

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 L^AT_EX.
Kompilerad 7 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

Glossary	2
1 Inledning	3
1.1 Introduktion till ämnet	3
1.2 Syfte	3
2 Teori	3
2.1 Beräkningsteori	3
2.2 Slumpmässighet	3
2.3 Bakgrund av blandings metoder	3
2.3.1 Riffle shuffle	3
2.3.2 Pile shuffle	4
2.3.3 Inblick i en professionell kortleksblandare	5
2.4 Klassiskt poker test	5
2.4.1 Chi-två-test	5
2.5 Standardavvikelse- och medelvärdestest (STDMean) testet	5
2.6 Slumptalsgeneratorer (RNG)	6
2.7 Programmerings verktyg	6
3 Metod	6
3.1 Testmiljö	7
3.2 Framtagande av blandningsalgoritmerna	7
3.2.1 Anpassning av Riffle Shuffle	7
3.2.2 Anpassning av Pile Shuffle	8
3.2.3 Anpassning av design av professionell kortleksblandare	9
3.3 Simulering av kortblandningsprocesser	10
3.4 Statistisk analys av insamlad data.	11
3.4.1 Implementation av den klassiska poker testet	11
3.4.2 Implementation av STDMean testet	12
4 Resultat	12
5 Diskussion	13
5.1 Den mest passande blandningsmetoden	14
5.1.1 Preliminär gallring	14
5.1.2 Jämförelse av blandningsmetoderna	14
5.2 Felkällor och Källkritik	14
5.3 Slutsats	14
Bibliografi	14
Bilagor	15
A Källkod	15
B Kod för GSR Riffle Shuffle	16
C Kod för Pile Shuffle	16
D Kod för Wheel Fisher-Yates shuffle	16
E Resultat av STDMean test	16

Ordlista

***p*-värde** Den uppmätta sannolikheten att nollhypotesen är sann

ApEn Approximate entropy

df Frihetsgrader (Degrees of freedom)

GSR Gilberts-Shanons-Reeds modell

matrix Data i två dimensioner som en tabell

MP Medelvärde av Position

ns Nanosekunder

PK Position av Kort

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

SOC Shuffle-O-Matic

STDMean Normalfördelning och medelvärde test

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 Beräkningsteori

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.

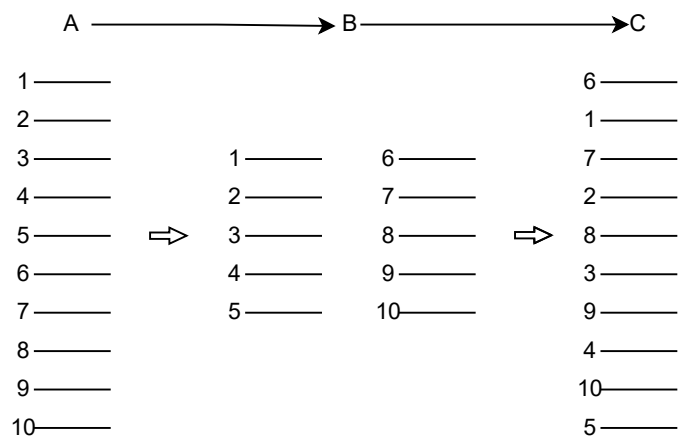
2.2 Slumpmässighet

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).

2.3 Bakgrund av blandings metoder

2.3.1 Riffle shuffle

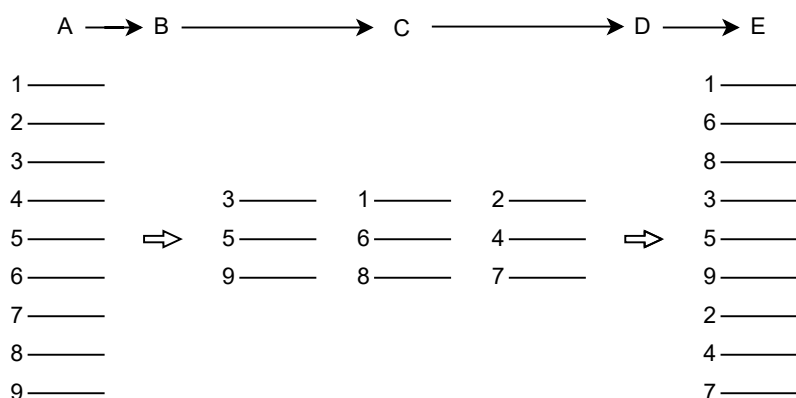
Riffle Shuffle är den mest kända kortleksblandningsmetoden. Riffle shuffle har undersökts över långt tidsperiod. Den matematiska modellen utvecklades av Gilbert och Shannon 1955 och senare oberoende av Reeds 1981 (Diaconis, 2003, s. 77–79).



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

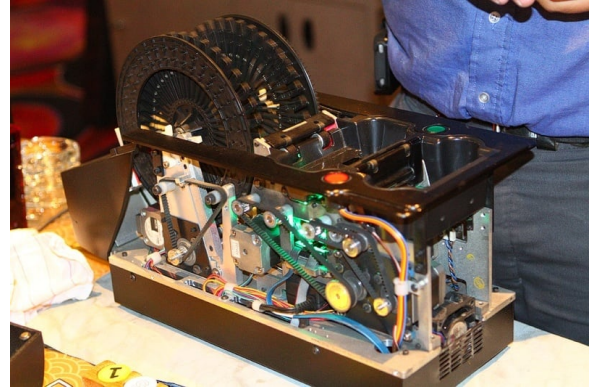
Pile shuffle är en kortleksblandningsmetod som genomförs med fysiska kort. Processen utgår från att ett kort från kortleken läggs i en av flera olika högar. Processen försätter tills alla kort från kortleken har flyttats till en av högarna. Sedan läggs alla högar ihop i en ny kortlek. Flöde av en sådan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



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

2.3.3 Inblick i en professionell kortleksblandare

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 literärforsknings 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 definierad 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 Klassiskt poker test

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 sannolikheten för en pokerhand i är p_i , därför borde antalet mönster i den kategori vara

$$e_i = p_i * n$$

Där o_i är antalet mönster som borde matcha pokerhand i . För att bestäma om resultaten kan komma från en rättvis kortlek jämförs de resultat som fått 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 chitvåfördelning. I testet jämförs 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 (National Institute of Standards and Technology, 2023). Det kritiska värdet fås av antalet frihetsgrader (kategorier - parametrar) och signifikansnivån som bestämts för testet.

Ett aspekt i chi-två-test är att data mängderna som testas inte kan vara för små. Det rekommenderas att O_i inte är mindre än 5.

2.5 Standardavvikelse- och medelvärdestest (STDMean) testet

Datamängden är utformat på sätt att kort värde är alltså dess position i kortleken. Första kort är 0 dvs position 0 i kortleken. Kalkylerar varje columns korts medelvärde och standaravvikelse ska man förutspå att medelvärde

Tabell 1: Vanliga kategorier till poker test

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

borde ligga nära $51/2 = 25.5$ detta betyder att i position 0 har alla kort varit i, och om standaravvikelse i figuren är höga torn har position 0 stort variation av vilka kort som var där. Detta är då en indikator på en uniform distribution som betyder en slumpmässigt algoritm.

Förklara logiken till standaravvikelse och medelvärde i logiken med avgörande av normal distribution och hur den ger ett mått/bild på randomness

2.6 Slumptalsgeneratorer (RNG)

Här berätta om ChaCha20 (Bernstein, 2008) den som är biobliotekets rand (The Rand Project Developers, 2022) default PRNG och mersenne twister som egentligen var första hands val (Matsumoto och Nishimura, 1998) men som vi sedan bytte till ChaCha för att i rust är det relativt svårt att implementera och använda mersenne twister. Berätta om dem skillnaderna som finns.

En väldigt känd och testad algoritm är Mersenne Twister (MT 19937) som har en väldigt långt period innan siffrorna börjar att upprepa sig, mer exakt 2^{19937} (Matsumoto och Nishimura, 1998).

ltså nämn att för att det här är som et proof of concept, borde vid senare tillfälle användas en hårdvara PRNG och icke software PRNG. Men på grund av detta undersökning handlar mesta dels om stora mängder av data simulationer så måste vi använda och Sacrifice”verkliga till att lättare skapa simulationer!

2.7 Programmerings verktyg

Inom ramen för denna studie är användningen av programmeringsverktyg som Python och Rust av stor betydelse. Detta sektion inkluderar beskrivningar av alla verktyg som ska användas. Tyngden betonas på den tidigare forsknings yttersts utmärkta arbete inom datanalis verktyg som gjorts 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 utförandet.

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 fundamentala datatyper och funktioner (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 programmeringsspråk som fokuserar på minnessäkerhet, parallellism samt minnes effektivitet. 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 ökat populariteten av användning av Rust i inbyggda system som ett motkandidat till C (Sharma m. fl., 2023).

Rand crate (The Rand Project Developers, 2022).

Rayon crate (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 med denna sektion är att djupgående 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. Genom att generera ett bestämt mängd av kortleksblandningar, där datamängder uppfyller kravet ifrån statistiska metoder. Dessa statistiska analysmetoder omfattar det klassiska pokertestet (2.4), med resultat given av chi-två-testet (2.4.1). Samt STDMean testet (2.5), 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 implementation, programmet som användes för simulation, samt implementationen av analysmetoderna på Github se Bilaga A.

3.1 Testmiljö

I valet av testmiljö prioriterades operativsystemets kompatibilitet 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 systemsresurser, vilket är avgörande för att uppnå följdriktiga och tillförlitliga testresultat. Därför utfördes undersökningen på en dator med Linux som operativsystem. Se Tabell 2 för testmiljöns 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 systemresurser (för detaljer se avsnitt 2.7). Därmed att i ett fysiskt maskin skulle blandningsalgoritmen 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 kortblandningsmaskin 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 implementation av ChaCha20 (se sektion 2.6 för detaljer). Som tidigare nämnts behöver ChaCha20 mer systemresurser 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 trovärdiga kortblandningar som är mindre influenserade av dess underliggande PRNG implementation.

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 betecknas 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 lämpligt kandidat till den potentiella kortleksblandaren.

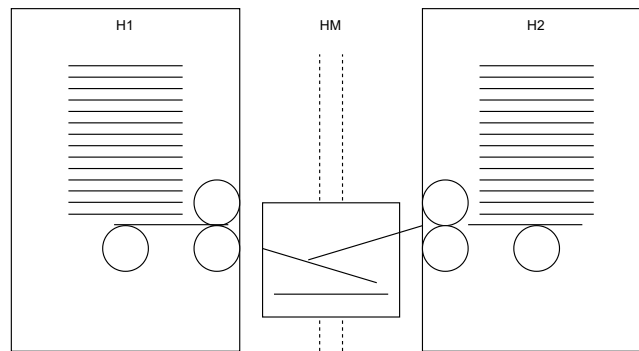
Matematiska formeln för GSR distrubution är anpassad på rad 6 se Algoritm 1. För att inte distrahera med tekniska utmaningar implementation i Rust finns i Bilaga B.

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 ordning)
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  $r \leq (\text{längden av } H1)/(\text{längden av } H1 + \text{längden av } H2)$  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 illustrativt blandningsprocess 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.

Potentiellt fysikt implementation: Anpassnings 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 $H2$. Komplikeringar i detta design kan inkludera timing problem d.v.s. att kortet från $H1$ träffar kortet från $H2$ på vägen till HM .

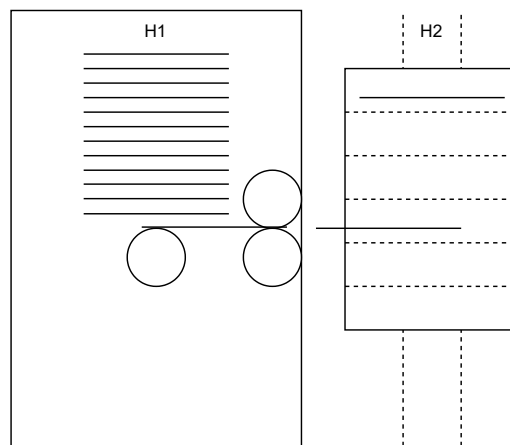
3.2.2 Anpassning av Pile Shuffle

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. Inspiration 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 varierande 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**.

Algorithm 2 Pile Shuffle pseudokod

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

```
1:  $n \leftarrow num$  ▷ Antal fack
2:  $M \leftarrow num$  ▷ Max antal kort per facket
3:  $H2 \leftarrow$  lista av  $n$  tomma listor
4: for  $kort$  i  $H1$  do
5:   loop
6:      $sf \leftarrow$  slumpmässigt fack index mellan 0 och  $(n - 1)$  ▷ slumpmässigt fack (sf)
7:     if längden av  $H2[sf] < M$  then
8:       Lägg  $kort$  i  $H2[sf]$ 
9:       break
10:    end if
11:  end loop
12: end for
13:  $H1 \leftarrow$  sammanfoga alla facken i  $H2$  ▷ Samla ihop alla fack till en kortlek
```



Figur 5: Ett illustrativt diagram av kortblandningsprocessen 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.

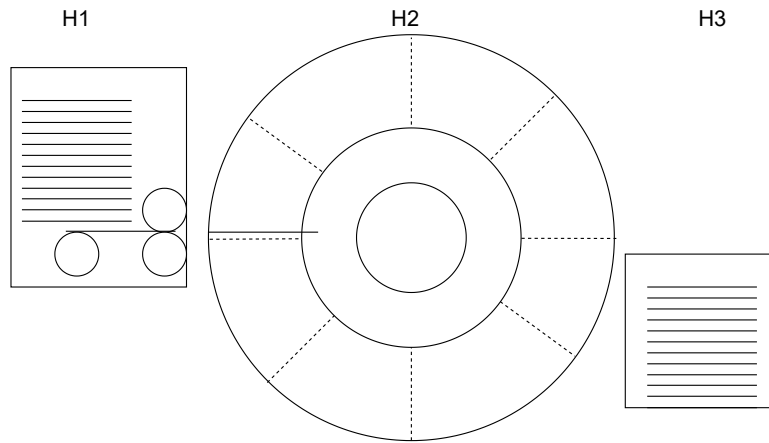
3.2.3 Anpassning av design av professionell kortleksblandare

Tredje blandningsmetoden som skapades var inspererad 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.

Algorithm 3 Wheel Fisher-Yates Shuffle pseudokod

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

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



Figur 6: Ett illustrativt diagram av kortblandningsprocessen 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 scenario simulerades vart ett helt ny öppenad 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 jokrar). 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 rekommendationen för chi-två-testet. Där NIST rådgör 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 kortdelningar kan approximeras 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 datamängden), och där x är en av talen ur den tidigare definierade 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 experimintell metodik valdes det att utföra 15 iterationer per algoritm. En iteration definieras till att kortleken blandas successiv på föregående 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 tidigare nämndes utfördes simulationen parallellt där varje algoritm exekverades på egen tråd, oberoende av varandra. Simulationen utfördes i följande steg: 1) Algoritmerna 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 avslutad iteration i medelvärde av ledtiden kalkylerades och sparades i en csv fil; 8) Resultaterande blandade kortlekarna utplattades 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 användning vid statistisk analys (se sektion 2.7).

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 tidsbegränsningen. Istället fokus ligger på en annan branch av slumpmässighet baserad på sannolikhetslära.

3.4.1 Implementation av den klassiska poker testet

Syfte med denna metod är att utföra preliminär galring av algoritmernas iterationer som har betydande 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 addopterades 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 slumpgeneratorer. Motiveringen till komplexiteten var att i denna studie utforskas särskilt kortlekar till spel vart kombinationer av kort ofta spelar betydande roll och därför var det särskilt viktigt att anpassa slumpmässighets analysmetoderna till hur den potentiella kortleksblandare skulle fungera i drift.

Tabell 3: Namn på alla pokerhänder och dess relativ mönster. Antal₁ är teoretisk framkommande kombinationer med en standard kortlek. Antal₂ är faktorerade 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

Kategoriseringsprocessen till att få ett hand till ett pokerhand börjades med att det utnyttjades NumPy vektoriserad aritmetik funktionalitet. Med denna funktionen på ett effektivt sätt valdes ut 5 kort i.e en hand och kördes igenom en kategoriserings 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 kategoriserings funktionen rad mässigt.

kategoriserings 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 definierade mängden av kort x , i.e. (0, 1, 2, ..., 51) till kategoriserbart kort utfördes. För att optimera kategoriseringsprocessen användes Numba för att skapa en ny tråd, detta kunde enbart göras p.g.a. att det användes den fundamentala 8bit datatypen, som definierades tidigare. Efter att ett ny tråd skapades processen försattes på följande sätt:

$$\text{valör}(x) = x \mod 13$$

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

där $\text{valör}(x)$ ger värde från 0-12 och $\text{färg}(x)$ 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 datamängden 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 $\text{Chi2.ppf}(1 - \alpha, df)$. 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 Implementation 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 särskilt viktigt för att i kortspel, där positioner på första korterna spelar betydande roll i kortspels rättvisshet. Snedvridningen kan alltså potentiell indikera logiska fel i implementationen 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 fel 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 utnyttjade NumPy vektoriserad aritmetik i symbios med dens inbyggda funktioner som medelvärde och standardavvikelse. Både funktionerna utfördes kolumn vis på m antal rader. Denna resultatet användes sedan till att rita punktdiagram av Medelvärde av Position på y-led (MP) och Position av Kort på x-led (PK). Standardavvikelse från medelvärdet visas som symmetrisk linje ifrån denna punkten. Den teoretiska medelvärdet i en slumpmässigt blandning borde ligga runt $51 \div 2 = 25.5$ och den experimentellt testade standardavvikelse ifrån medelvärdet ≈ 15 .

4 Resultat

Totalt 75 simulationer av olika kortblandnings algoritmer 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. Den viktigaste av simulationerna plockades ut och deras resultat presenteras här. Resultat av poker test samt medelvärde för algoritmers ledtid presenteras i tabeller. Resultaten för medelvärde av dem positioner ett kort har varit på och standardavvikelsen presenteras via punktdiagram. 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 linjen som går genom punkten. I Bilaga E Figur 7

Tabell 4: Resultatet från det klassiska pokertestet: Gränsvärde fastställt 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				(b) Six Pile Shuffle			
Iteration	χ^2	p -värde	Ledtid [ns]	Iteration	χ^2	p -värde	Ledtid [ns]
3	233941.28	0	1681	1	3854.97	0	1116
4	7077.30	0	1687	2	18.36	0.031	1124
5	449.67	3.38	1660	3	12.27	0.20	1131
6	34.46	7.42	1667	4	5.94	0.75	1184
7	6.03	0.74	1654	5	12.34	0.19	1167
8	8.50	0.48	1642	6	8.89	0.45	1168
9	19.64	0.020	1582	7	4.33	0.89	1211
10	8.88	0.45	1495				
11	16.19	0.063	1432				

(c) SOC Pile Shuffle				(d) Ten Pile Shuffle			
Iteration	χ^2	p -värde	Ledtid [ns]	Iteration	χ^2	p -värde	Ledtid [ns]
1	3349.24	0	1450	1	3349.24	0	1587
2	7.02	0.64	1465	2	5.83	0.76	1654
3	19.11	0.024	1518	3	4.83	0.85	1716
4	7.53	0.58	1441	4	5.38	0.80	1625
5	8.50	0.48	1501	5	5.09	0.82	1697
6	12.34	0.19	1536	6	8.80	0.46	1698
7	8.36	0.50	1577	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 tankeprocessen lättare att följa. Den första subsektionen handlar om hur den mest passade blandningsmetoden togs fram, den här sektionen är updelad i två delar. Den preliminära gallringen är där de blandningsmetoder som är självklart ompassande gallras bort från undersökningen. I jämförelsen av blandningsmetoderna kommer vi närmare undersöka de metoder som återstår efter gallringen för att komma fram till vilken av dem som passar bäst in på de krav som ställdes i syftet 1.2. Felkällor och källkritik nämns efter att den mest passande blandningsmetoden tagit fram för att diskutera deras påverkan på de tidigare slutsaterna i diskussionen. I slutsatsen binds presenteras den blandningsmetod som undersökningen kom fram 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 än dem andra i dem färdigheterna som blandningsmetoderna jämförs på. Alla blandningsmetoder som var signifikanta enligt chi-två-testet kan gallras då dem inte uppnår en av kraven för blandningsmetoder, dem blandar inte på ett helt slumpmässigt sätt.

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).

Bernstein, D. J. (jan. 2008). "ChaCha, a variant of Salsa20". I: URL: <https://cr.yp.to/papers.html#chacha> (hämtad 2023-12-06).

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.

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>.

Matsumoto, Makoto och Takuji Nishimura (1998). "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator". I: *ACM Trans. Model. Comput. Simul.* 8.1, s. 3–30. ISSN: 1049-3301. DOI: 10.1145/272991.272995. URL: <https://doi.org/10.1145/272991.272995>.

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).

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 2024-12-09).

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.

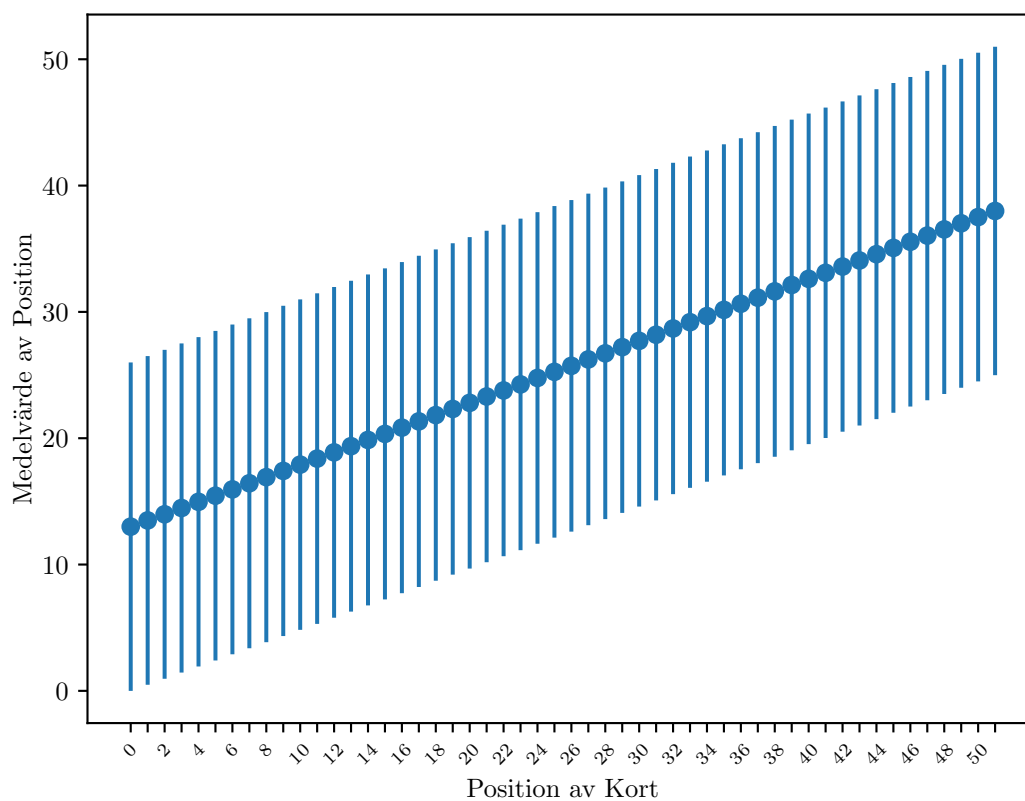
Yan, Kevin (16 aug. 2016). *1-8 decks Casino Full-Automatic card shuffler*. URL: <https://www.youtube.com/watch?v=txl3gqIfwHM> (hämtad 2024-02-06).

Bilagor

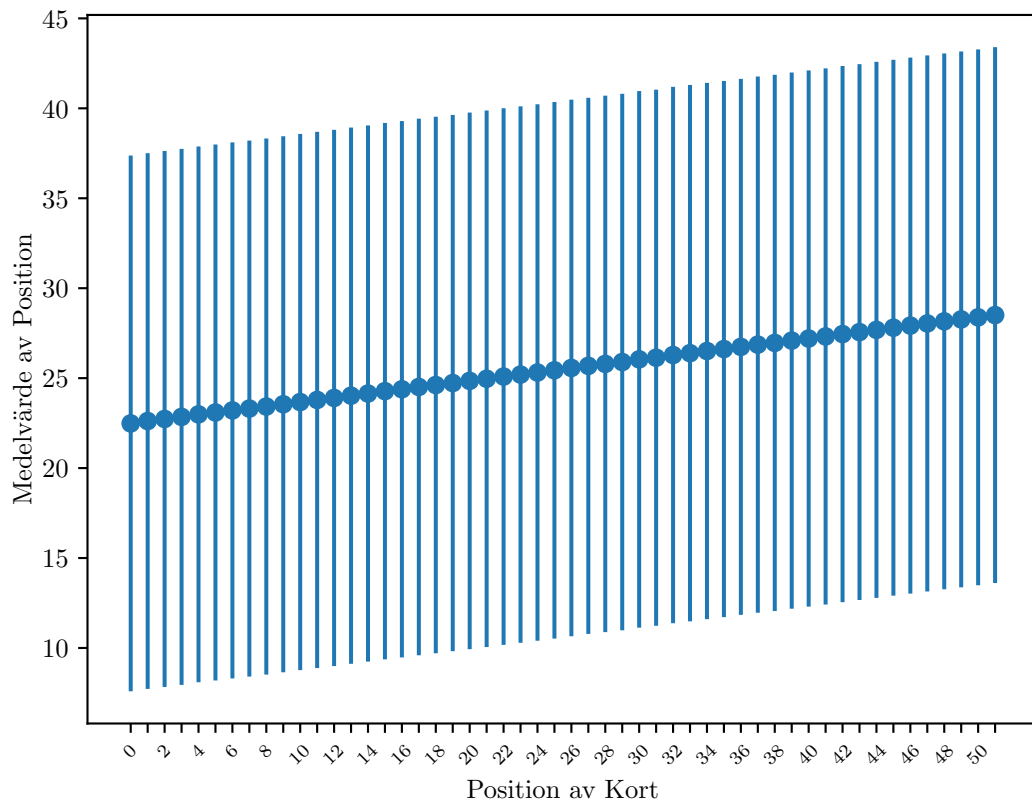
A Källkod

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

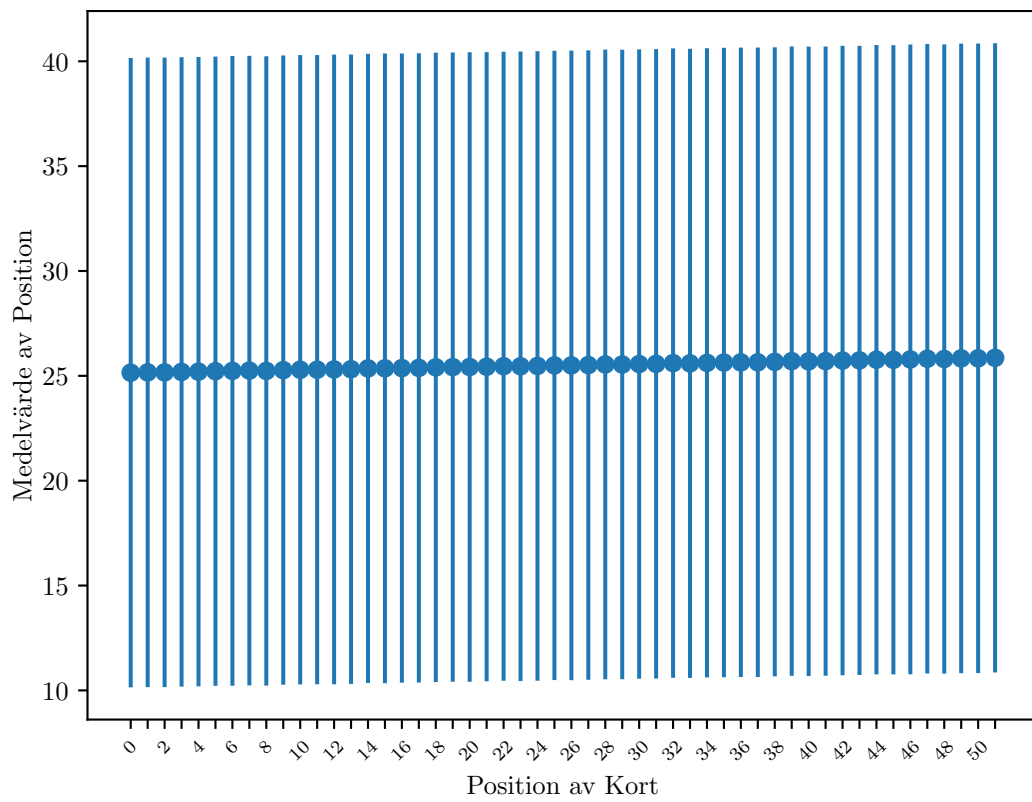
- B Kod för GSR Riffle Shuffle
- C Kod för Pile Shuffle
- D Kod för Wheel Fisher-Yates shuffle
- E Resultat av STDMean test



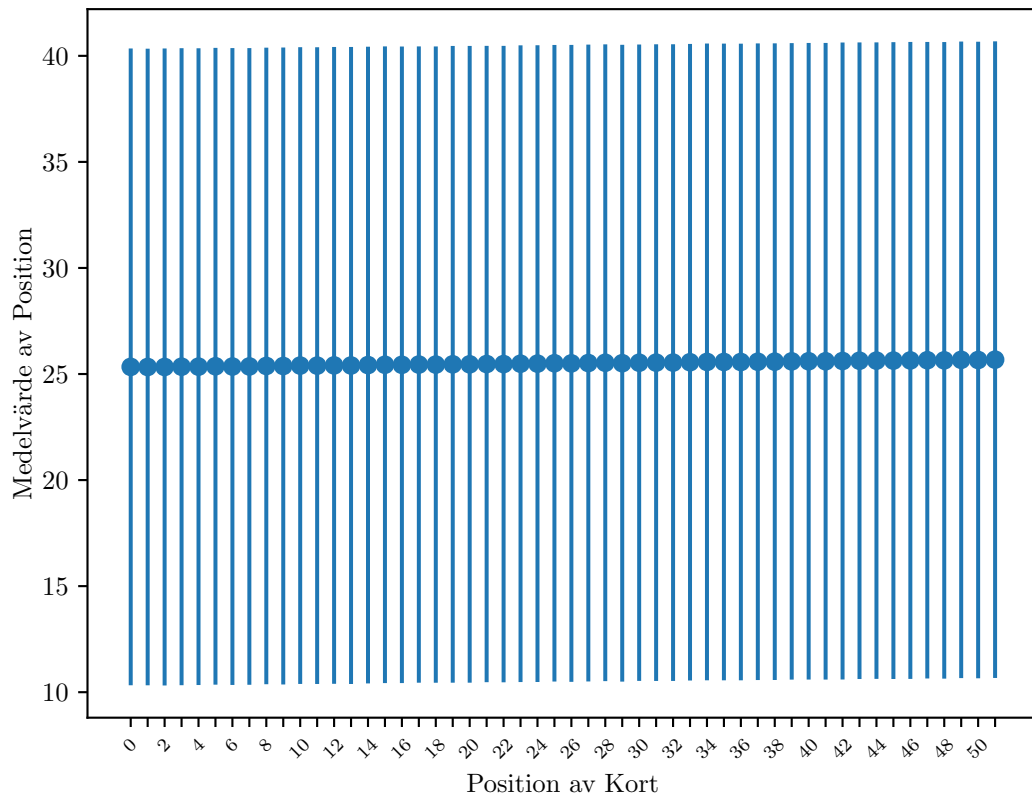
Figur 7: Resultatet från STDMean testet för GSR Riffle Shuffle med 1 iteration.



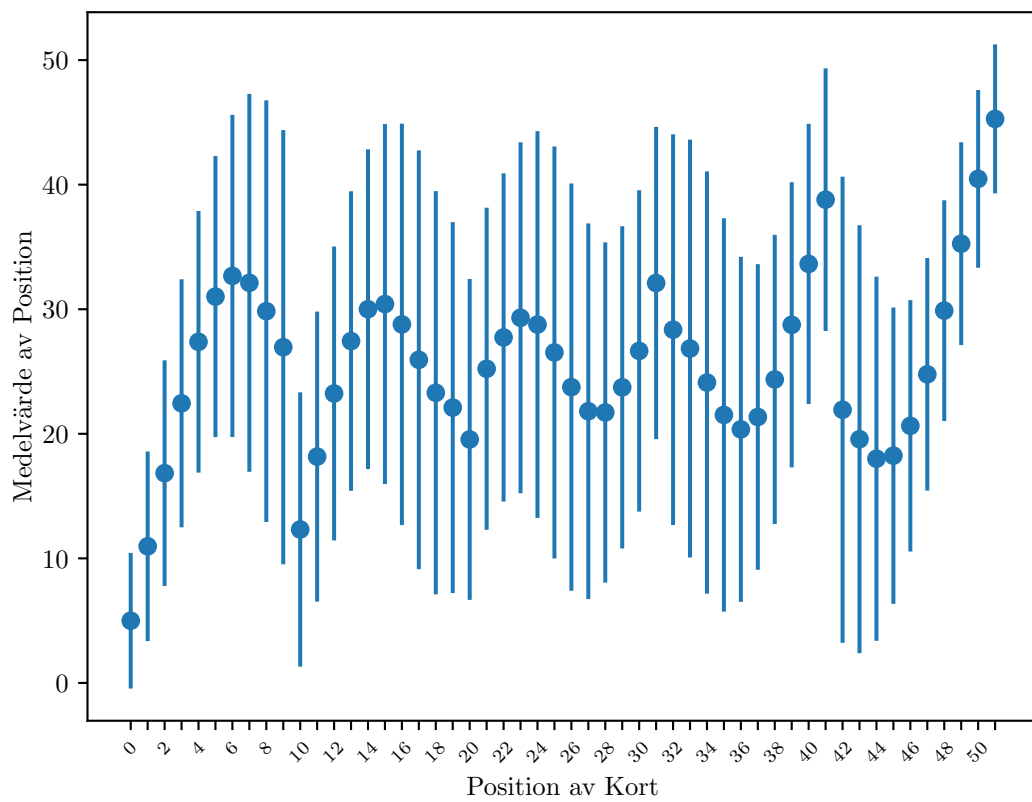
Figur 8: Resultatet från STDMean testet för GSR Riffle Shuffle med **3** iterationer.



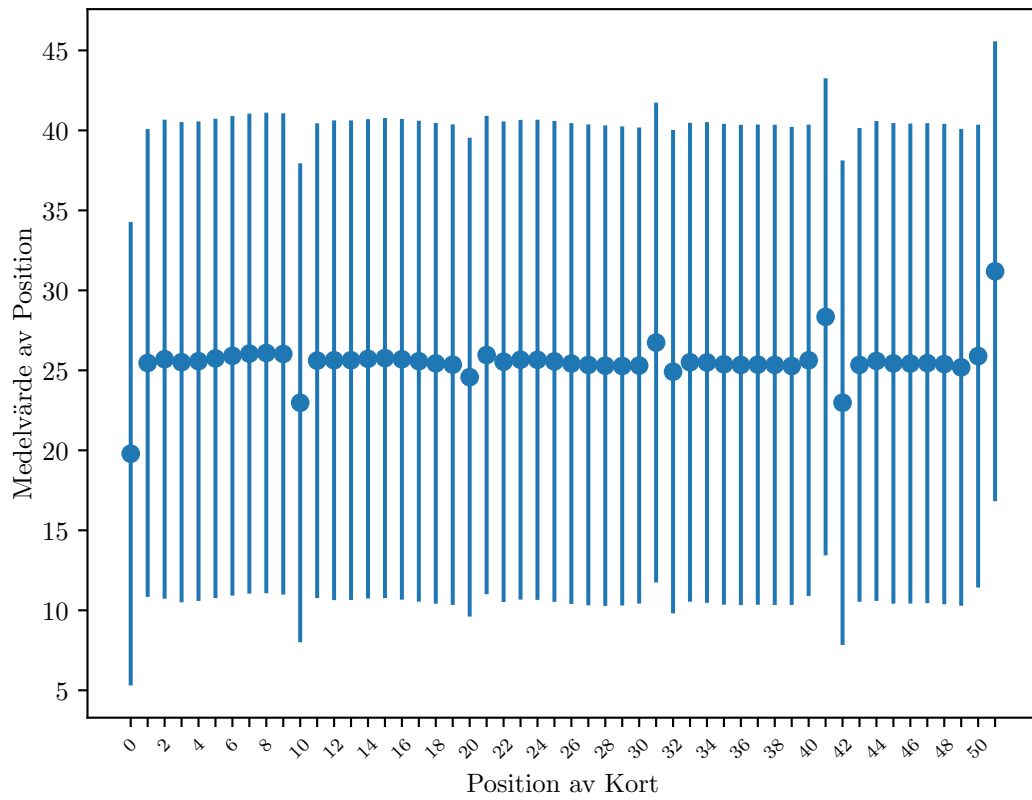
Figur 9: Resultatet från STDMean testet för GSR Riffle Shuffle med **6** iterationer.



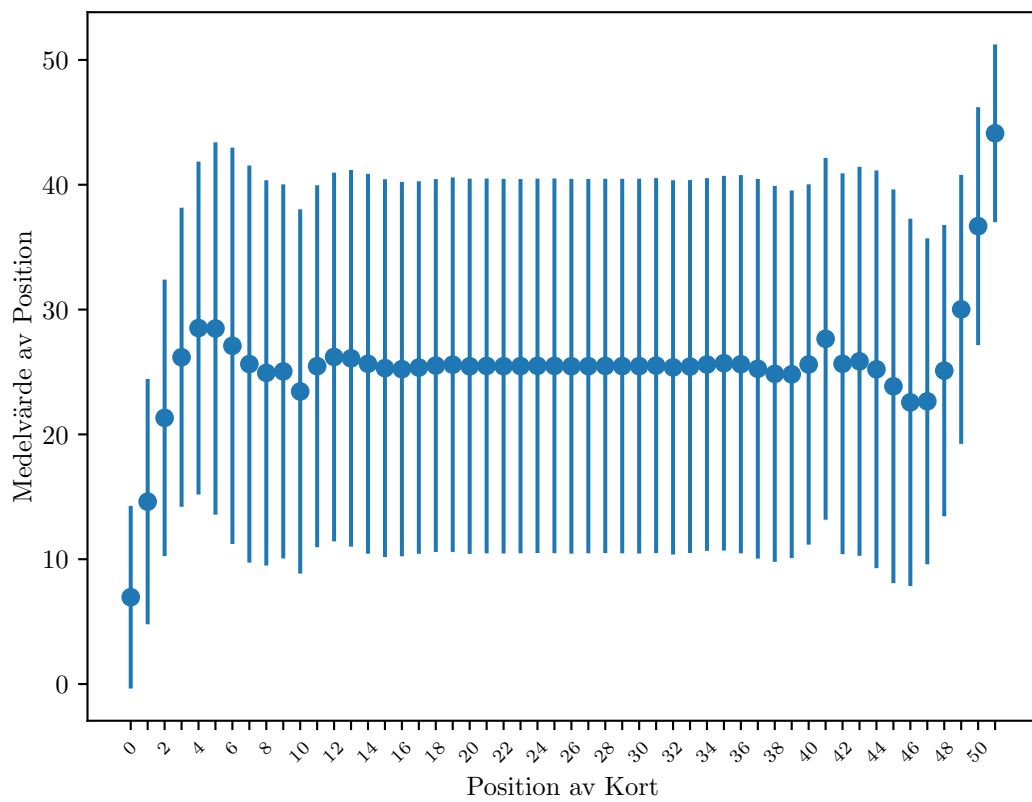
Figur 10: Resultatet från STDMean testet för GSR Riffle Shuffle med **7** iterationer.



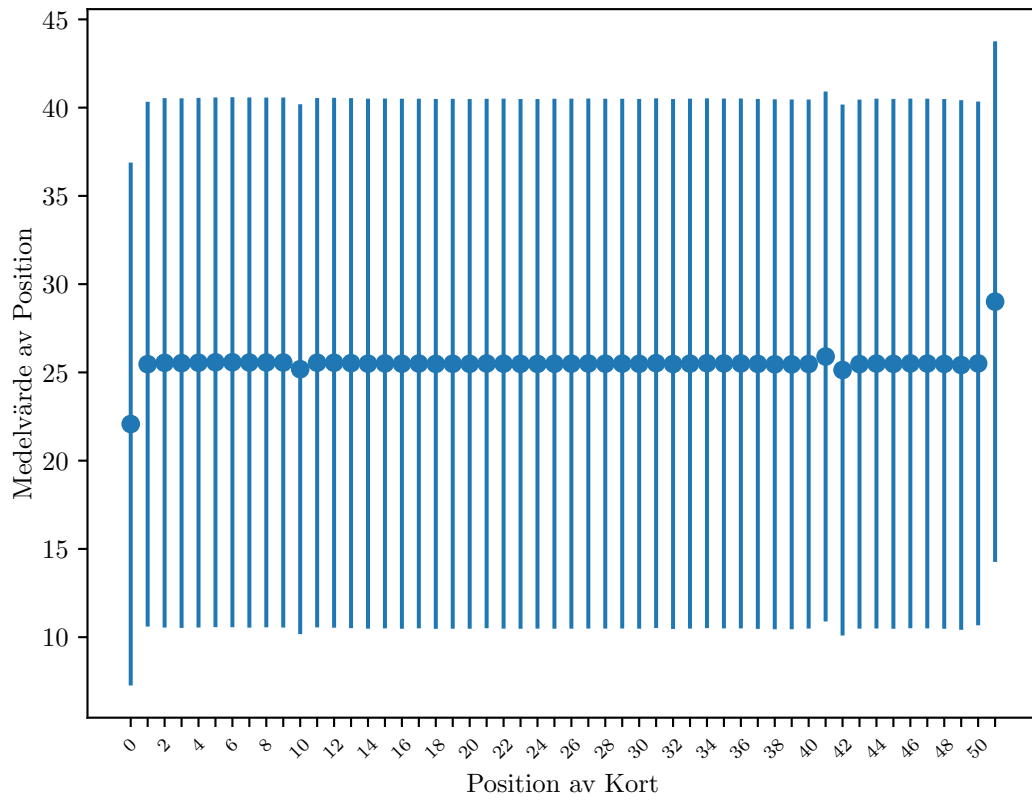
Figur 11: Resultatet från STDMean testet för Six Pile Shuffle med **1** iterationer.



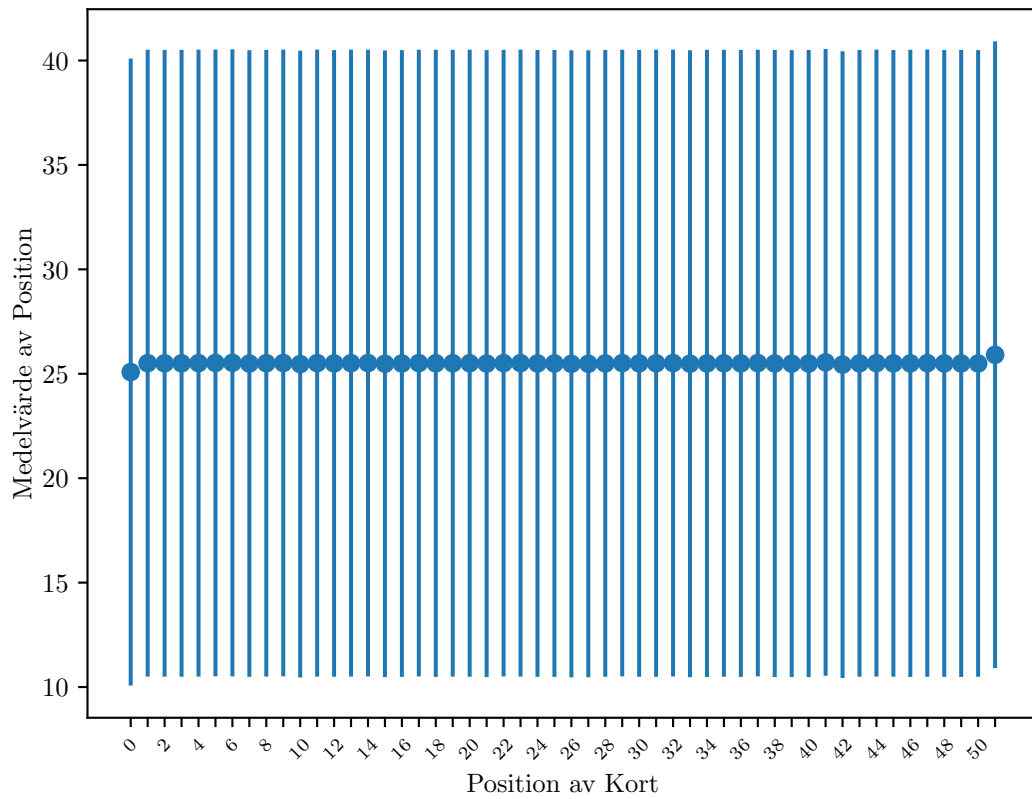
Figur 12: Resultatet från STDMean testet för Six Pile Shuffle med **2** iterationer.



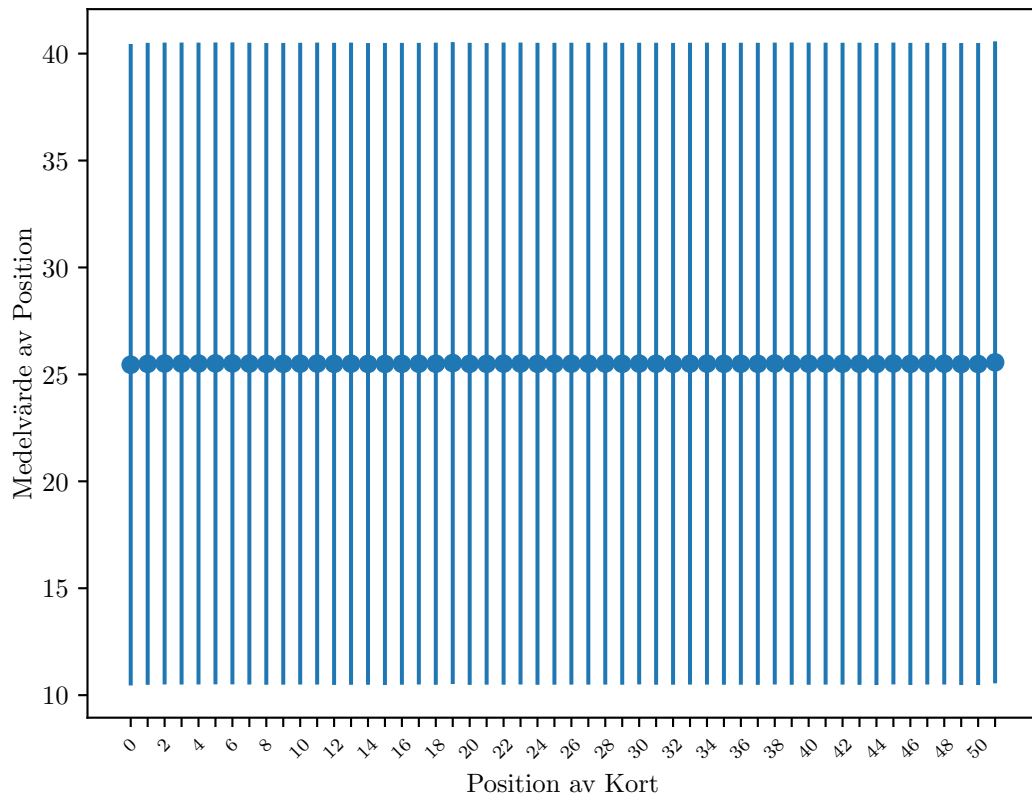
Figur 13: Resultatet från STDMean testet för SOC Pile Shuffle med **1** iteration.



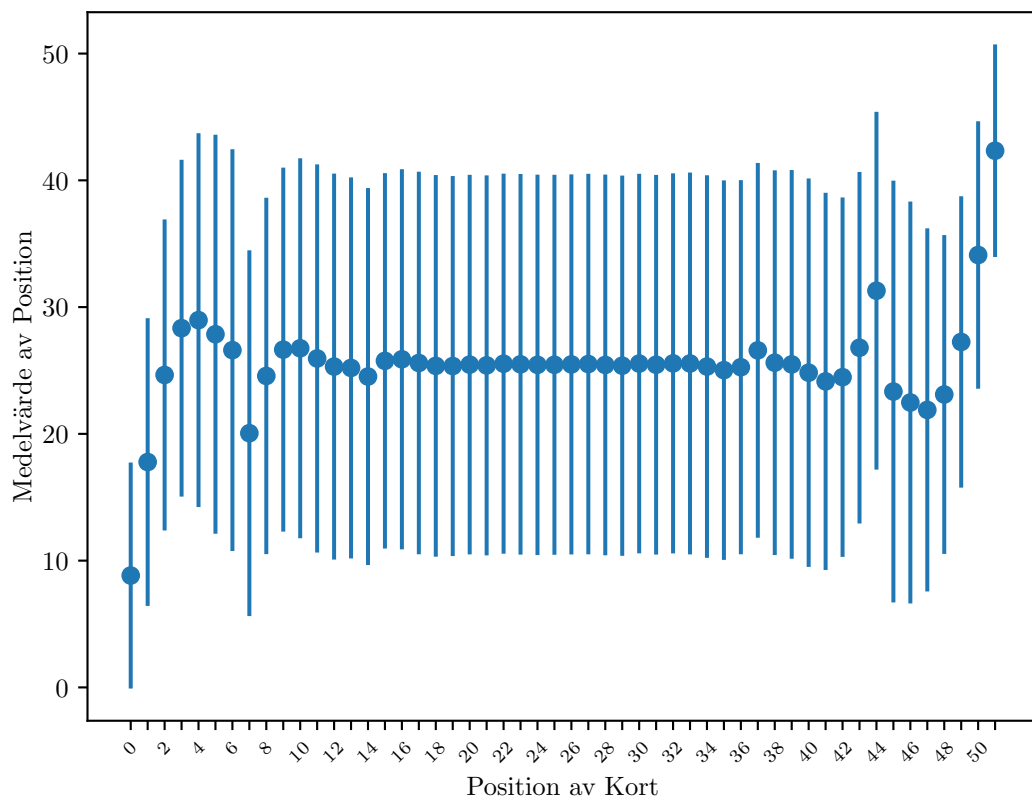
Figur 14: Resultatet från STDMean testet för SOC Pile Shuffle med **2** iterationer.



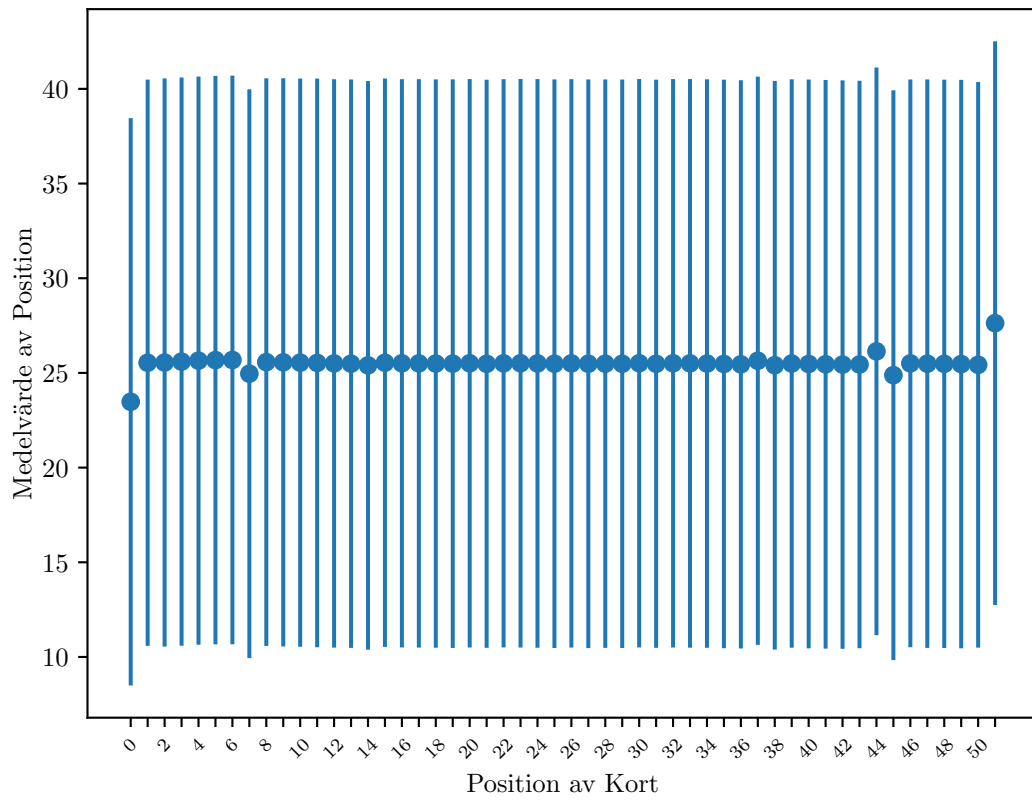
Figur 15: Resultatet från STDMean testet för SOC Pile Shuffle med **3** iterationer.



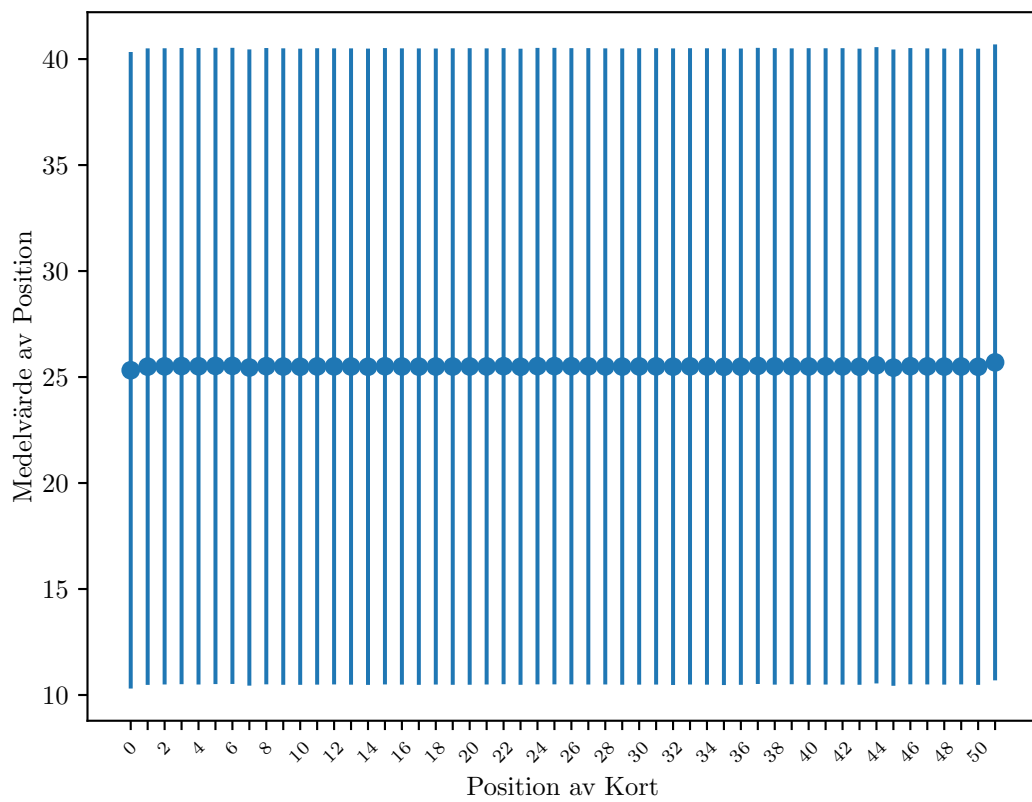
Figur 16: Resultatet från STDMean testet för SOC Pile Shuffle med 4 iterationer.



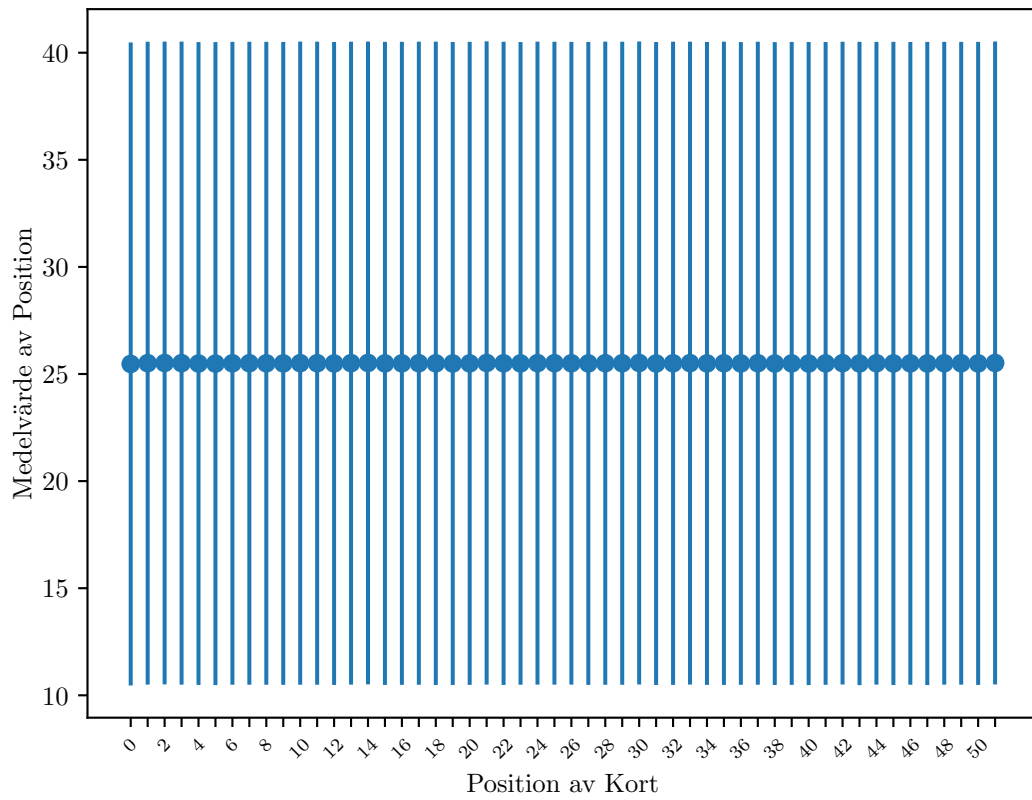
Figur 17: Resultatet från STDMean testet för Ten Pile Shuffle med 1 iteration.



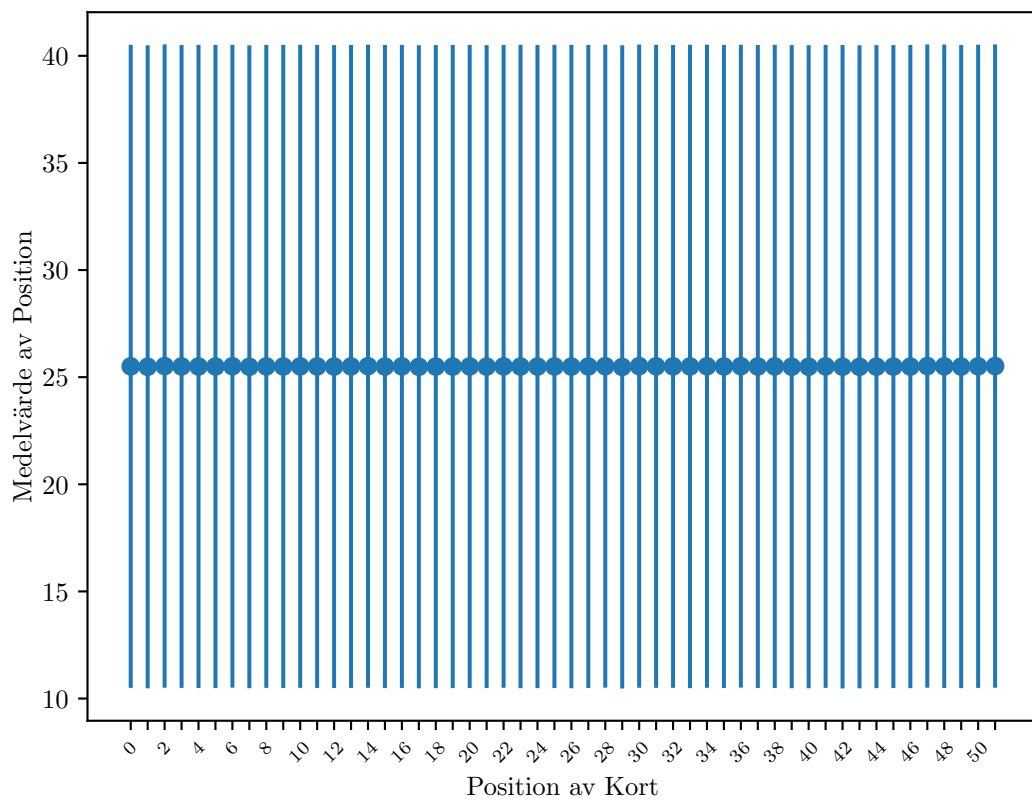
Figur 18: Resultatet från STDMean testet för Ten Pile Shuffle med **2** iterationer.



Figur 19: Resultatet från STDMean testet för Ten Pile Shuffle med **3** iterationer.



Figur 20: Resultatet från STDMean testet för Ten Pile Shuffle med 4 iterationer.



Figur 21: Resultatet från STDMean testet för Wheel Fisher-Yates shuffle med 1 iteration.