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

En ny potentiell automatiskt kortleksblandare

Proof of Concept av en kortblandare

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 3 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

df Frihetsgrader (Degrees of freedom)

GSR Gilberts Shanons och Reeds modell

matris 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/processer, alltså att kalkylera.

2.2 Slumpmässighet

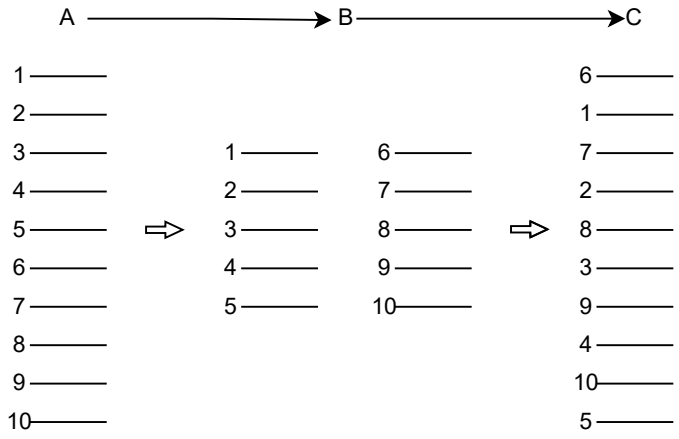
Slumpmässighet är en egenskap som anger att något sker utan klart mönster. När något är helt slumpmässigt är det i princip omöjligt att förutse. Slumpmässighet inom matematik är byggd beräkningsteori och den delas upp i två beroende på om det gäller slumpmässighet av en bestämd mängd eller en oändlig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

2.3 Bakgrund av blandings metoder

Theoretical Whats, Whys and Hows angående algorithmer.

2.3.1 Riffle shuffle

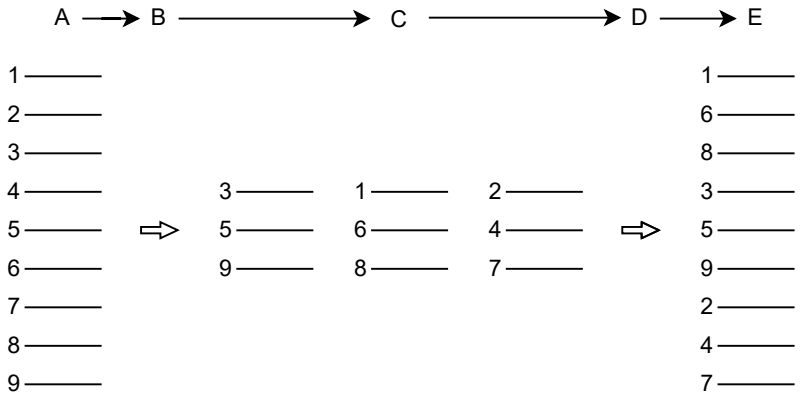
Det finns många olika metoder att blanda kort med riffle shuffle. Undersökningar har genomförts och dem har kommit fram till att



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.



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

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.

Tabell 1: Vanliga kategorier till poker test

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

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

$$e_i = p_i * n$$

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 (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 en 5.

2.5 standardavvikelse- och medelvärdestestet (STDMean) testet

Förklara logiken till stanardavvikelse och medelvärdet i logiken med avgörande av normal distribitution och hur den ger ett mått/bild på randomness

2.6 Pseudoslumptalsgenerator (PRNG)

Här berätta om ChaCha20 (Bernstein, 2008) den som är biobliotekes rand (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).

Alltså 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

Python är en objektorienterad, dynamiskt typad programmerings språk med fokus på läsbarhet med en enkelt syntax (Van Rossum och Drake, 2009). Detta som tillför att det finns stora mängder av biblioteker (samling av färdig kod). Python används ofta i statiskt och numeriskt analys på grund av dens läsbarhet och lätt

användning. På grund av Python är skriven i programmeringspråket C finns det också tilgännliga C API's med andra betecknat CPython. Dem gör det möjligt att skapa program som är snabba och effektiva i stora datamängd analys. Dessa egenskaper har medföljt början av skapande av tiotals avancerade numeriska och bibliotek som är snabba i utförande och minskar antal radar av kod som behövs att skrivas i färdigt program. Dem bibliotek som är speciellt intressenta i detta undersökning är NumPy, Matplotlib, SciPy och Numba. NumPy som är snabbt i vektoriserad aritmetik och SciPy som har många inbygga statistiska tester som χ^2-testet (Harris m. fl., 2020; Virtanen m. fl., 2020).

Rust är ett programmeringsspråk som fokuserar på minnessäkerhet, parallellism och minneseffektivitet. 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) (Matsakis och Klock II, 2014). Detta har ökad populäritet i användning av Rust i inbyggda system som ett motkandidat till c/c++.

3 Metod

Metoden för denna studie kan indelas i tre huvudområden 1) Framtagande av blandningsalgoritmerna. 2) Simulering av kortblandningsprocesser. 3) Statistisk analys av insamlad data.

Inledningsvis kommer att framtagande av blandningsalgoritmerna fokusera på utvecklingen av specifika kortblandningsmetoderna. 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 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 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 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 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 2.7). 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 2.6 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

Som det diskuterades tidigare är Riffle Shuffle den mest kända kortblandingsmetoden. I denna studie anpassades GSR modelen (för matematiken bakom modellen se sektion 2.3.1). Den valdes för att utforska hur väl den presterar i kortleksblandingar. Och om den skulle vara en lämligt kandidat för den potentiella kortleksblandaren.

Potentiell fysikt design

B

3.2.2 Anpassning av Pile Shuffle

SOC Pile Shuffle: Inspiration att undersöka pile shuffle kom ifrån 3DprintedLife (2021), där pile shuffle implementerades i en 3d printat kortleksblandare.

Six Pile Shuffle:

Ten Pile Shuffle:

3.2.3 Anpassning av design av profesionell kortleksblandare

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. Togs logiska valet att alla simulationer ska ha ett och samma utgångspunkt. 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 ordningen spelar roll.

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

För att bestämma värdet för m följdes ett kriterium ifrån klassiskt poker test (mer i detalj beskriven i underrubriken 2.4.1). Till hjälps användes NIST internätkälla i denna nämns det att det är viktigt för chi-square approximationens trovärdighet att minsta kategorin ska inte vara mindre än fem teoretiska framkommande i den kategorin (National Institute of Standards and Technology, 2023). Därför är det viktigt att hitta den minsta kategorin som kan förekomma. I poker är den mest sällsynta pokerhanden den kungliga färgstegen (Royal Flush) som kan räknas ut på följande sätt antal av alla kortkombinationer med fem kort. $\binom{52}{5} = 2\,598\,960$ Kungliga färgstege finns det fyra kombinationer av i kortleken som kan räknas ut på följande sätt totala antal kombinationer av kungliga färgstege delat med totala antal femkorts kombinationer från standardkortlek.

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

$$m = P(\text{Kunglig färgstege}) \times 5 = 3\,248\,700$$

detta blir den minimala datamängden i simulationen .

3.3.1 Val av datatyp & rådatalagring

Efter teoretiska beräkningar av datamängden är det viktigt att välja en lämplig datatyp att representera data i. Högsta talet som behövs är 51 definerad tidigare och betecknat med x. Detta betyder att den minsta datatypen som får användas är 8 bit eller 1 byte långa som kan innehålla följande värden

$$\{b \in \mathbb{N}, 0 \leq b \leq 255\}$$

som övernsstämmer med datamängdens högsta talet x. Detta ger möjligheten att använda den fundermantala datatypen som används i minnesadressen. Fördelen med detta är det simplifierar lagring och inmatning av datamängder i t.ex Python med NumPy.

Andra aspekten till varför datatypen är viktigt är att spara minnen för att det kommer att simuleras och sparas tiotals rådata filer varje fil kommer att väga $161MB$, detta kan räknas ut på följande sätt

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

som det är enkelt att spara data i binärt format för att 8 bits eller 1 byte är ett fundamentalt och uniformt mått i minnesadressen. Med det sagt, ingen bearbetning av data behövs och det är enkelt att ladda in denna i till exempel Python med numpy till statistiska testerna av rådata

Därför att sista teoretiska delen att bestämma är hur många iterationer per algoritm ska genomföras. Med iterationer menas hur många gånger kortleken blandas följande på varandra.

3.4 Statistisk analys av insamlad data.

Detta sektion handlar om hur statistiska metoder som Poker testet och STDMean testet implementerades för att analyser slumpmässighet av valda och simulerade algoritmerna.

Statistiska tester utfördes med Python version 3.11. Python valdes på grund av dens rika utbud av bibliotek, speciellt inom vetenskaplig databehandling och dataanalys. Därför valdes det att utföra statistika tester som Poker test och STDMean testet i Python. En till perspektiv till varför Python valdes är att analys och avgörande av slumpmässighet är fundementalt komplex process icke räknebart, men det som underlättar framtagande av resulat är visuella grafer och tabeller. I denna studie saknas det erfaranhet och kunskaper inom högre grads områden inom matematiken som statistikt analys. Detta betyder att rent matematiska modeller till avgörande av slumpmässighet som approximate entropi (ApEn) (Delgado-Bonal och Marshak, 2019). Men den valdes bort på grund av bristande kunskaper i detta området.

Med detta sagt Pythons Matplotlib bibliotek underlättar processen av graf ritanade tillsammans i en symbious relation med NumPy som har styrka i datamängd represention och vektoriserad databearbetning. Utfördes det mindre matematisk komplicerade tester med fokus på storleken av datamägderna som ger en god överblick av slumpmässighet av dem olika blandnings algoritmerna som simulerades. På avseende på ett teoretiskt perfekt blandnings resulat för samtliga metoder.

Rådata som sparades efter simulationen är laddat in som en dimensionell lista med hjälp av Numpy efter det är rådata återställt tillbaka till tvådimensionell matris för vidare bearbetning och analys med följande metoder: Klassisk Poker Test 2.4 och medelvärde av korta positioner i kortleken över hela datamängden kalkylerad såsom avvikelser av dessa positioner. Utförligt beskrivning i underrubriker.

3.4.1 Implemention av Klassiskt poker test

Poker Test utgår ifrån att man karaktäriserar typer av mönster till pokerhänder och kalkylerar den absolutvärdet χ^2 i detalj beskrevs teorin bakom χ^2 testet i sektion 2.4.

Kategorisering

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). Av skall att testerna utförs med standard kortleken,

valdes det att anpassa χ^2 testet att använda alla poker händer. Matematiken speciellt kombinatoriken att räkna ut sannolikheter och frekvensen av alla poker händer är relativt svårt uppdrag därför addopterdes det Armstrong (2006) uträkningar se tabell 3. Detta medföljde med mer komplex karakterisering av pokerhänder än när man använder χ^2 testet att till example testa slumptalsgeneratorer. Valet log med att datamängderna innehåller kortlekar till att göra simulationer mer realistiska och anpassade till hur ett riktigt kortleksblandare skulle fungera i drift. Därför valet att anpassa statistiska tester till slutmål användnings sätt verkade vara ett logiskt steg att ta.

Poker Test kan anpassas till vilken som helst sekvens av nummer 3-5 stora mängder. Men för att vår simulation använder vi samma nummer som pokerkort därför kan vi anpassa klassiska poker test var man använder alla möjliga pokerhänder av sekvens av 5 kort. Resultatet från karakterisering av händer använder man i en chi-square testet. För att göra testet rätt följde jag Engineering Statistics handbok (NIST 2012). Dvs för att avgöra hur många händer uppkom med avseende till teoretiska värdena EXPECTED vs OBSERVED värdena. sen jämför man chi-square resultatet till en CRITICAL värde om den värden som vi fick är mindre kan vi med säkerhet säga att algoritmen är slumpmässigt och vice versa.

Testet utgår i två generella steg kategorisera fem kort och seden

Tabell 3: Alla pokerhänders namn, relativt mönster och antal av kombinationer

Pokerhand	Example mönster	Antal kombinationer
Royal flush	$A\spadesuit\ K\spadesuit\ Q\spadesuit\ J\spadesuit\ 10\spadesuit$	4
Färgstege	$K\spadesuit\ Q\spadesuit\ J\spadesuit\ 10\spadesuit\ 9\spadesuit$	36
Fyrtal	$A\spadesuit\ A\heartsuit\ A\diamondsuit\ A\clubsuit\ K\spadesuit$	624
Kåk	$A\spadesuit\ A\heartsuit\ A\diamondsuit\ K\clubsuit\ K\spadesuit$	3 744
Färg	$K\spadesuit\ Q\spadesuit\ J\spadesuit, 10\spadesuit\ 8\spadesuit$	5 108
Stege	$K\spadesuit\ Q\heartsuit\ J\diamondsuit\ 10\clubsuit\ 9\spadesuit$	10 200
Triss	$A\spadesuit\ A\heartsuit\ A\diamondsuit\ K\clubsuit\ Q\spadesuit$	54 912
Två par	$A\spadesuit\ A\heartsuit\ K\diamondsuit\ K\clubsuit\ Q\spadesuit$	123 552
Ett par	$A\spadesuit\ A\heartsuit, K\diamondsuit\ Q\clubsuit\ J\spadesuit$	1 098 240
Högt kort	$A\spadesuit\ Q\heartsuit\ J\diamondsuit\ 5\clubsuit\ 4\spadesuit$	1 302 540
	Summan:	2 598 960

3.4.2 Implemention av 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 förustpå att medelvärde borde ligga nära $51/2 = 25.5$ detta betyder att i position 0 har alla kort varit i, och om standaravviklse 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.

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. Dem 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.

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

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

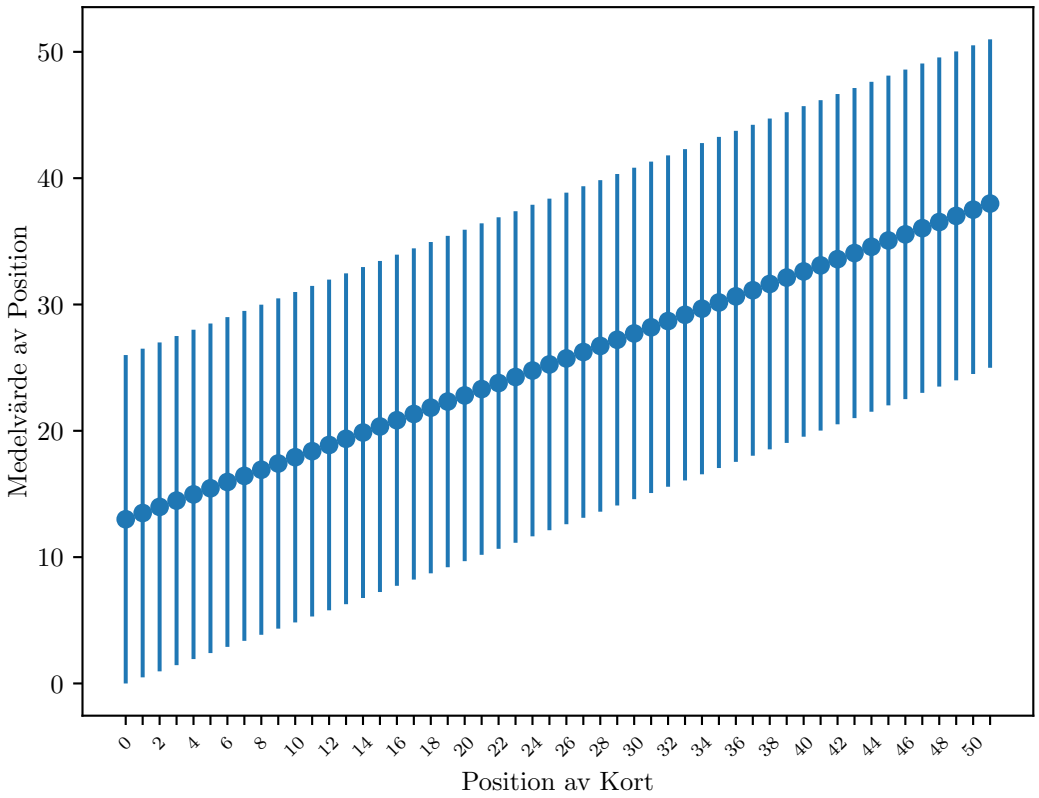
Iteration	χ^2	p-värde	Ledtid [ns]
1	3349.24	0	1450
2	7.02	0.64	1465
3	19.11	0.024	1518
4	7.53	0.58	1441
5	8.50	0.48	1501
6	12.34	0.19	1536
7	8.36	0.50	1577

(d) Ten Pile Shuffle

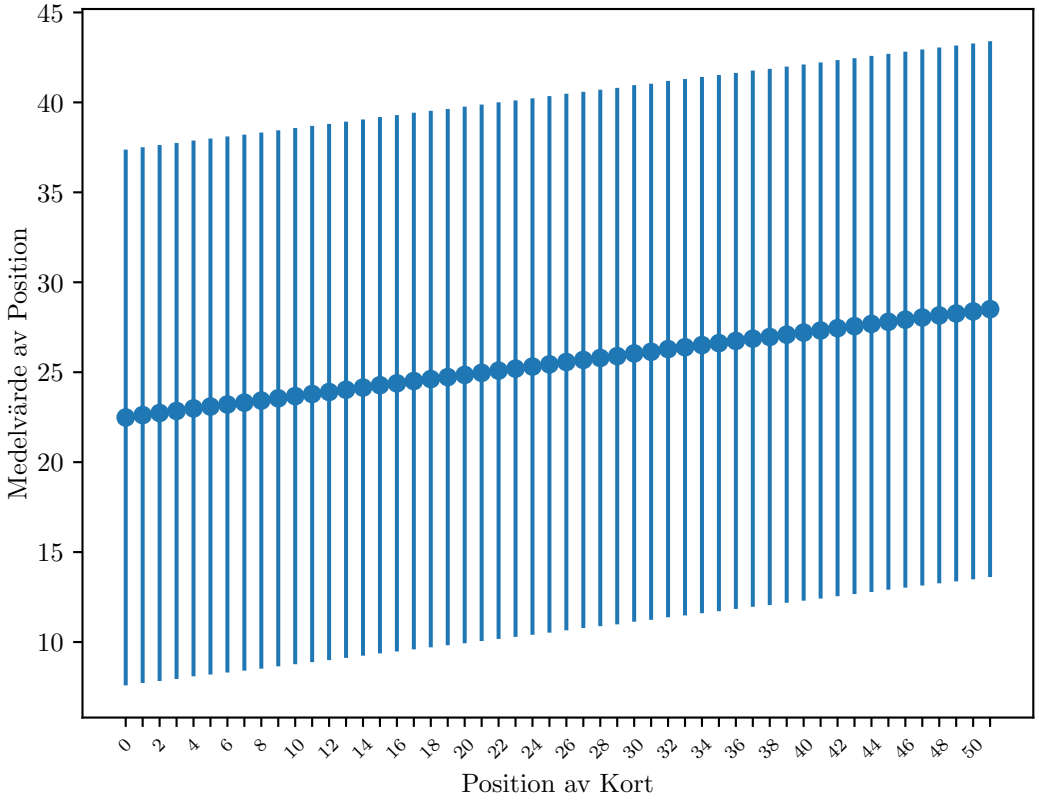
Iteration	χ^2	p-värde	Ledtid [ns]
1	3349.24	0	1587
2	5.83	0.76	1654
3	4.83	0.85	1716
4	5.38	0.80	1625
5	5.09	0.82	1697
6	8.80	0.46	1698
7	14.46	0.11	1710

(e) Wheel Fisher-Yates Shuffle

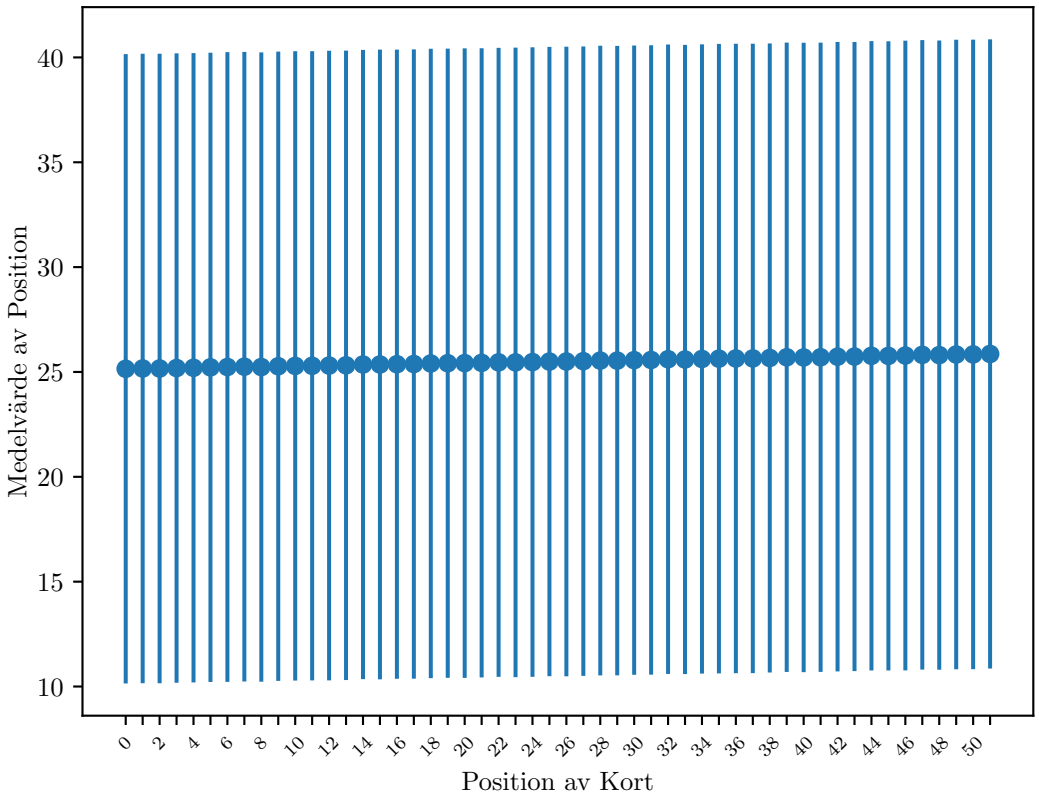
Iteration	χ^2	p-värde	Ledtid [ns]
1	10.71	0.30	715
2	13.46	0.14	720
3	6.66	0.67	728
4	9.05	0.43	733
5	8.95	0.44	772
6	6.10	0.73	794
7	9.41	0.40	778



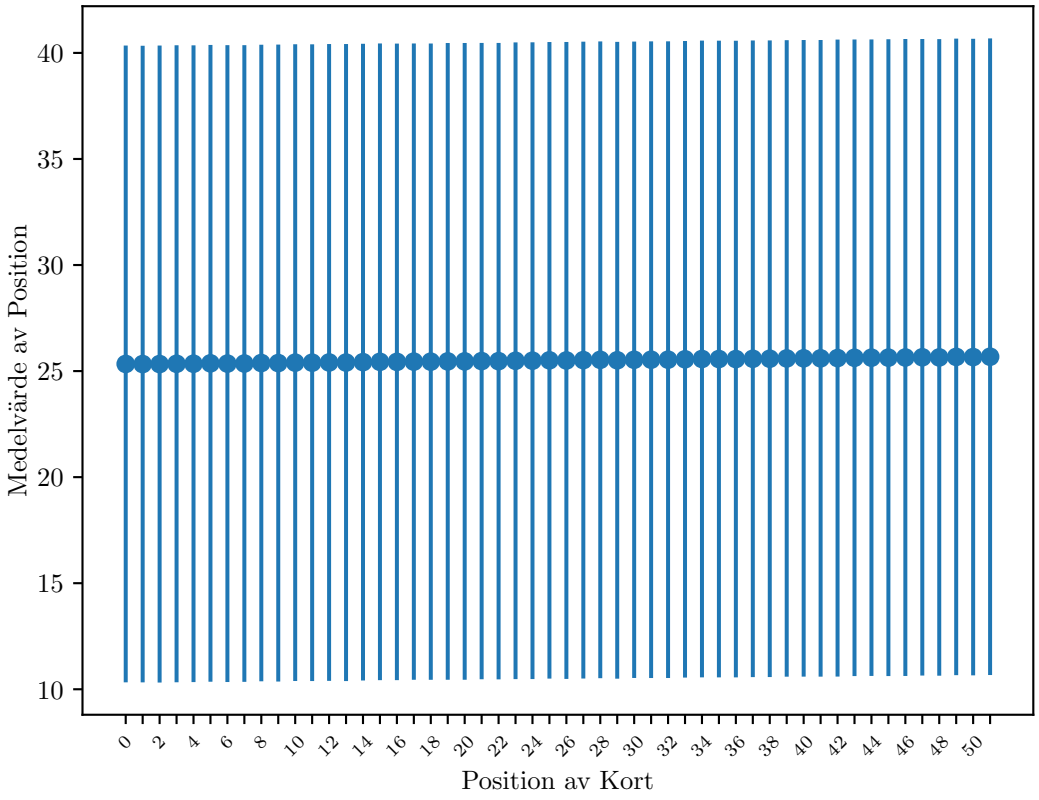
Figur 3: Resultatet från STDMean testet för GSR Riffle Shuffle med **1** iteration.



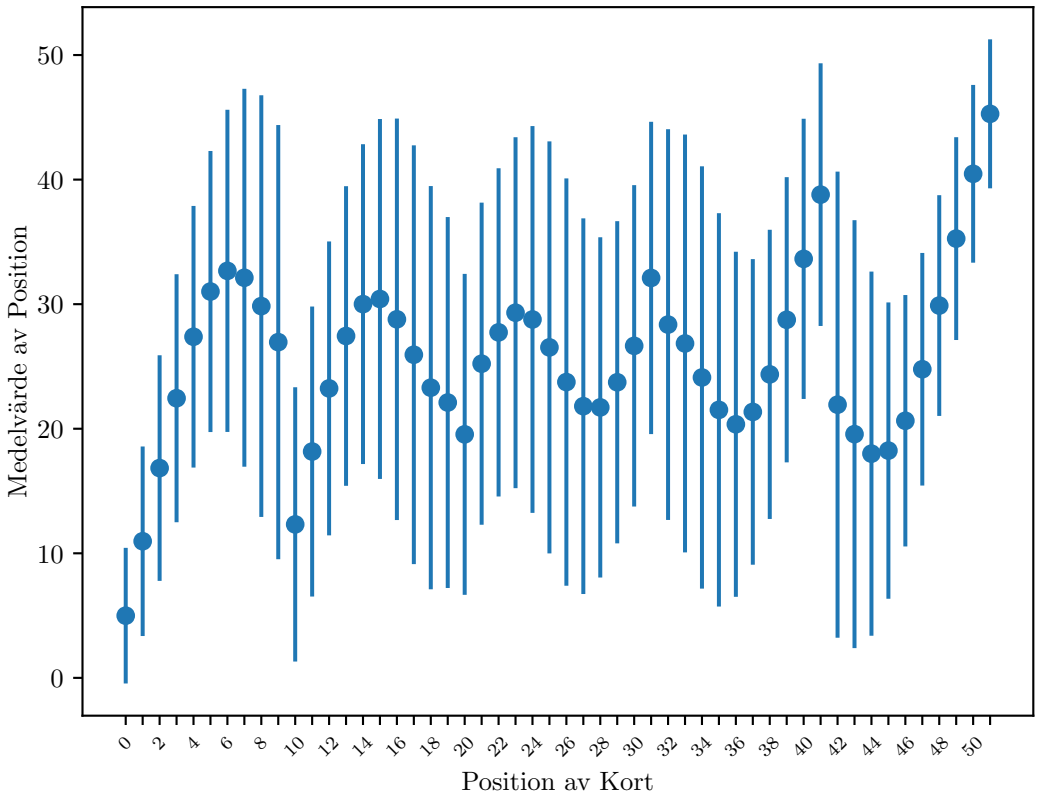
Figur 4: Resultatet från STDMean testet för GSR Riffle Shuffle med **3** iterationer.



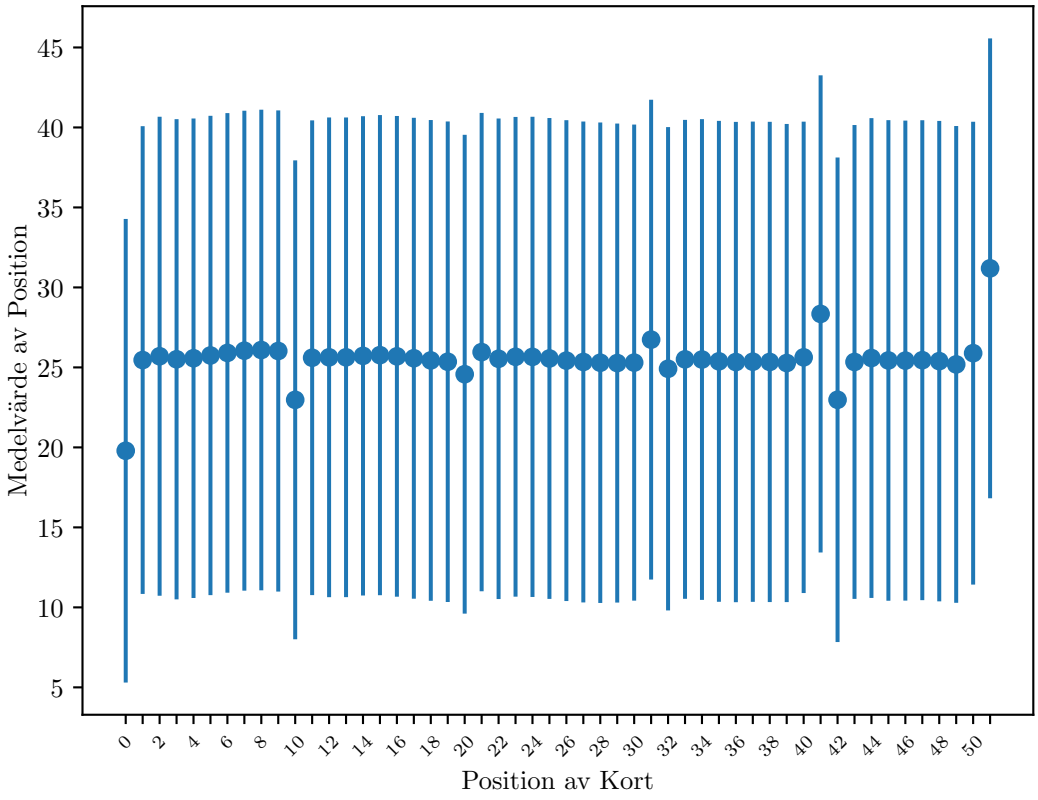
Figur 5: Resultatet från STDMean testet för GSR Riffle Shuffle med **6** iterationer.



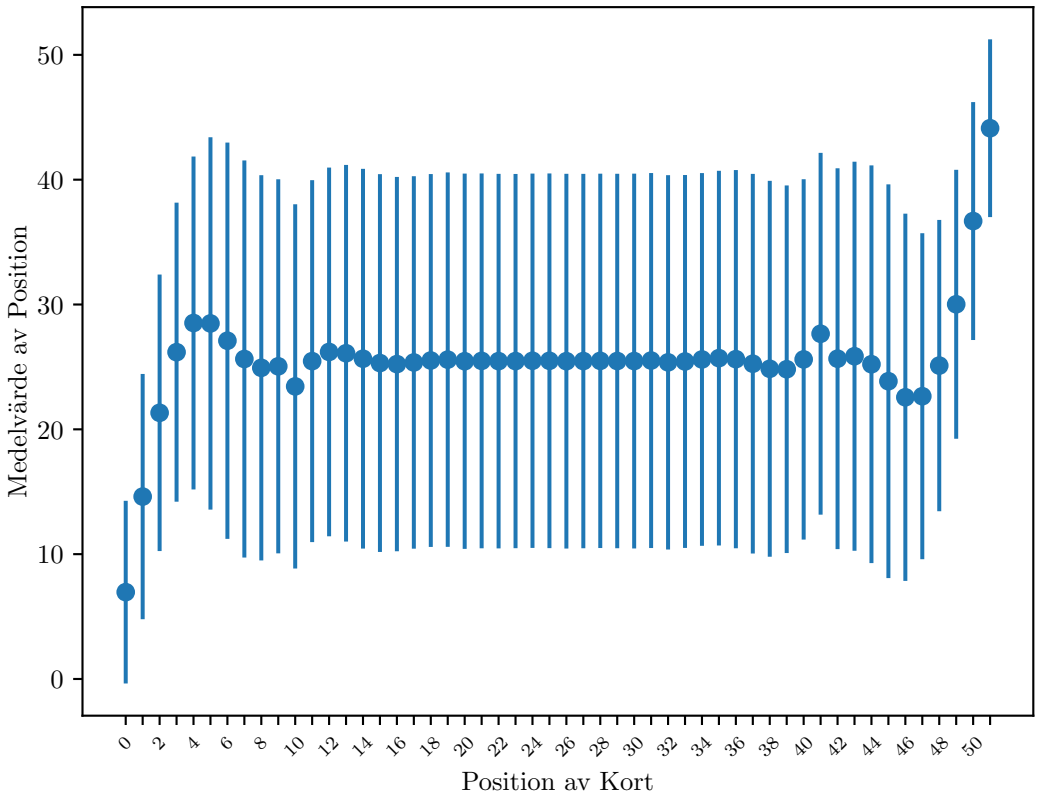
Figur 6: Resultatet från STDMean testet för GSR Riffle Shuffle med **7** iterationer.



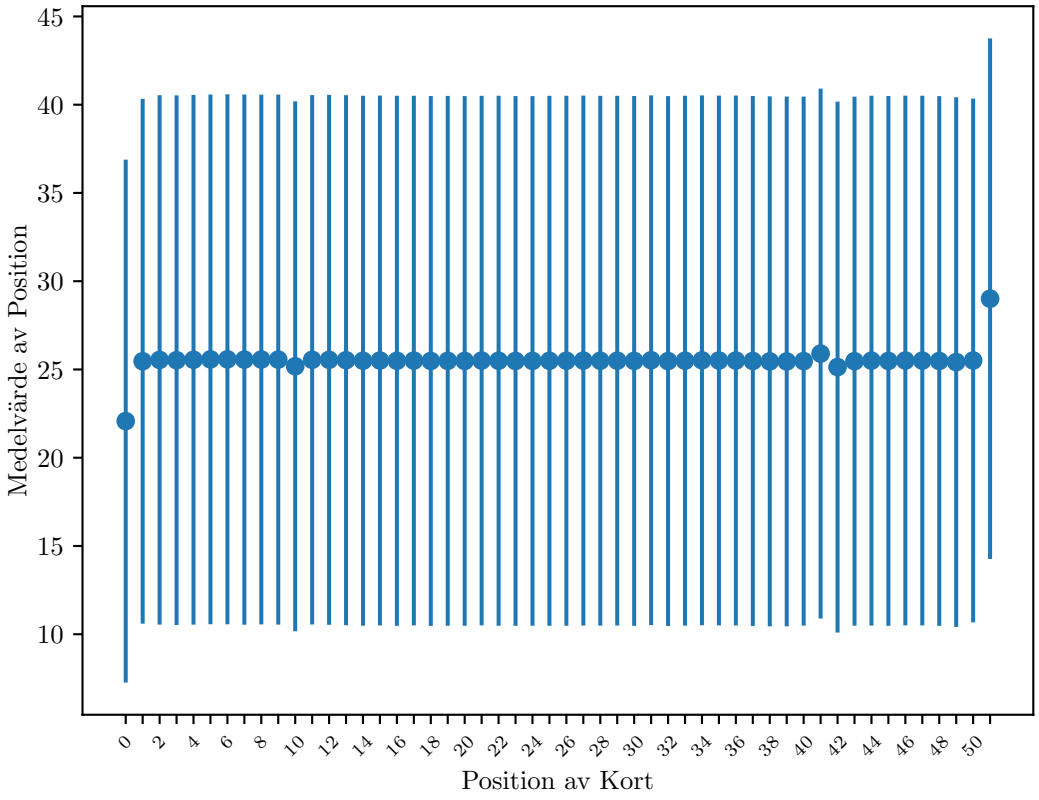
Figur 7: Resultatet från STDMean testet för Six Pile Shuffle med **1** iterationer.



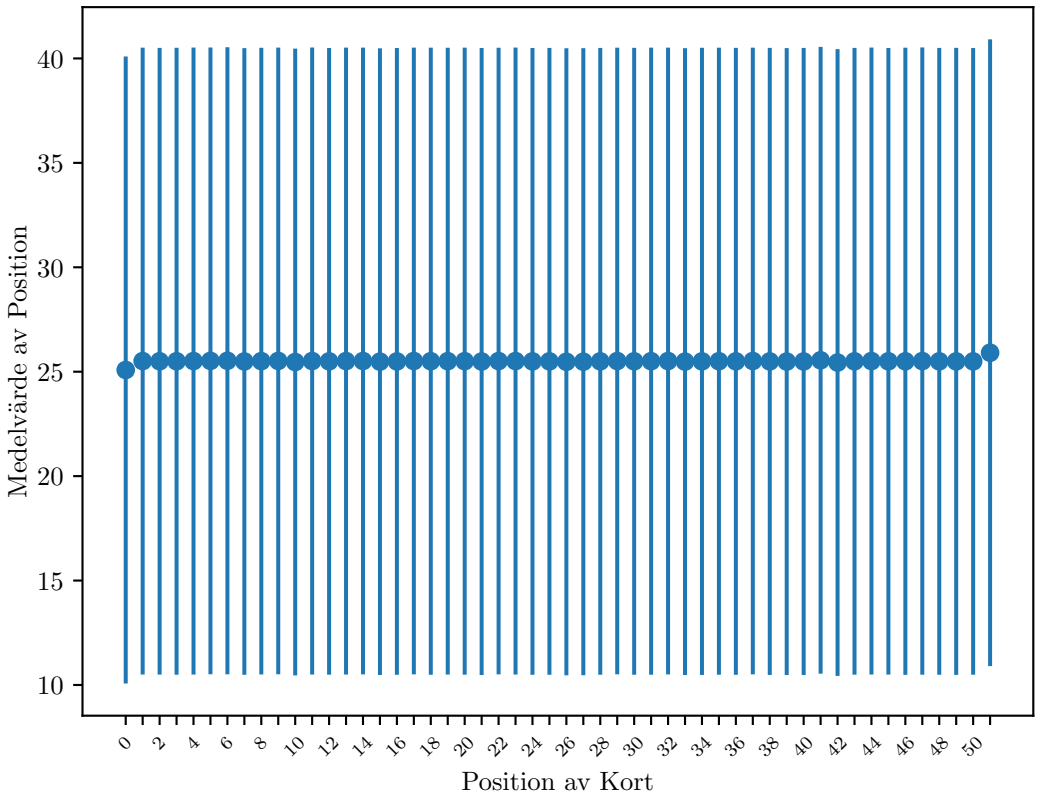
Figur 8: Resultatet från STDMean testet för Six Pile Shuffle med **2** iterationer.



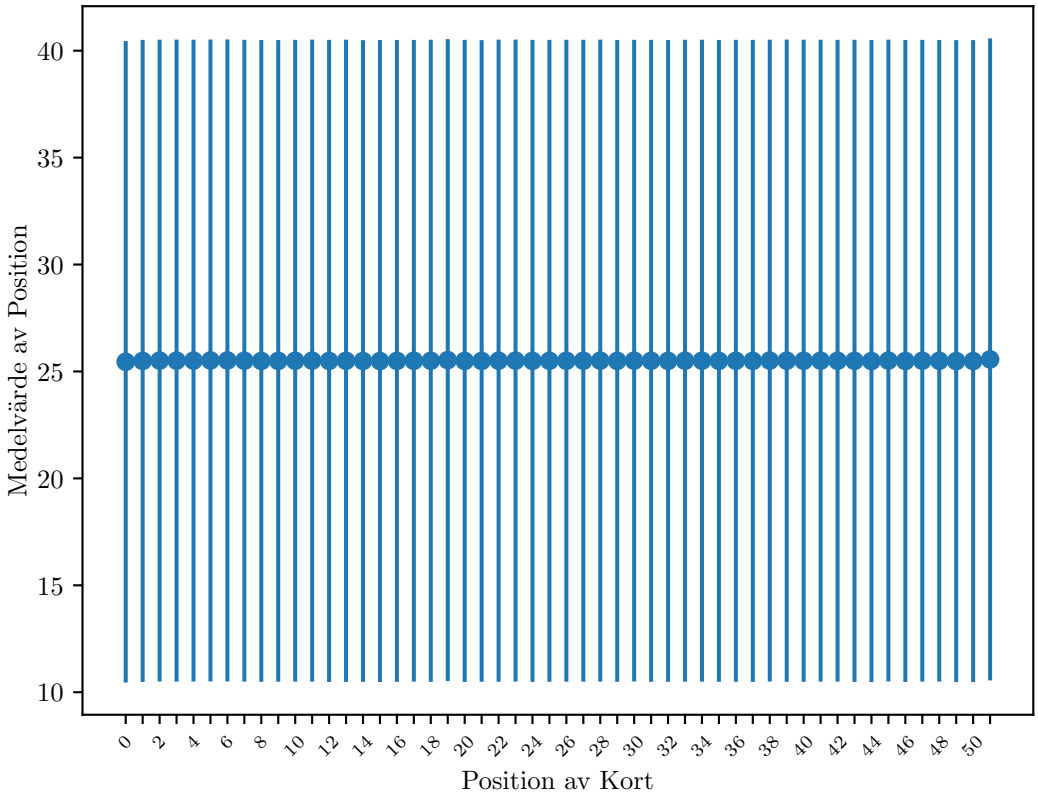
Figur 9: Resultatet från STDMean testet för SOC Pile Shuffle med **1** iteration.



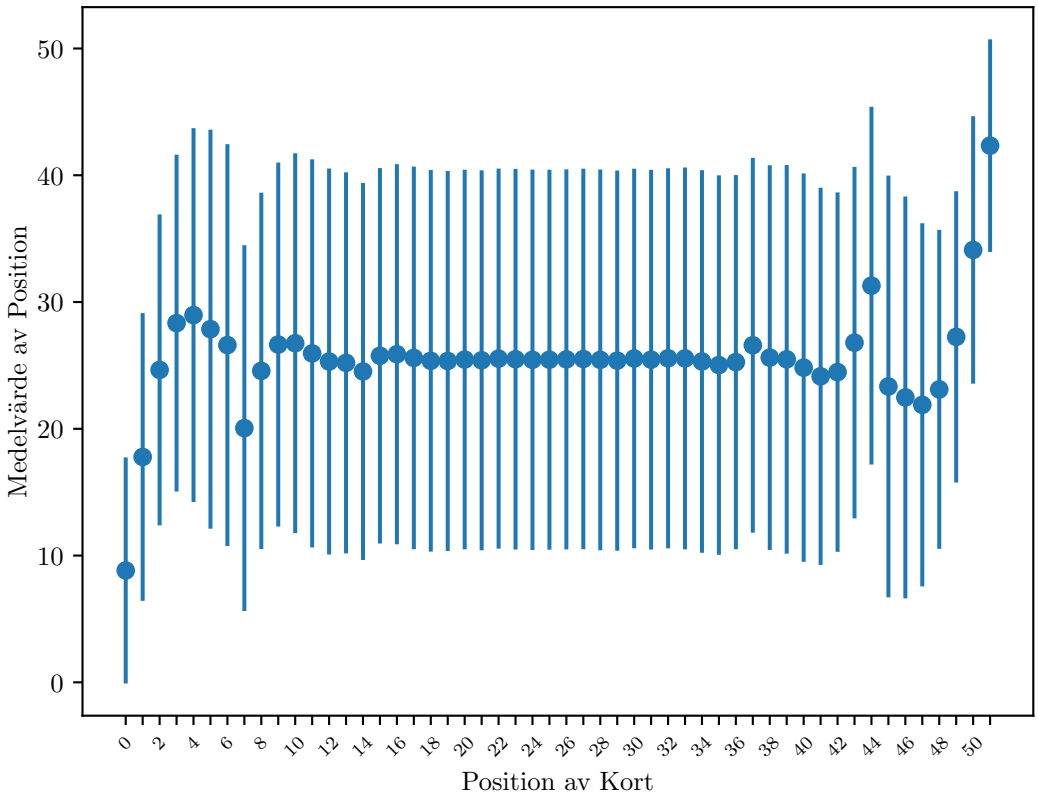
Figur 10: Resultatet från STDMean testet för SOC Pile Shuffle med **2** iterationer.



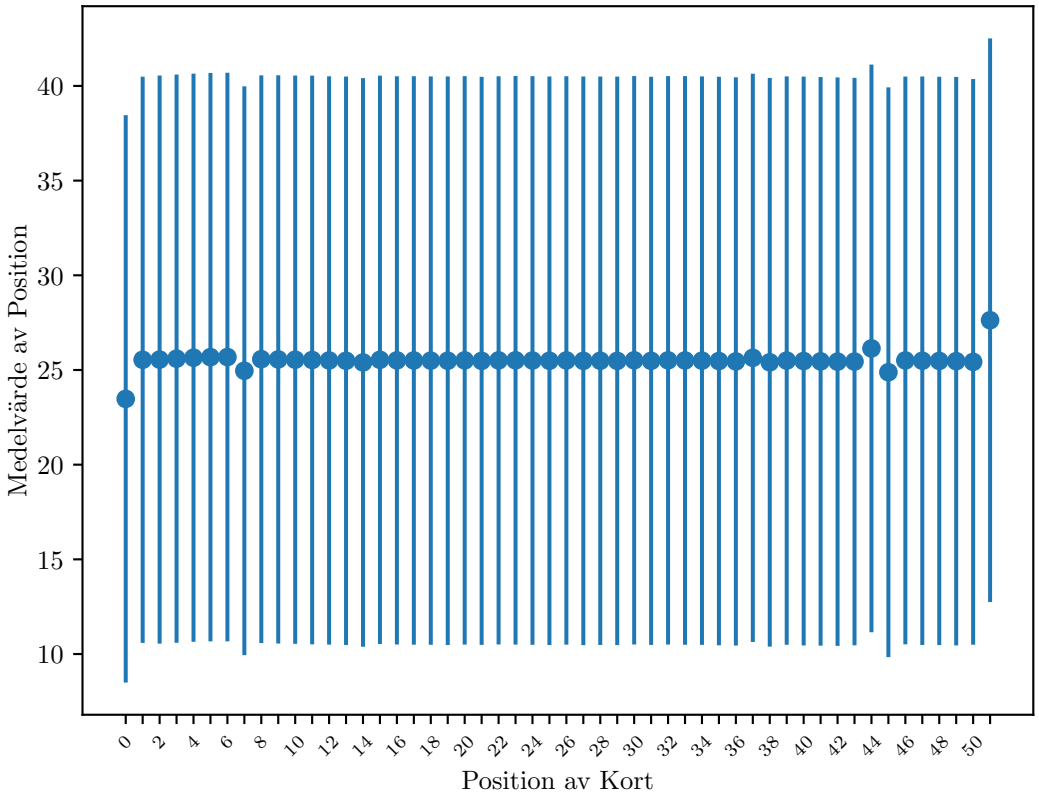
Figur 11: Resultatet från STDMean testet för SOC Pile Shuffle med **3** iterationer.



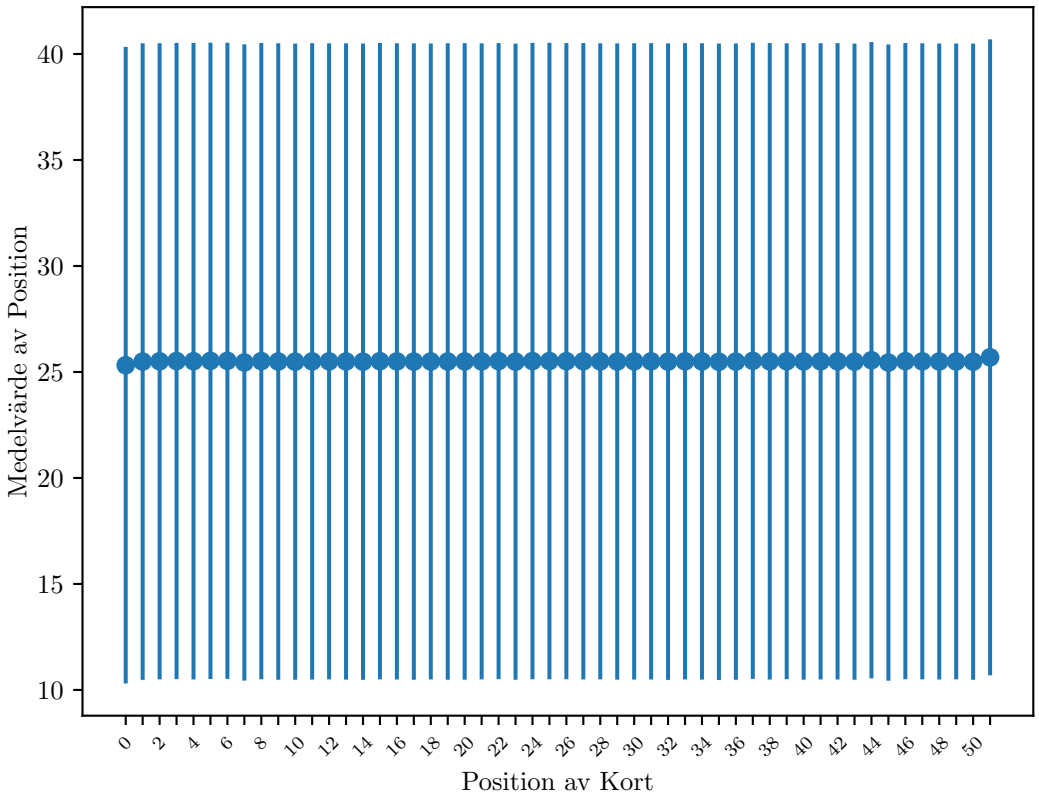
Figur 12: Resultatet från STDMean testet för SOC Pile Shuffle med **4** iterationer.



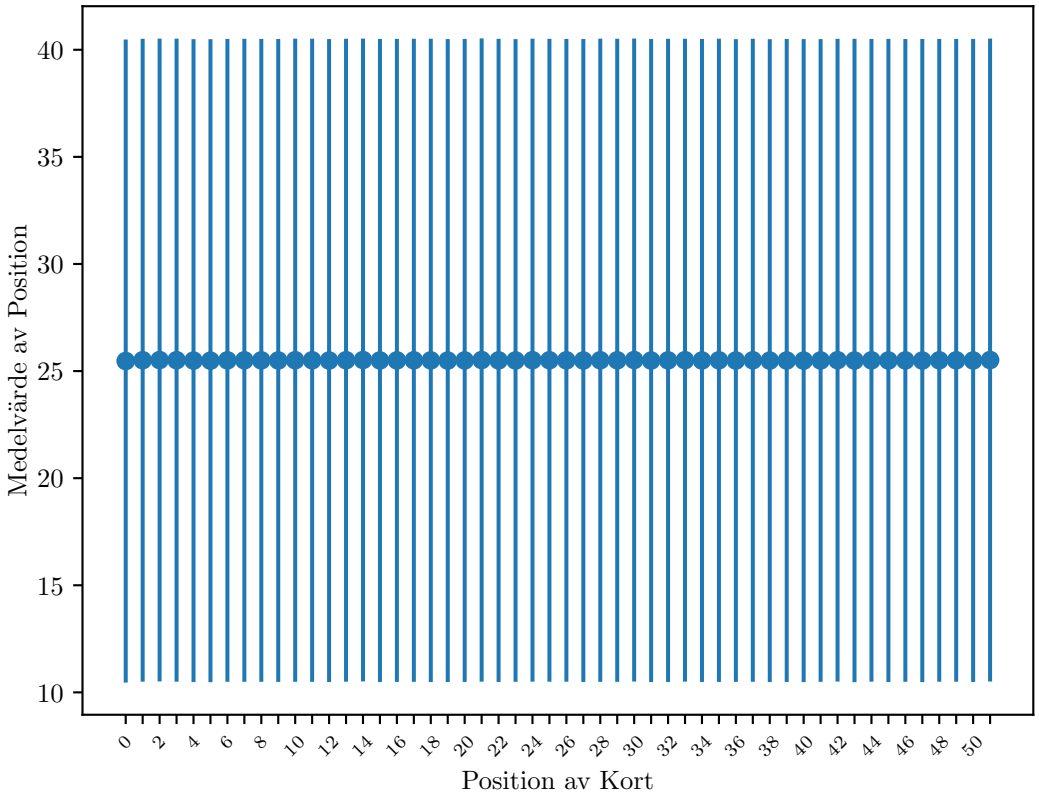
Figur 13: Resultatet från STDMean testet för Ten Pile Shuffle med **1** iteration.



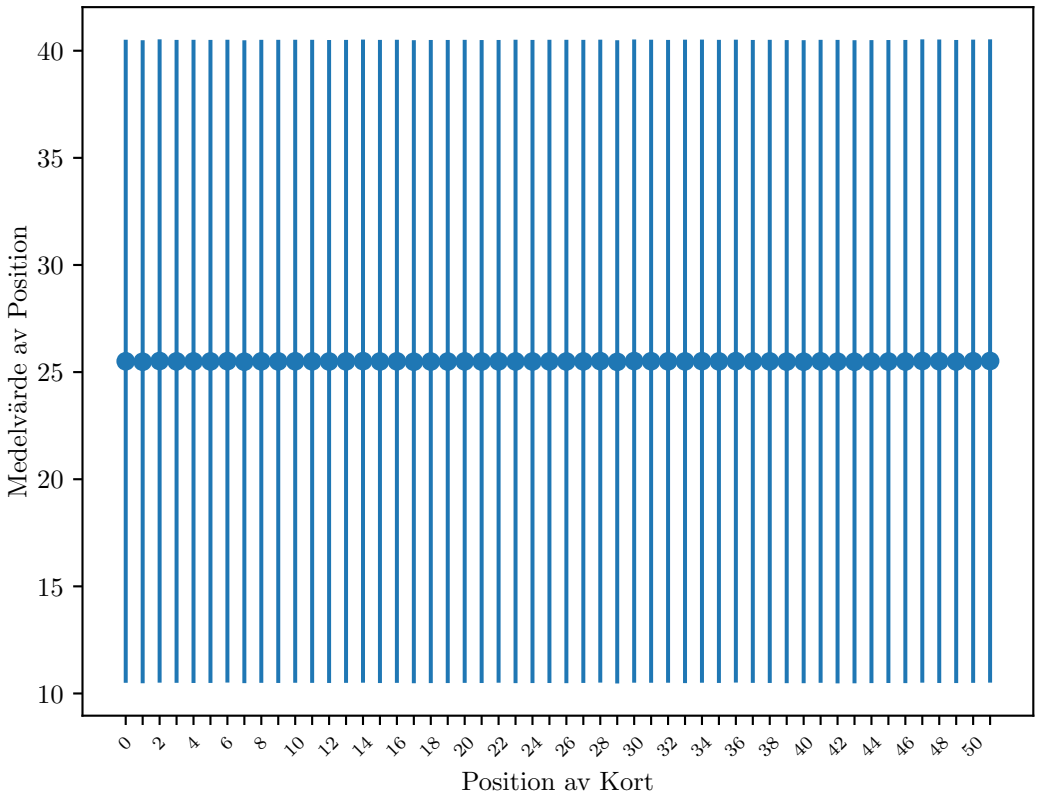
Figur 14: Resultatet från STDMean testet för Ten Pile Shuffle med **2** iterationer.



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



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



Figur 17: Resultatet från STDMean testet för Wheel Fisher-Yates shuffle med **1** iteration.

5 Diskussion

Och sista sektionen här, men den är relativt straightforward.

5.1 Den mest passande blandningsmetoden

5.1.1 Preliminär gallring

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`.

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

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

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

Matsumoto, Makoto och Takuji Nishimura (jan. 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).

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).

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.

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