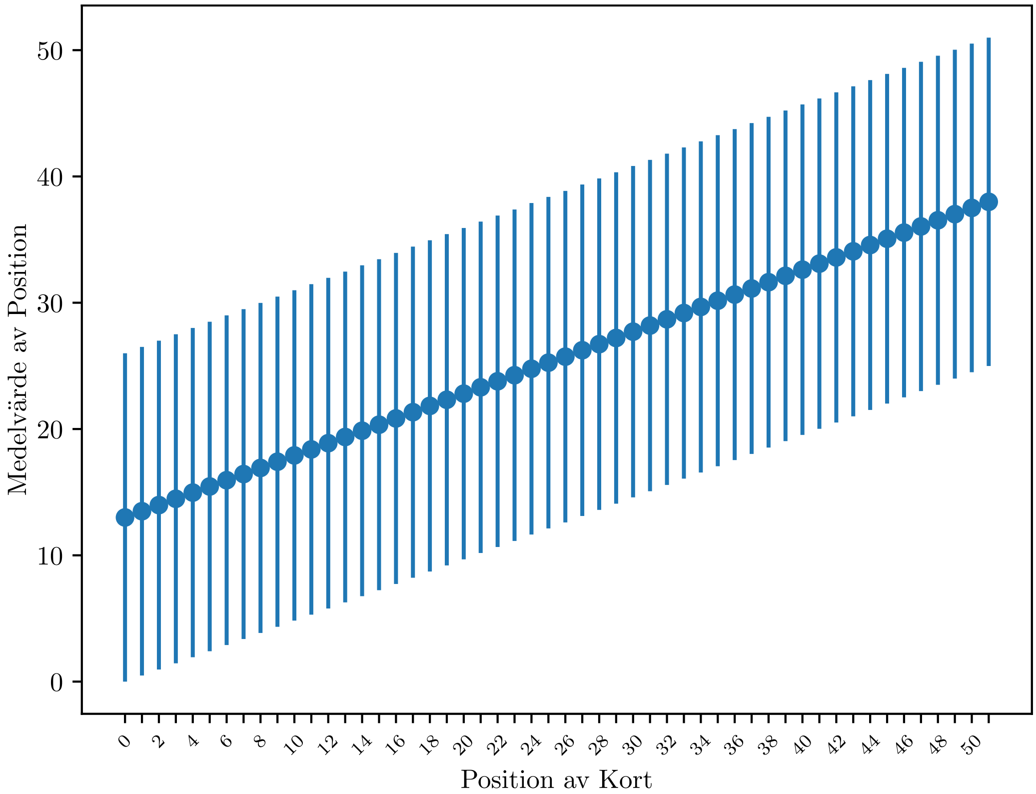
Länk till Github repository vart Rust och Python källkod till simulationen respektive statistikst anlys är samlad, samt källkod till latex med vilken detta rapport hade skrivits länk: https://github.com/Abishevs/gymarbete

B Kod för GSR Riffle Shuffle

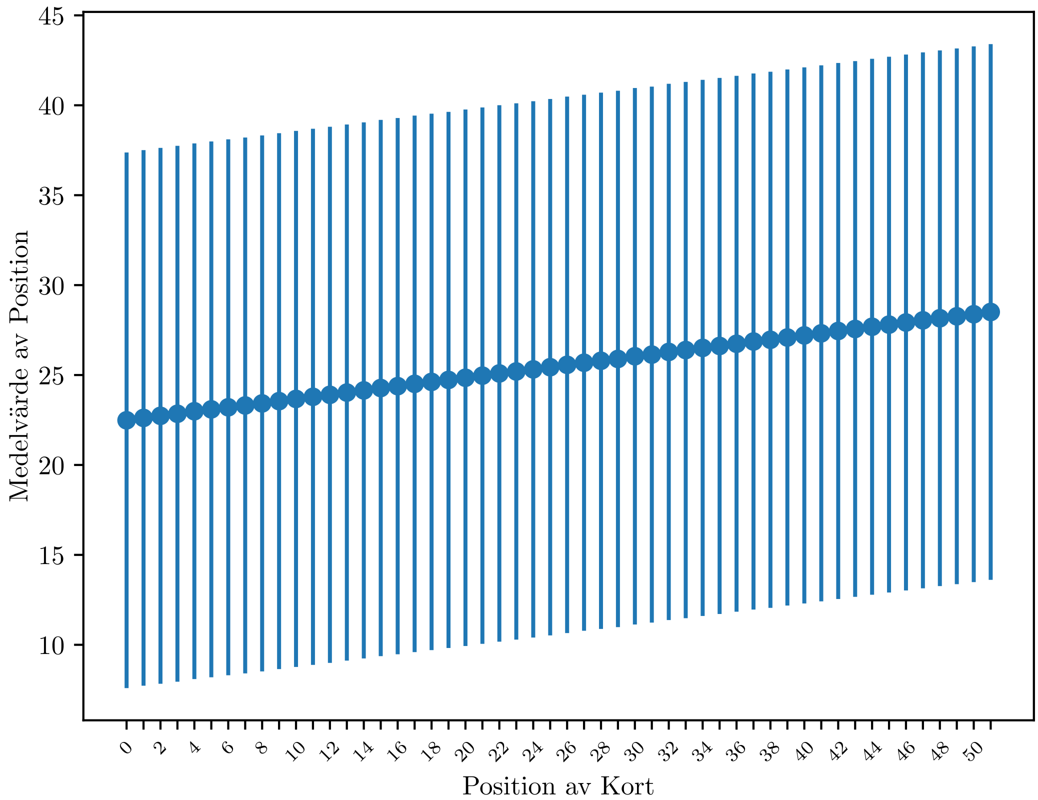
C Kod för Pile Shuffle

D Kod för Wheel Fisher-Yates shuffle

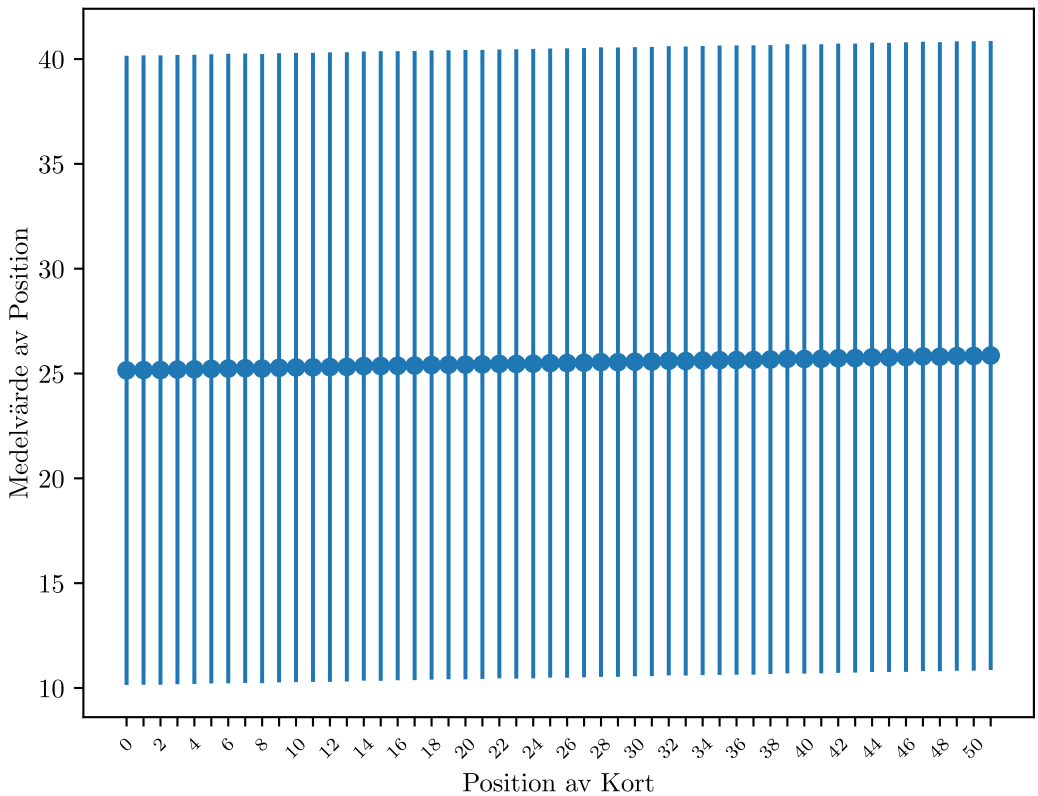
E Resultat av STDMean test



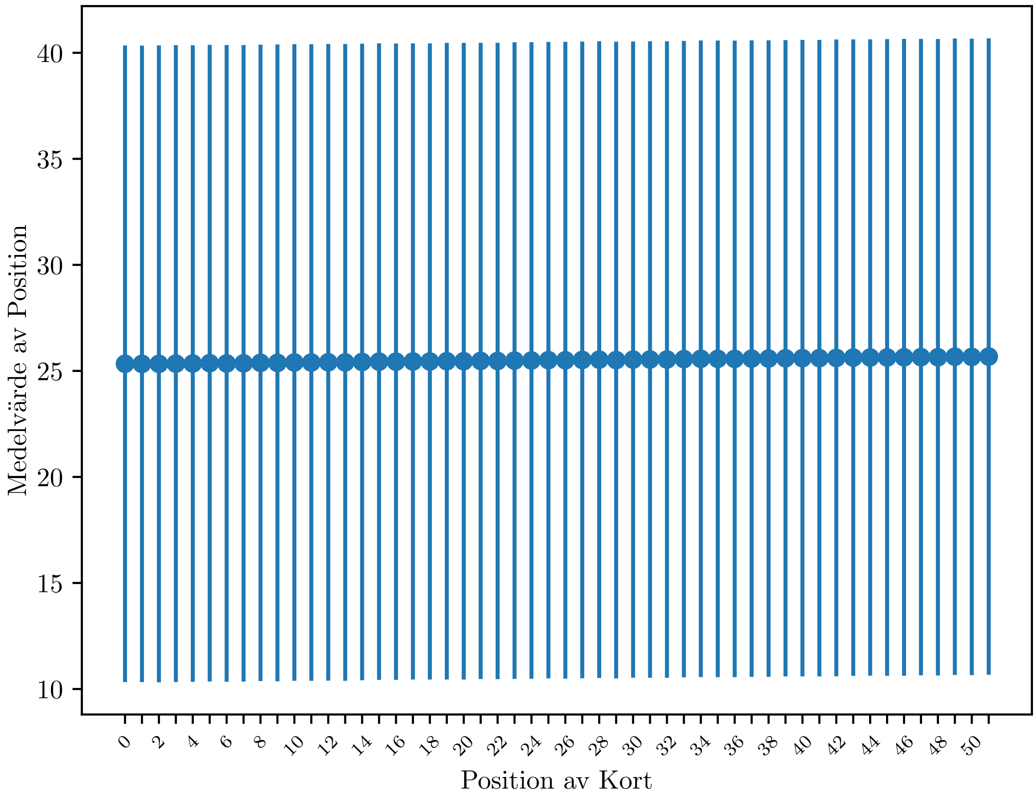
Figur 7: Resultatet från STDMean testet för GSR Riffle Shuffle med **1** iteration.



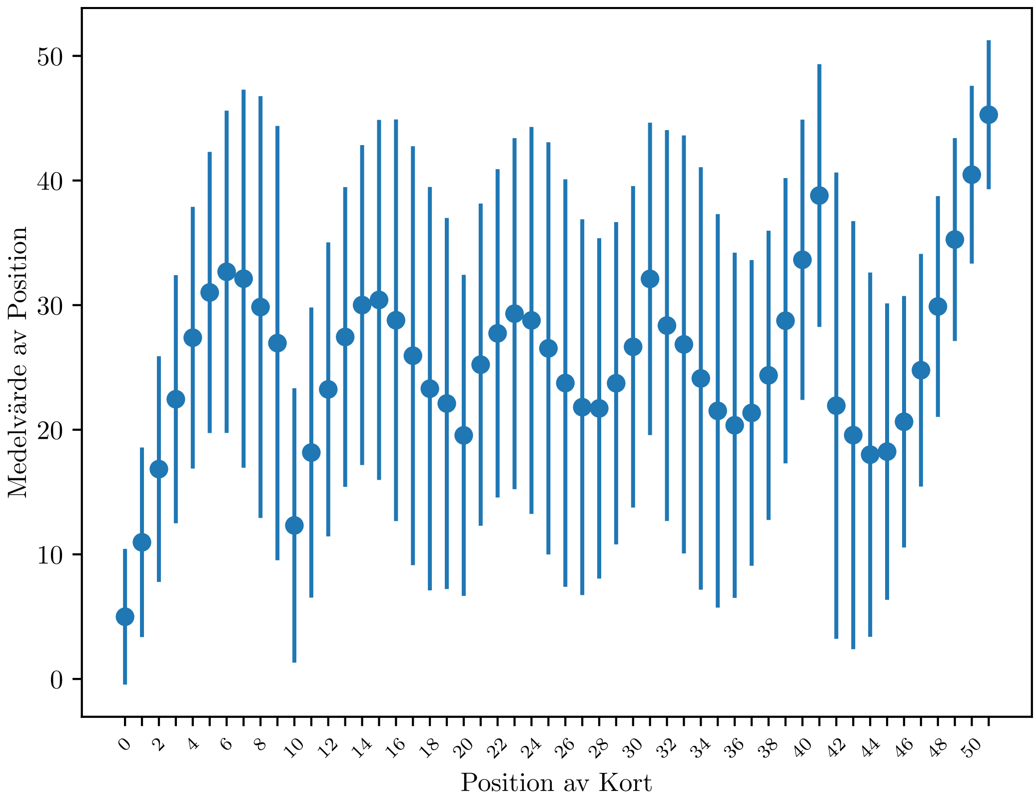
Figur 8: Resultatet från STDMean testet för GSR Riffle Shuffle med **3** iterationer.



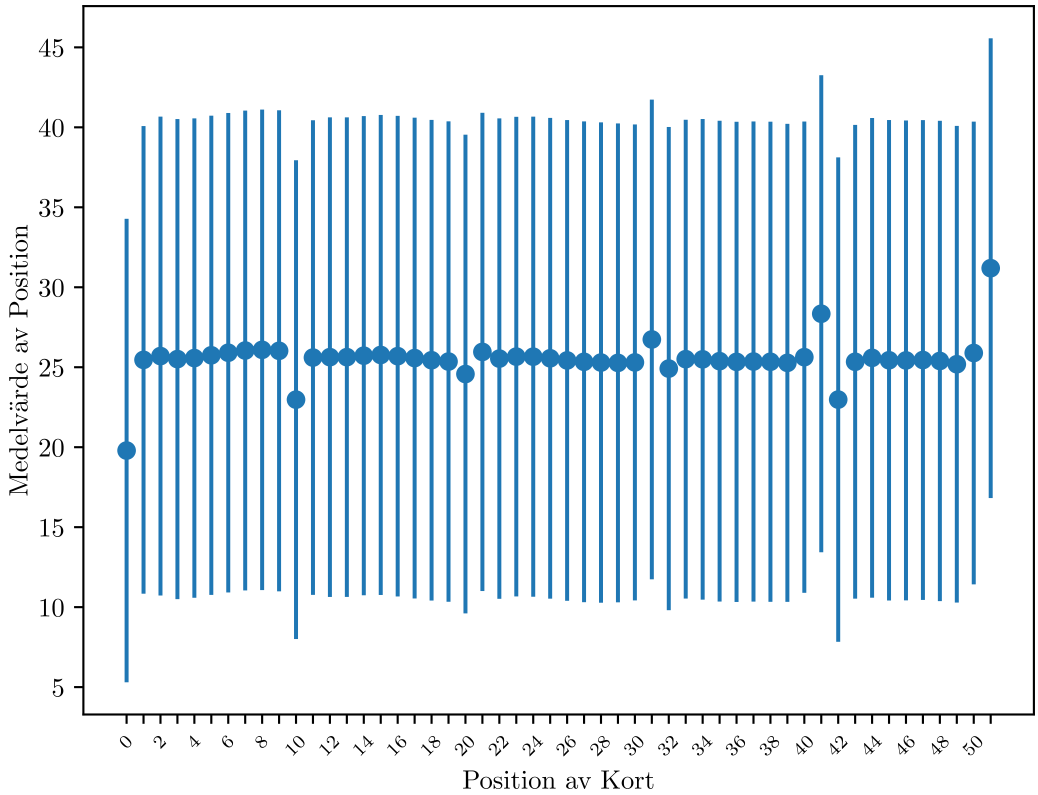
Figur 9: Resultatet från STDMean testet för GSR Riffle Shuffle med **6** iterationer.



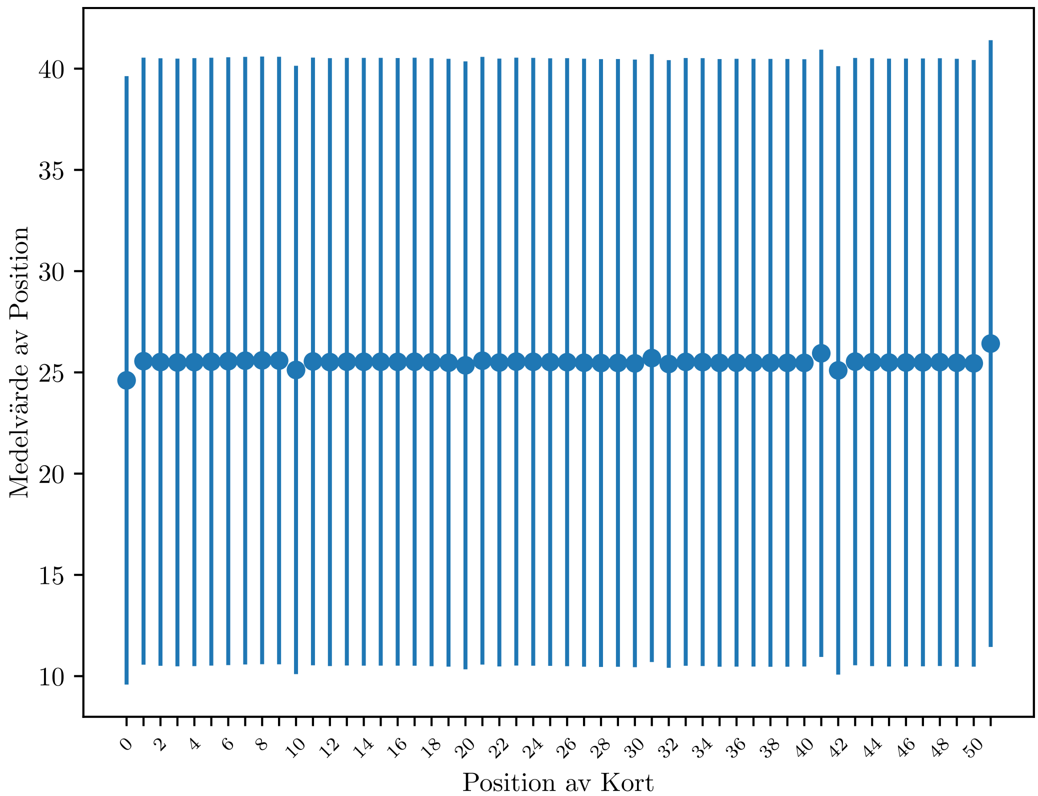
Figur 10: Resultatet från STDMean testet för GSR Riffle Shuffle med **7** iterationer.



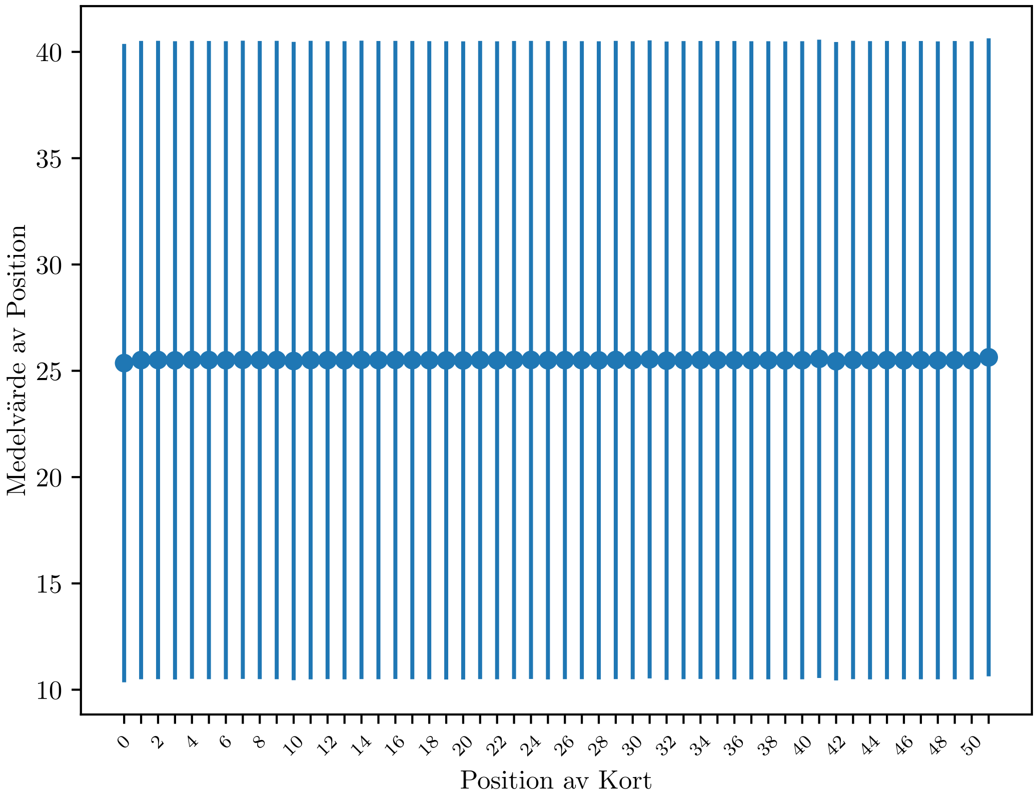
Figur 11: Resultatet från STDMean testet för Six Pile Shuffle med **1** iterationer.



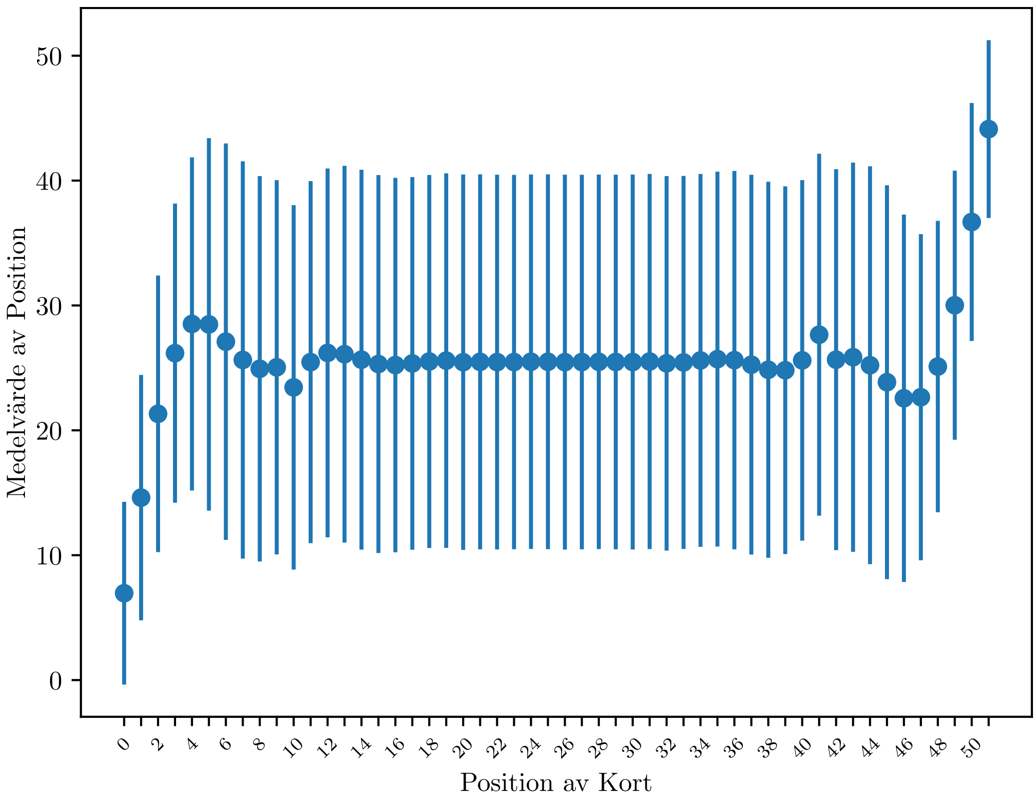
Figur 12: Resultatet från STDMean testet för Six Pile Shuffle med **2** iterationer.



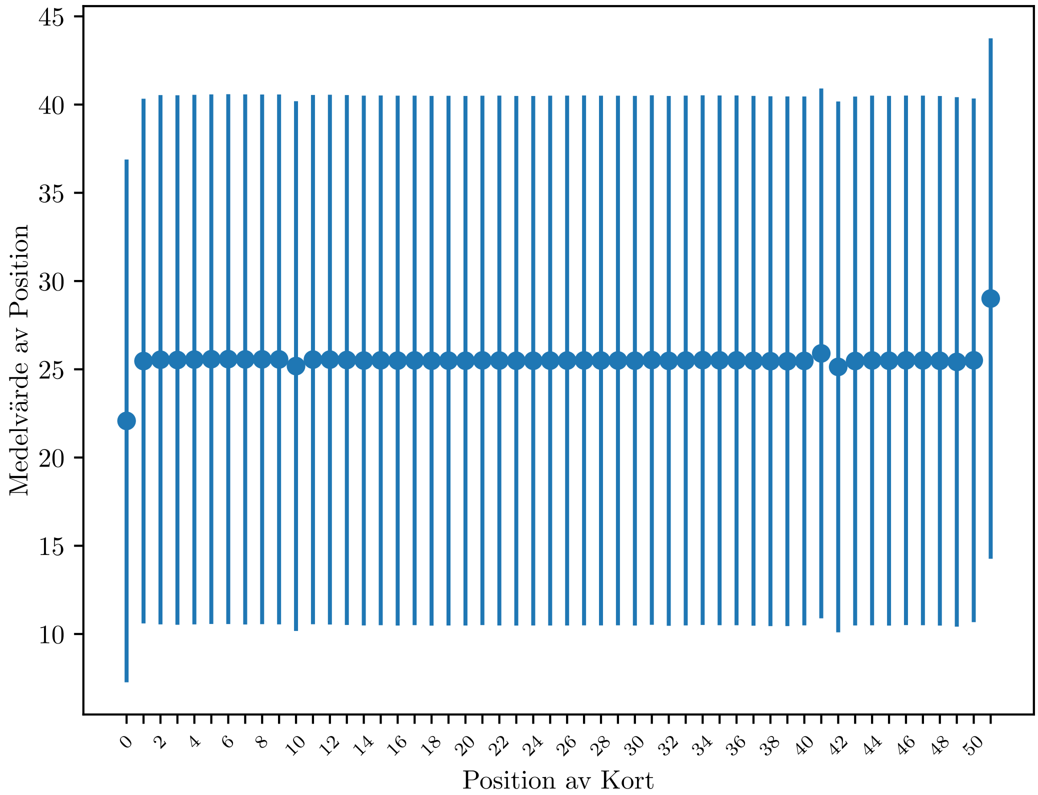
Figur 13: Resultatet från STDMean testet för Six Pile Shuffle med **3** iterationer.



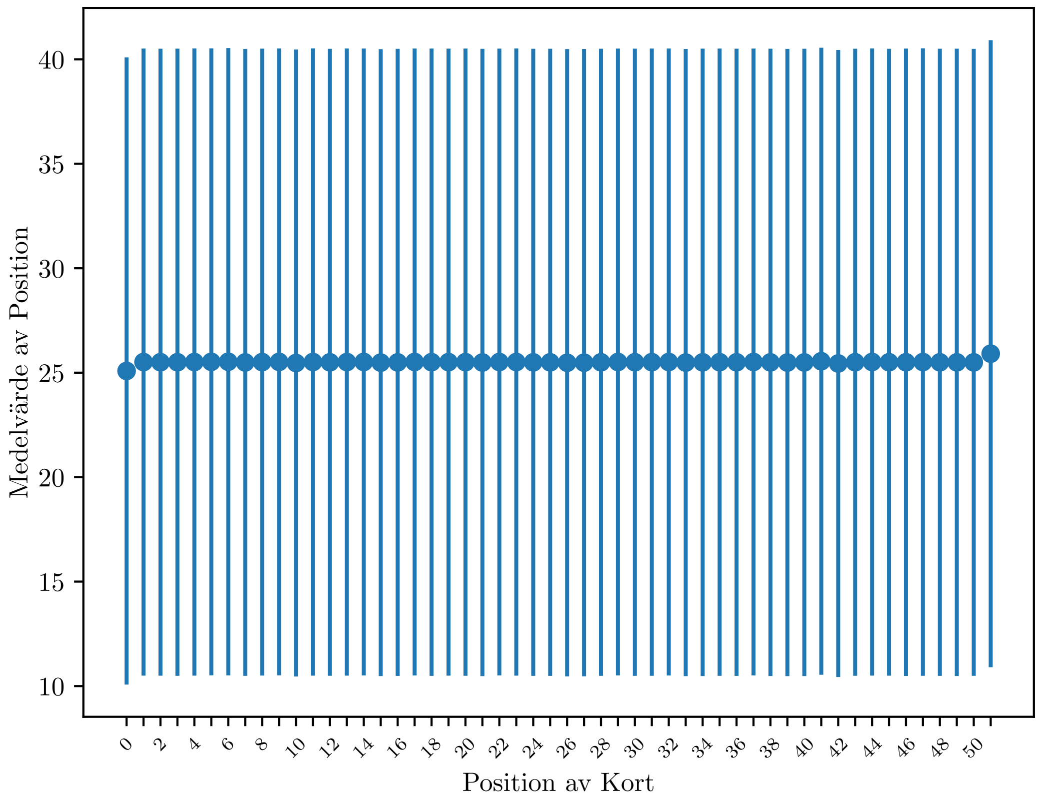
Figur 14: Resultatet från STDMean testet för Six Pile Shuffle med **4** iterationer.



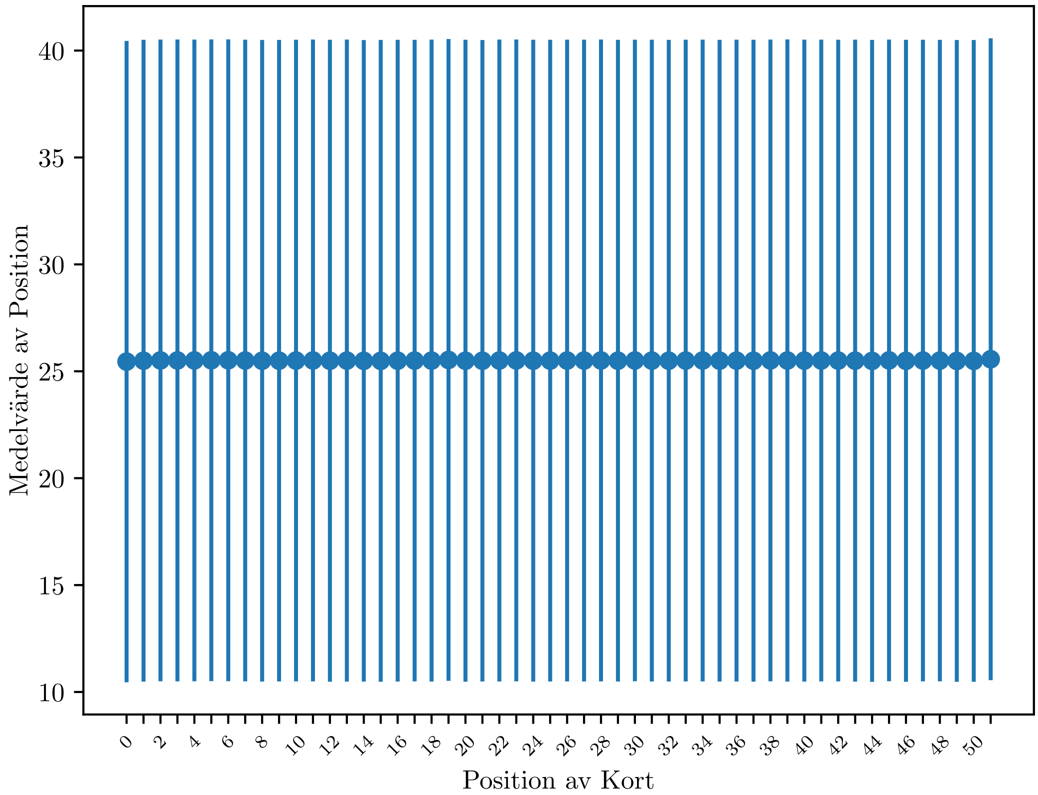
Figur 15: Resultatet från STDMean testet för SOC Pile Shuffle med **1** iteration.



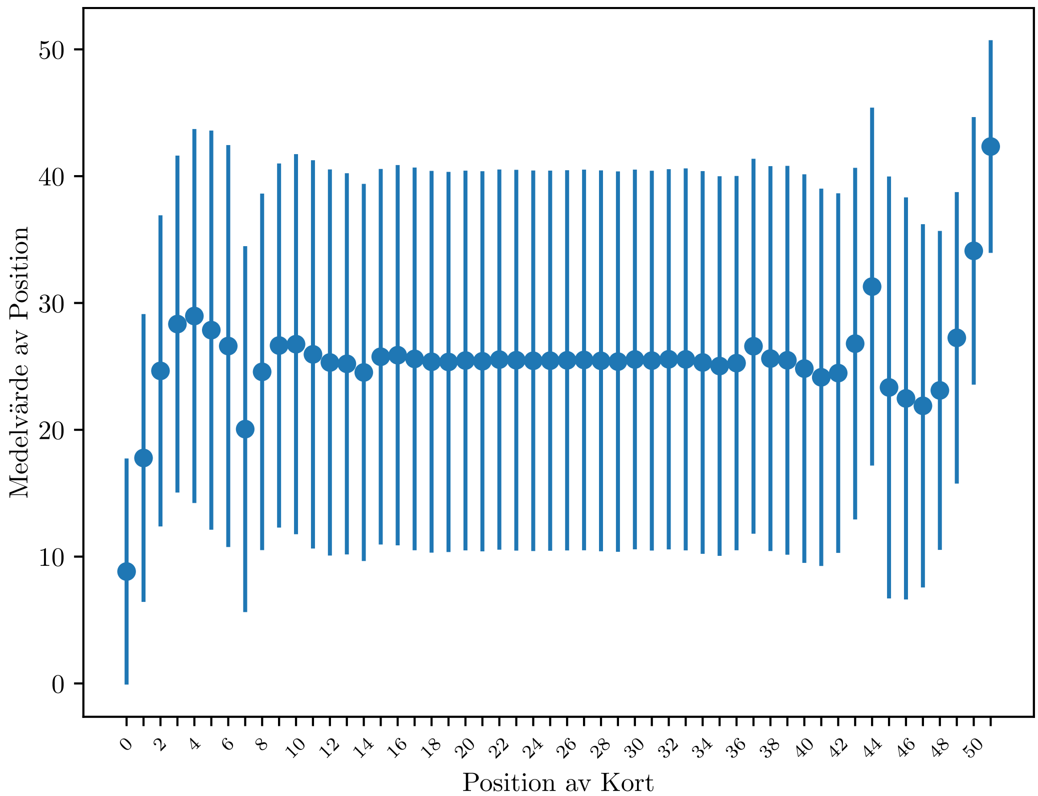
Figur 16: Resultatet från STDMean testet för SOC Pile Shuffle med **2** iterationer.



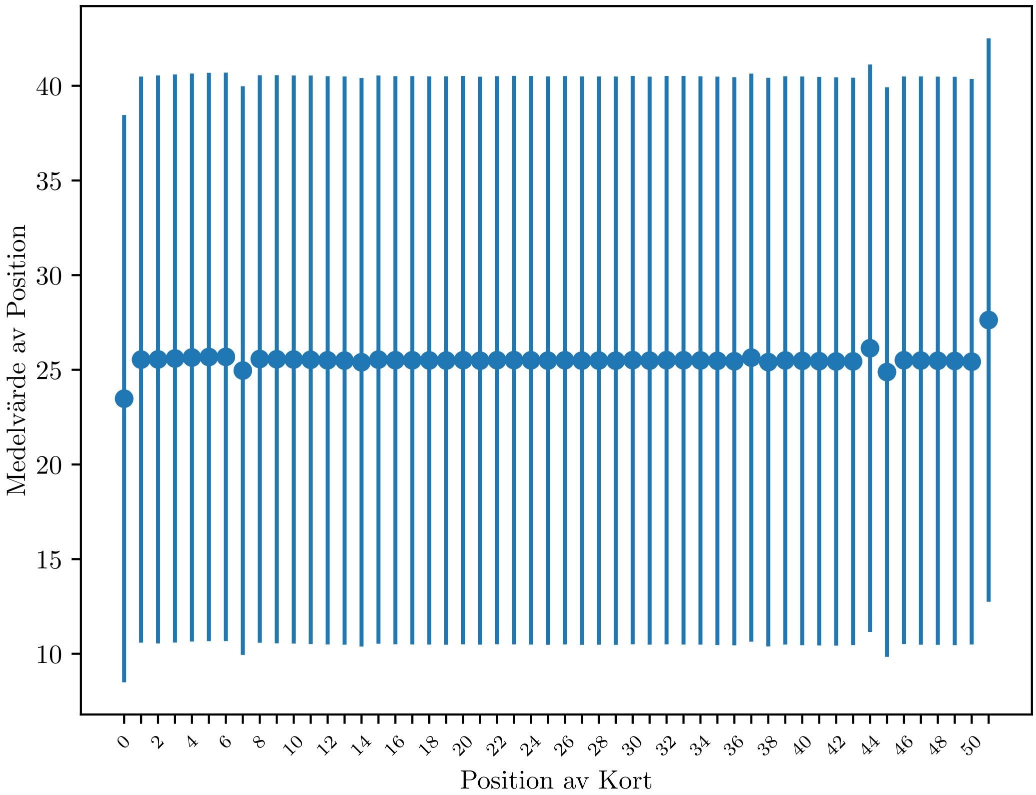
Figur 17: Resultatet från STDMean testet för SOC Pile Shuffle med **3** iterationer.



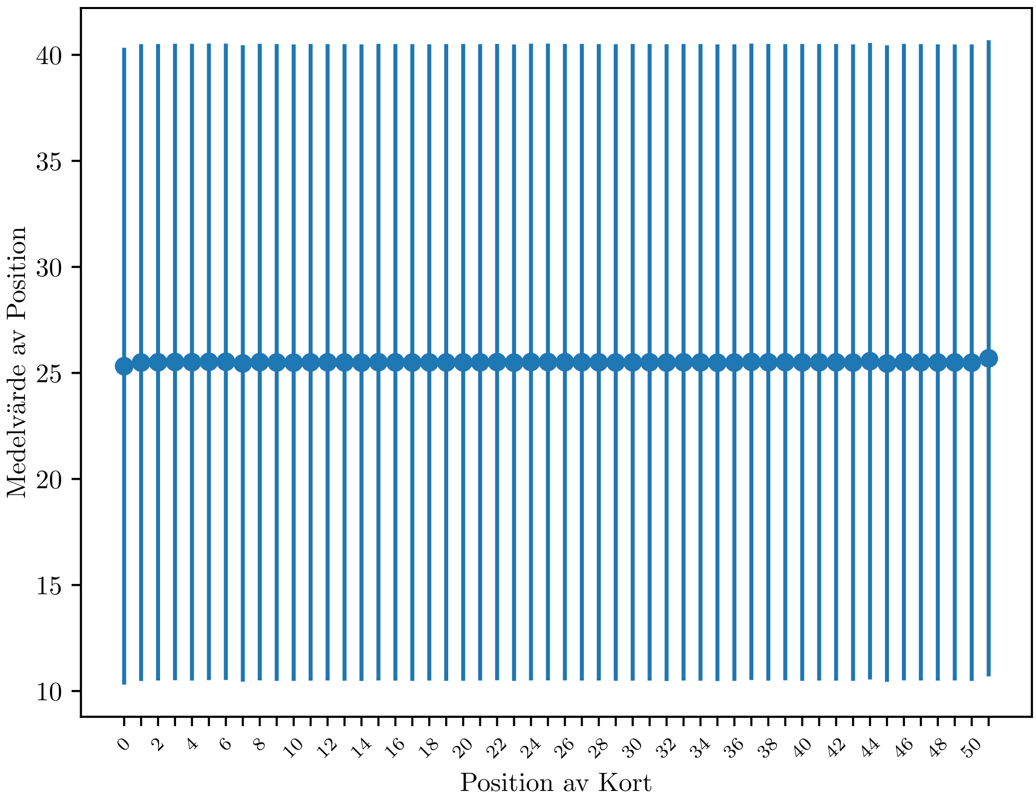
Figur 18: Resultatet från STDMean testet för SOC Pile Shuffle med **4** iterationer.



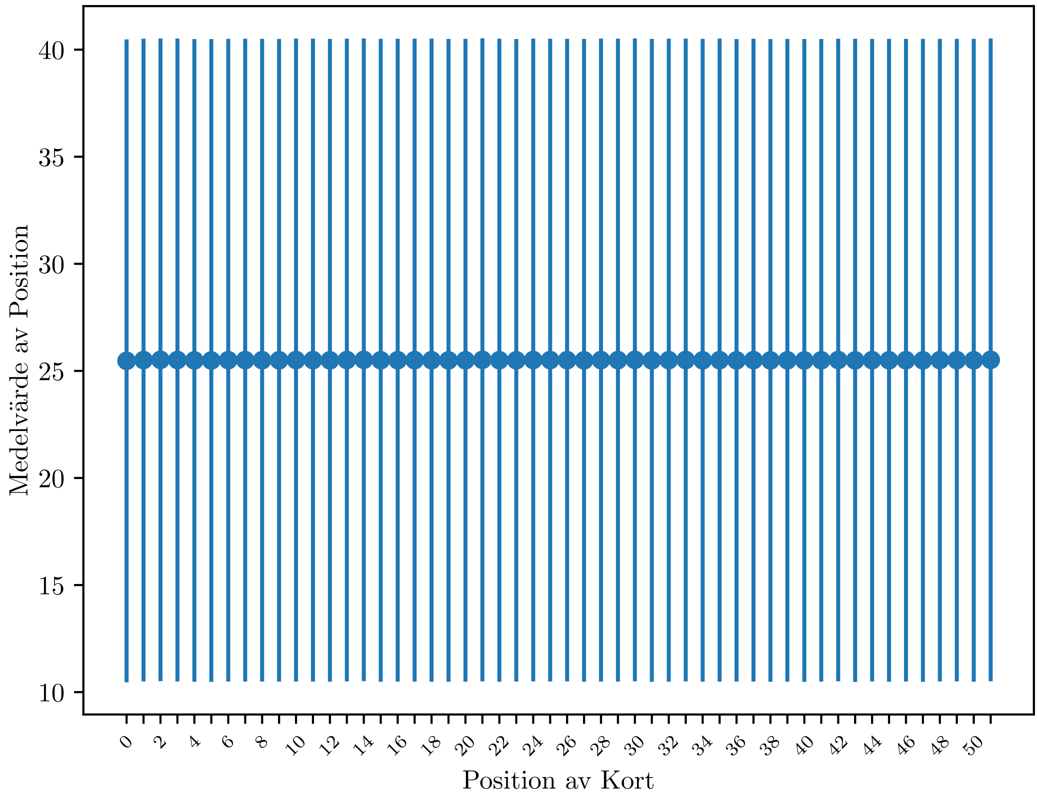
Figur 19: Resultatet från STDMean testet för Ten Pile Shuffle med **1** iteration.



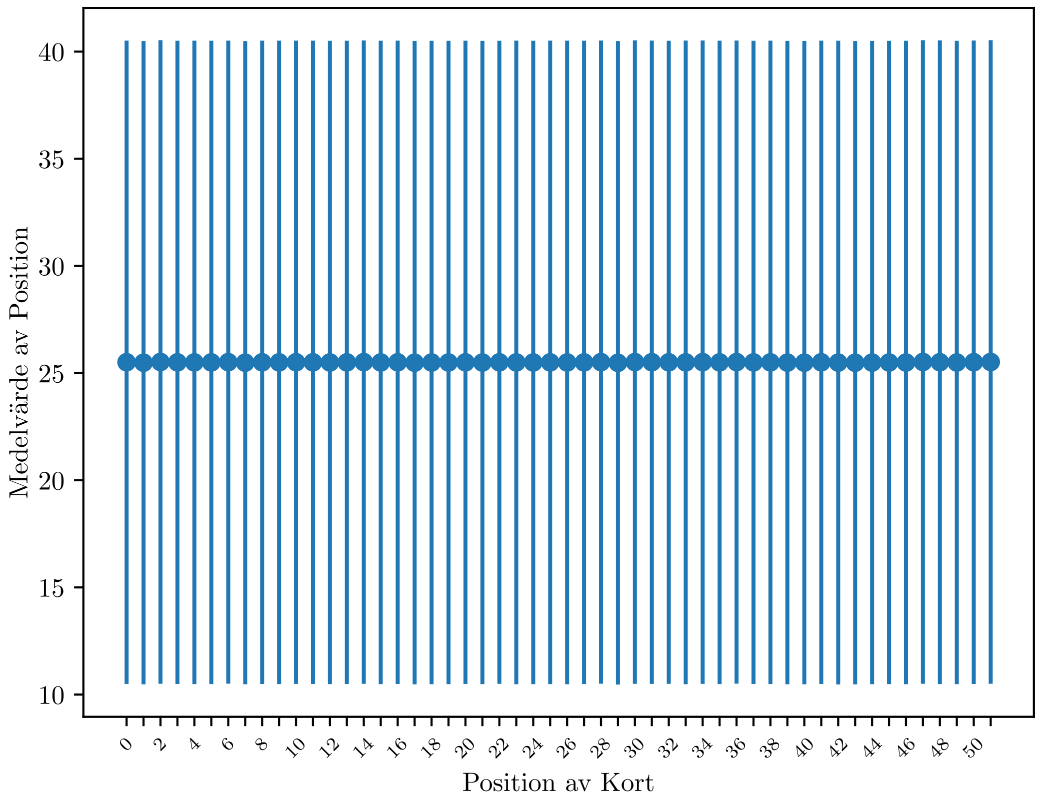
Figur 20: Resultatet från STDMean testet för Ten Pile Shuffle med **2** iterationer.



Figur 21: Resultatet från STDMean testet för Ten Pile Shuffle med **3** iterationer.



Figur 22: Resultatet från STDMean testet för Ten Pile Shuffle med **4** iterationer.



Figur 23: Resultatet från STDMean testet för Wheel Fisher-Yates shuffle med **1** iteration.