МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ

_____

к.т.н, доц. Михелєв І.Л.

"____" _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

на тему: Розробка системи підрахунку еквілібріуму Неша в пуш-фолд холдемі

Виконав: студент групи 4142

_____ *Мудрієвський П. О.*
(підпис)

Керівник роботи:

старший викладач_____
(посада, науковий ступінь вчене звання)

_____ *Беркунський Є. Ю.*
(підпис)

м. Миколаїв – 2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами

Кафедра інформаційних управляючих систем та технологій

Спеціальність 122 "Комп'ютерні науки"

Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

_____

к.т.н, доц. Гайда А.Ю.

"____" _____ 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття ступеня вищої освіти "бакалавр"

Студенту Мудрієвському Петру Олеговичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка системи підрахунку еквілібріуму Неша в пуш-фолд холдемі

Керівник роботи старший викладач Беркунський Євгеній Юрійович

Затверджені наказом ректора № 269-уч від "18" травня 2022 року.

2. Термін подання роботи: 20 червня 2022 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем,

4. Перелік питань, що належать до розробки (найменування розділів): правила гри, побудова моделі гри, пошук оптимальної стратегії для двох гравців

5. Перелік презентаційних матеріалів: кількість різних комбінацій для кожної категорії та їх ранги, діапазони ручних карт, дерево рішень у пуш-фол холдемі з трьома гравцями, середній виграш проти опонентів з певними діапазонами, стратегія для двох гравців при розмірі стеку $S = 14$, стратегія для двох гравців при розмірі стеку $S = 21$

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Беркунський Є. Ю., старший викладач		
2	Беркунський Є. Ю., старший викладач		
3	Беркунський Є. Ю., старший викладач		
4			

7. Дата видачі завдання 14 лютого 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	28 лютого 2022 р.	
2	Постановка задачі	3 березня 2022 р.	
3	Аналіз математичної моделі проекту	17 березня 2022 р.	
4	Реалізація математичної моделі гри	2 квітня 2022 р.	
5	Реалізація проекту	19 квітня 2022 р.	
6	Виконання графічних документів	5 травня 2022 р.	
7	Вирішення задач з охорони праці	21 травня 2022 р.	
8	Подання дипломної роботи на перевірку рецензенту	10 червня 2022 р.	
9	Подання дипломної роботи на кафедральний захист	27 червня 2022 р.	
10			

Студент _____ П. О. Мудрієвський
(підпис) (ПІБ)

Керівник роботи _____ Є. Ю. Беркунський
(підпис) (ПІБ)

АНОТАЦІЯ

Вмотивовані успіхом застосування чисельних методів для обчислення рівноваги Неша в покері Куна на трьох гравців і широкою популярністю Техаського Холдему, ми розробили систему для розрахунку оптимальних стратегій для одного з варіантів гри – пуш-фолд холдему. Наша система використовує регуляризацію та метод Ньютона і здатна обчислювати стратегії для ігор для двох гравців з будь-яким розміром стека.

ABSTRACT

Motivated by success of applying numerical methods to computation of Nash equilibrium in three player Kuhn poker and widespread popularity of Texas Hold'em, we develop a system to compute optimal strategies for one of its variants – Jam-Fold Texas Hold'em. Our system uses regularization and Newton's method and is capable of computing strategies for two-player games with any stack size.

ЗМІСТ

ВСТУП

Техаський холдем – одна з найпопулярніших покерних ігор у світі. В нього грають вдома, в клубах, та онлайн. Популярність даної гри, а також її скалдність привертає увагу багатьох дослідників: це гра з неповною інформацією, де простір для стратегій надзвичайно великий.

Одним із підходів до створення агентів для ігор з неповною інформацією є пряме знаходження оптимальної стратегії – обчислення еквілібріуму Неша. Тому знаходження еквілібріуму Неша у іграх з багатьма гравцями – одна з центральних проблем теорії ігор. Для цього існує багато алгоритмів.

В даній роботі ми вибрали підхід, який використовує регуляризацію поліноміальних рівнянь і нерівностей, який до цього успішно застосовувався до розв'язання покеру Куна з 3 і більше картами [4]. Ми розширити цей метод до стандартного набору з 52 карт і більш цікавої версії покеру не тільки для дослідників, а й для звичайних людей [10].

Для реалізації системи обчислення оптимальних стратегій була обрана мова C++, як одна з найбільш ефективних у використанні обчислювальних ресурсів.

РОЗДІЛ 1. ПРАВИЛА ГРИ

1.1 Техаський пуш-фолд холдем покер

В техаський пуш-фолд холдем можуть грати два або більше гравці. Кожен з них має стек з S фішок. Спочатку передостанній гравеці кладе SB (англ. *small blind*) фішок у банк, а останній – BB (big blind) фішок (зазвичай $BB = 2SB$). Потім, як і в звичайному техаському холдемі, кожному гравцеві роздають дві карти з колоди. Ці дві карти називаються «ручними» або «дірковими» картками [1]. Гравці знають лише свої карти. Для роздачі використовується стандартна колода з 52 карт.

Після роздачі карт починається власе гра, де кожен гравець має послідовно зробити вибір: пуш – додати всі фішки, що залишилися, у банк, або фолд – скинути карти. Гравець 1 починає гру і робить свій вибір, потім гравець 2 і так далі. Вибір усіх попередніх гравців відомий наступному гравцю, який збирається діяти.

Якщо тільки один гравець вирішив «запушити», а всі інші скинули карит, він виграє всі фішки в банку. Якщо ж таких гравців було два, або більше, то з колоди роздається п’ять додаткових спільних карт [7], і гравець з найкращою покерною рукою виграє банк. Правила визначення найкращої руки в покері описані в Розділі 2.

Приклади:

1. У гру грають два гравці. Кожен має $S = 20$ фішок, $SB = 1$, $BB = 2$. Гравець 1 кладе $SB = 1$ фішку в банк. Гравець 2 кладе $BB = 2$ фішки в банк. Кожен отримує по дві карти. Гравець 1 вирішує покласти решту фішок $S - SB = 19$ у банк. На даний момент банк містить $S + BB = 22$ фішки. Гравець 2, побачивши рішення гравця 1, скидає карти. Гравець 1 виграє банк. Після гри Гравець 1 має $S + BB = 22$ фішки, а Гравець 2 має $S - BB = 18$ фішок.

2. У гру грають четверо гравців. Кожен має $S = 80$ фішок, $SB = 5$, $BB = 10$. Гравець 3 кладе $SB = 5$ фішок у банк. Гравець 4 кладе $BB = 10$ фішок у банк. У цей момент кількість фішок у банку становить $SB + BB = 15$. Кожен отримує по дві карти. Гравець 1 скидає карти. Гравець 2 пушить. Гравець 3 фолдить. Гравець 4 пушить. Тепер банк містить $2S + SB = 165$ фішок. Оскільки Гравець 2 і Гравець 4 запушили, роздаються спільні карти і переможець визначається за найкращою покерною комбінацією (Розділ 2). Скажімо, у Гравця 4 була краща комбінація, ніж у Гравця 2. У цьому випадку після гри Гравець 1 залишиться з усіма своїми $S = 80$ фішками, у Гравця 2 буде 0 фішок, у Гравця 3 – $S - SB = 75$ фішок, а у Гравця 4 – $2S + SB = 165$ фішок..

1.2 П’ятикарткові покерні комбінації

У Розділі 1.1 ми описали правила гри до роздачі спільних карт. Тепер давайте опишемо, що відбувається, якщо два або більше гравців пушать.

Спочатку роздається п’ять спільних карт. Далі кожен гравець, використовуючи будь-які п’ять карт із двох своїх карт і п’яти загальних карт, складає найкращу п’ятикарткову покерну комбінацію. Гравець з найкращою п’ятикартковою комбінацією виграє і забирає всі фішки в банку.

Кожна карта в колоді має масть і номінал. Існує 4 різних масті і 13 номіналів, з яких і складаються $4 \cdot 13 = 52$ карти. Номінали у покері впорядковані (від найвищого до нижчого): туз, король, королева, валет і від 10 до 2. Ми пронумеруємо їх від 0 до 12, де 0 означає туза, 1 — короля, і так далі: нижчі числа означають вищі номінали. Масті у покері вони не мають особливого порядку. Назви мастей: піки, чирви, бубни і трефи.

Замість повних українських назв ми будемо використовувати односимвольні скорочення англійських варіантів для номіналів: A (англ. *ace*), K (англ. *king*), Q (англ. *queen*), J (англ. *jack*), T (англ. *ten*), $9, \ldots 2$, – і для

мастей: s (англ. *spades*), h (англ. *hearts*), d (англ. *diamonds*), c (англ. *clubs*). Ми будемо писати скорочення номіналів та масті разом, для позначення карт, і декілька карт у фігурних дужках, для позначення комбінацій. Наприклад: піковий туз – As, десять чирв і 7 треф – $\{Th, 7c\}$.

Усі покерні комбінації діляться на 9 категорій [2]. Починаючи від найстаршої до найнижчої вони:

1. Стріт-флеш – 5 карт однієї масті з послідовними номіналами. Приклад: $\{Ts, 9s, 8s, 7s, 6s\}$. У цій комбінації туз можна використовувати і як найвищу, і як найнижчу карту. Це означає, що, наприклад, $\{5d, 4d, 3d, 2d, Ad\}$ також є стріт-флешем, але не $\{4h, 3h, 2h, Ah, Kh\}$. Стріт-флеші впорядковуються відповідно до карти, з якої вони починаються. Тобто, стріт-флеш, який починається з туза, є найвищим, а той, який починається з 5, — найнижчим.
2. Каре – 4 карти одного номіналу і 1 карта іншого номіналу (кікер). Приклад: $\{3s, 3c, 3h, 3d, Js\}$. Порядок: спочатку за рангом 4 карт, а потім за рангом кікера.
3. Фул-хаус – 3 карти одного номіналу і 2 карти іншого номіналу. Приклад: $\{5c, 5d, 5h, Ks, Kc\}$. Порядок: спочатку за номіналом 3 карт, а потім за номіналом 2 карт.
4. Флеш – 5 карт однієї масті, за винятком стріт-флешів. Приклад: $\{Ah, Jh, 9h, 8h, 2h\}$. Порядок: спочатку за номіналом найвищої карти, потім за номіналом другої найвищої карти і так далі.
5. Стріт – 5 карт з послідовними номіналами, за винятком стріт-флешів. Приклад: $\{Ks, Qs, Jh, Th, 9s\}$. Туз можна використовувати так само, як і для стріт-флешів: і як найстаршу, і як найнижчу карту. Стріти впорядковуються відповідно до карти, з якої вони починаються.
6. Трійка – 3 карти одного номіналу і 2 карти двох інших номіналів (кікери). Приклад: $\{7d, 7h, 7s, 6c, Kd\}$. Порядок: спочатку за

номіналом 3 карт, потім за номіналом вищого кікера і, нарешті, за номінолом нижчого кікера.

7. Дві пари – 2 карти одного номіналу, 2 карти іншого номіналу і 1 карта третього номіналу (кікер). Приклад: $\{Tc, Td, 4d, 4s, 5h\}$. Порядок: спочатку за номіналом вищої пари, потім за номіналом нижчої пари і, нарешті, за номіналом кікера.
8. Пара – 2 карти одного номіналу і три карти трьох інших номіналів (кікери). Приклад: $\{Ac, Ad, Qs, Th, 2d\}$. Порядок: спочатку за номіналом пари, потім за номіналом кікерів (від найвищого до нижчого).
9. Старша карта – 5 карт різних номіналів, за винятком стртів, флешів і стріт-флешів. Приклад: $\{Jc, 9s, 8h, 5h, 2s\}$. Порядок: спочатку за номіналом найвищої карти, потім за номіналом другої найвищої карти і так далі.

Покерні комбінації оцінюються спочатку за категоріями, а потім всередині категорій. Дві комбінації, що містять карти з однаковими номіналами, але різними мастями, вважаються рівними. Якщо два або більше гравців мають рівні комбінації наприкінці гри, то банк ділиться між ними порівну.

1.3 Оцінка покерних комбінацій

Існує $\binom{52}{5} = 2598960$ можливих п’ятикарткових покерних комбінації, але багато з них ідентичні. Наприклад, $\{6d, 6s, Kc, 9h, 3d\}$ ідентична до $\{6h, 6h, Kd, 9c, 3c\}$. Щоб вирішити, яка комбінація краща, ми використаємо той самий підхід, що й у [3], і присвоїмо ранг $HR(h)$ кожній комбінації h. Кращі комбінації матимуть нижчі ранги, а ідентичні комбінації матимуть однаковий ранг.

Категорія	**Кількість**	**Ранги**
Стріт-флеш	10	$HR = 0 \dots 9$
Каре	$\binom{13}{1}\binom{12}{1} = 156$	$HR = 10 \dots 165$
Фул-хаус	$\binom{13}{1}\binom{12}{1} = 156$	$HR = 166 \dots 321$
Флеш	$\binom{13}{5} - 10 = 1277$	$HR = 322 \dots 1598$
Стріт	10	$HR = 1599 \dots 1608$
Трійка	$\binom{13}{1}\binom{12}{2} = 858$	$HR = 1609 \dots 2466$
Дві пари	$\binom{13}{2}\binom{11}{1} = 858$	$HR = 2467 \dots 3324$
Пара	$\binom{13}{1}\binom{12}{3} = 2860$	$HR = 3325 \dots 6184$
Старша карта	$\binom{13}{5} - 10 = 1277$	$HR = 6185 \dots 7461$
Усього	**7462**	$\boldsymbol{HR = 0 \dots 7461}$

Таблиця 1.1 – Кількість різних комбінацій для кожної категорії та їх ранги

Реалізація алгоритму оцінки комбінацій знаходиться в Додатку А.

Приклад використання:

```
#include <iostream>
#include "poker.h"

int main() {
  std::cout
  << "hands ranks:\n"
  << "  straight flush\n"
  << "    As Ks Qs Js Ts: " << hand_rank({"As","Ks","Qs","Js","Ts"}) <<
"\n"
  << "    5s 4s 3s 2s As: " << hand_rank({"5s","4s","3s","2s","As"}) <<
"\n"
  << "  four of a kind\n"
  << "    As Ad Ac Ah Ks: " << hand_rank({"As","Ad","Ac","Ah","Ks"}) <<
"\n"
  << "    2s 2d 2c 2h 3s: " << hand_rank({"2s","2d","2c","2h","3s"}) <<
"\n"
```

```
  << "  full house\n"
  << "    As Ad Ac Kh Ks: " << hand_rank({"As","Ad","Ac","Kh","Ks"}) <<
"\n"
  << "    2s 2d 2c 3h 3s: " << hand_rank({"2s","2d","2c","3h","3s"}) <<
"\n"
  << "  flush\n"
  << "    As Ks Qs Js 9s: " << hand_rank({"As","Ks","Qs","Js","9s"}) <<
"\n"
  << "    7s 5s 4s 3s 2s: " << hand_rank({"7s","5s","4s","3s","2s"}) <<
"\n"
  << "  straight\n"
  << "    As Kd Qc Jh Ts: " << hand_rank({"As","Kd","Qc","Jh","Ts"}) <<
"\n"
  << "    5s 4d 3c 2h As: " << hand_rank({"5s","4d","3c","2h","As"}) <<
"\n"
  << "  three of a kind\n"
  << "    As Ad Ac Kh Qs: " << hand_rank({"As","Ad","Ac","Kh","Qs"}) <<
"\n"
  << "    2s 2d 2c 3h 4s: " << hand_rank({"2s","2d","2c","3h","4s"}) <<
"\n"
  << "  two pair\n"
  << "    As Ad Kc Kh Qs: " << hand_rank({"As","Ad","Kc","Kh","Qs"}) <<
"\n"
  << "    2s 2d 3c 3h 4s: " << hand_rank({"2s","2d","3c","3h","4s"}) <<
"\n"
  << "  pair\n"
  << "    As Ad Kc Qh Js: " << hand_rank({"As","Ad","Kc","Qh","Js"}) <<
"\n"
  << "    2s 2d 3c 4h 5s: " << hand_rank({"2s","2d","3c","4h","5s"}) <<
"\n"
  << "  high card\n"
  << "    As Kc Qh Jd 9s: " << hand_rank({"As","Kc","Qh","Jd","9s"}) <<
"\n"
  << "    2s 3d 4c 5h 7s: " << hand_rank({"2s","3d","4c","5h","7s"}) <<
"\n";
}
```

Результат:

```
hands ranks:
  straight flush
    As Ks Qs Js Ts: 0
    5s 4s 3s 2s As: 9
  four of a kind
    As Ad Ac Ah Ks: 10
    2s 2d 2c 2h 3s: 165
  full house
    As Ad Ac Kh Ks: 166
    2s 2d 2c 3h 3s: 321
  flush
    As Ks Qs Js 9s: 322
    7s 5s 4s 3s 2s: 1598
  straight
    As Kd Qc Jh Ts: 1599
```

```
  5s 4d 3c 2h As: 1608
three of a kind
  As Ad Ac Kh Qs: 1609
  2s 2d 2c 3h 4s: 2466
two pair
  As Ad Kc Kh Qs: 2467
  2s 2d 3c 3h 4s: 3324
pair
  As Ad Kc Qh Js: 3325
  2s 2d 3c 4h 5s: 6184
high card
  As Kc Qh Jd 9s: 6185
  2s 3d 4c 5h 7s: 7461
```

1.4 Діапазони ручних карт

Існує $\binom{52}{2} = 1326$ комбінацій з двох карт, які кожен гравець може мати у своїй початковій руці. Оскільки під час оцінки п'ятикарткових покерних комбінацій усі масті симетричні, лише 169 з цих 1326 комбінацій є унікальними: $\binom{13}{2} = 78$ комбінацій двох карт з різними номіналами та різними мастями, $\binom{13}{2} = 78$ комбінацій з різними номіналами та однаковими мастями, а також 13 комбінацій з однаковими номіналами. В подальшому ми будемо називати ці $78 + 78 + 13 = 169$ різних комбінацій ручних карт «діапазонами».

Ми визначимо номер діапазона R для карт c_1 і c_2, з номіналами r_1 і r_2 та мастями s_1 і s_2, де $r_1 \leq r_2$ і $0 \leq r_1, r_2 \leq 12$, таким чином:

$$R(c_1, c_2) = \begin{cases} r_1 + 13r_2, \text{if } s_1 \neq s_2 \\ 13r_1 + r_2, \text{if } s_1 = s_2 \end{cases}$$

Так, наприклад, номер діапазону «Туз і Король однакової масті» дорівнює $13 \cdot (A = 0) + (K = 1) = 1$, а номер діапазону $\{A, K\}$ різних мастей дорівнює $(A = 0) + 13 \cdot (K = 1) = 13$ (див. таблицю 1.2).

		Однакові масті												
Різні масті		A	K	Q	J	T	9	8	7	6	5	4	3	2
	A	0	1	2	3	4	5	6	7	8	9	10	11	12
	K	13	14	15	16	17	18	19	20	21	22	23	24	25
	Q	26	27	28	29	30	31	32	33	34	35	36	37	38
	J	39	40	41	42	43	44	45	46	47	48	49	50	51
	T	52	53	54	55	56	57	58	59	60	61	62	63	64
	9	65	66	67	68	69	70	71	72	73	74	75	76	77
	8	78	79	80	81	82	83	84	85	86	87	88	89	90
	7	91	92	93	94	95	96	97	98	99	100	101	102	103
	6	104	105	106	107	108	109	110	111	112	113	114	115	116
	5	117	118	119	120	121	122	123	124	125	126	127	128	129
	4	130	131	132	133	134	135	136	137	138	139	140	141	142
	3	143	144	145	146	147	148	149	150	151	152	153	154	155
	2	156	157	158	159	160	161	162	163	164	165	166	167	168

Таблиця 1.2. Діапазони ручних карт. Блакитний символізує карти однакової масті, а оранжевий – карти різних мастей.

РОЗДІЛ 2. ПОБУДОВА МОДЕЛІ ГРИ

2.1 Ігрове дерево

У дереві грі з N гравцями існує $\sum_{i=1}^{N} 2^{i-1} = 2^N - 1$ вузлів прийняття рішень і 2^N термінальних вузлів. Гравець i контроцює 2^{i-1} вузлів прийняття рішень, оскільки кожен попередній гравець прийняв одне з двох рішень: пуш, або фолд.

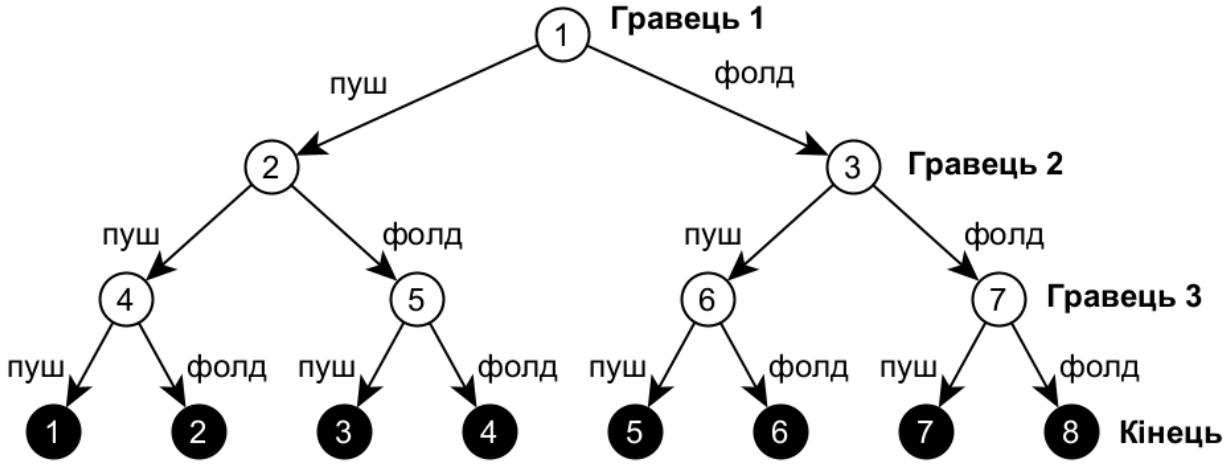

Рисунок 2.1. Дерево рішень у пуш-фол холдемі з трьома гравцями. Білі кола – вузли прийняття рішень. Чорні кола – термінальні вузли. Мітки, виділені жирним шрифтом, застосовуються до всіх вузлів по горизонталі. Гравець 1 приймає рішення у вузлі 1, гравець 2 – у вузлах 2 і 3, гравець 3 – у вузлах 4, 5, 6 і 7.

Гравець i контролює вузли $2^{i-1} \ldots 2^i - 1$. Якщо гравець у вузлі n прийняв рішшеня пушити, вузол наступного гравця — $2n$, інакше — $2n + 1$ (рисунок 2.1).

2.2 Вартість гри

Ми позначимо через x_{nr} частоту з якою гравець у вузлі n пушить, де $n = 1 \dots 2^N - 1$, маючи карти з діапазону r у руці, $r = 0 \dots 168$. Частота, з якою гравець скидає карти буде $1 - x_{nr}$ відповідно.

Вартість гри для гравця i позначимо $E_i(x_{nr})$, $i = 1 \dots N$.

Нехай $P\left(r, \{r_1^j, \dots r_m^j\}, \{r_1^f, \dots r_k^f\}\right)$ – математичне очікування частини банку, яку виграє гравець з діапазоном r, якщо гравці з діапазонами $r_1^j, \dots r_m^j$ пушать, а гравці з діапазонами $r_1^f, \dots r_k^f$ фолдять, $m + k + 1 = N$. І нехай $T(r_1, \dots r_N)$ – ймовірність того, що гравці мають певні діапазони $\{r_1, \dots r_N\}$ під час гри. Маючи P, T і знаючи частоти x_{nr}, ми можемо перебрати всі термінальні вузли та обчислити значення гри E_i для кожного гравця i, $i = 1 \dots N$.

Наприклад, у грі з трьома гравцями, вартість для гравця 1 у термінальному вузлі, де гравець 1 пушить, гравець 2 фолдить і гравець 3 пушить (термінальний вузол 3 на рисунку 2.1) дорівнює

$$\sum_{r_1, r_2, r_3} \left[T(r_1, r_2, r_3) \cdot x_{1r_1} \cdot \left(1 - x_{2r_2}\right) \cdot x_{5r_3} \cdot \{P(r_1, \{r_2, r_3\}, \{\}) \cdot (2S + SB) - S\}\right]$$

де S – розмір стеку, а SB – малий блайнд. Імовірність того, що гравець 1 виграє банк, дорівнює $P(r_1, \{r_2, r_3\}, \{\})$, розмір банку дорівнює $2S + SB$ і він ставить S фішок.

Вартість для гравця 2 у тому самому термінальному вузлі дорівнює

$$\sum_{r_1, r_2, r_3} \left[T(r_1, r_2, r_3) \cdot x_{1r_1} \cdot \left(1 - x_{2r_2}\right) \cdot x_{5r_3} \cdot (-SB)\right]$$

Тому що він кладе SB у банк і скидає карти.

Вартість для гравця 3 дорівнює

$$\sum_{r_1, r_2, r_3} \left[T(r_1, r_2, r_3) \cdot x_{1r_1} \cdot \left(1 - x_{2r_2}\right) \cdot x_{5r_3} \cdot \{P(r_3, \{r_1, r_2\}, \{\}) \cdot (2S + SB) - S\}\right]$$

Тому що для нього імовірність виграти дорівнює $P(r_3, \{r_1, r_2\}, \{\})$.

Якщо просумувати вартість усіх термінальних вузлів, отримаємо вартість гри E_i.

Оскільки гравець i прагне максимізувати E_i, еквілібріум досягається, коли множина частот x_{nr} задовольняє набору рівнянь і нерівностей

$$\frac{\partial E_i}{\partial x_{nr}} = 0, \text{ або}$$

$$\frac{\partial E_i}{\partial x_{nr}} < 0 \text{ і } x_{nr} = 0, \text{ або}$$

$$\frac{\partial E_i}{\partial x_{nr}} > 0 \text{ і } x_{nr} = 1,$$

де $n = 2^{i-1} \ldots 2^i - 1$

(1)

2.3 Регуляризована система

Щоб розв'язати систему (1), ми скористаємося тим же методом, що й у [4], і замість цього розв'яжемо систему нелінійних рівнянь

$$g\left(\epsilon \frac{\partial E_i}{\partial x_{nr}}\right) - x_{nr} = 0,$$

(2)

де ϵ – параметр регуляризації і

$$g(y) = \frac{1}{2} + \frac{1}{\pi}\tan^{-1}(y)$$

При $\epsilon \to \infty$ регуляризована система (2) наближається до (1).

У наступному ми розв'яжемо (2) для двох гравців за допомогою методу Ньютона, використовуючи Eigen – C++ бібліотеку для лінійної алгебри [6].

РОЗДІЛ 3. ПОШУК ОПТИМАЛЬНОЇ СТРАТЕГІЙ ДЛЯ ДВОХ ГРАВЦІВ

3.1 Таблиці пошуку для значень *P* і *T*

Перед тим, як почати роз'вязувати систему (1), ми маємо порахувати значення P і T, які використовуються при підрахунку вартості гри.

Використовуючи наш алгоритм оцінки комбінацій (Додаток А), ми створимо таблицю пошуку точних значень P і T для двох гравців розміром $169x169$ (Додаток В).

Як приклад наведемо таблицю (3.1) значень $P(r,\{\acute{r}\},\{\,\})$, коли діапазоном суперника є пара тузів $\{A,A\}$ ($\acute{r} = R(Ax,Ay) = 0$). З таблиці ми бачимо, що найкращий діапазон проти пари тузів – це інша пара тузів з середнім виграшем 50%. А другий найкращий діапазон – $\{6,5\}$ однієї масті – з середнім виграшем $\frac{770497}{3424608} \approx 22.4988\%$.

		Однакові масті												
Різні масті		A	K	Q	J	T	9	8	7	6	5	4	3	2
	A	50%	12%	13%	13%	13%	12%	12%	12%	12%	13%	13%	13%	12%
	K	7%	18%	17%	17%	18%	17%	16%	16%	16%	16%	16%	15%	15%
	Q	7%	13%	18%	19%	20%	19%	18%	16%	17%	17%	16%	16%	15%
	J	8%	13%	15%	19%	21%	21%	19%	18%	17%	17%	17%	16%	16%
	T	8%	14%	16%	17%	19%	22%	21%	20%	19%	17%	17%	17%	16%
	9	6%	13%	15%	17%	18%	19%	22%	21%	20%	18%	16%	16%	16%
	8	7%	12%	14%	15%	17%	18%	20%	22%	21%	20%	18%	16%	16%
	7	7%	12%	12%	14%	16%	17%	19%	20%	22%	21%	19%	18%	16%
	6	6%	12%	13%	13%	14%	16%	17%	19%	20%	22%	21%	19%	18%
	5	8%	12%	12%	13%	13%	14%	16%	17%	19%	19%	21%	19%	18%
	4	8%	12%	12%	12%	13%	12%	14%	15%	17%	17%	19%	19%	17%
	3	7%	11%	12%	12%	12%	12%	12%	14%	15%	15%	15%	18%	17%
	2	7%	11%	11%	12%	12%	12%	12%	12%	13%	14%	13%	13%	18%

Таблиця 3.1. Середній виграш проти опонента з діапазоном $\{A, A\}$.

Ми також наведемо аналогічну таблицю для другого найкращого діапазону проти тузів $-\{6,5\}$ однієї масті (Таблиця 3.2), $\acute{r} = R(6x, 5x) = 13 \cdot 8 + 9 = 113$. Найкращий діапазон проти цього $-\{6,6\}$, з середнім виграшем $\frac{697019}{856152} \approx 81.413\%$. А найгірший діапазон $-\{5,2\}$ різних мастей $-$ $\frac{1150999}{3424608} \approx 33.6097\%$.

		Однакові масті												
		A	K	Q	J	T	9	8	7	6	5	4	3	2
Різні масті	A	78%	61%	61%	62%	62%	61%	61%	61%	68%	67%	57%	56%	55%
	K	58%	78%	62%	62%	62%	62%	61%	61%	67%	67%	56%	55%	55%
	Q	59%	59%	78%	62%	63%	62%	62%	61%	67%	66%	56%	55%	55%
	J	59%	59%	60%	78%	64%	63%	62%	62%	66%	66%	56%	55%	55%
	T	59%	60%	60%	62%	78%	64%	63%	63%	67%	65%	56%	55%	55%
	9	59%	59%	59%	60%	61%	79%	63%	63%	66%	65%	55%	55%	54%
	8	59%	59%	59%	59%	60%	61%	80%	63%	65%	65%	55%	54%	53%
	7	59%	59%	59%	59%	60%	60%	61%	81%	64%	64%	54%	53%	52%
	6	66%	66%	65%	65%	66%	64%	64%	63%	81%	50%	45%	44%	43%
	5	66%	65%	65%	64%	64%	64%	63%	62%	48%	60%	38%	38%	37%
	4	54%	53%	53%	53%	53%	52%	52%	52%	42%	35%	50%	38%	37%
	3	53%	52%	52%	53%	53%	52%	51%	50%	41%	34%	36%	49%	38%
	2	53%	52%	52%	52%	52%	51%	50%	49%	40%	34%	35%	35%	48%

Таблиця 3.2. Середній виграш проти опонента з діапазоном {6,5} однієї масті.

		Однакові масті												
Різні масті		A	K	Q	J	T	9	8	7	6	5	4	3	2
	A	24	16	16	16	16	16	16	16	12	12	16	16	16
	K	48	24	16	16	16	16	16	16	12	12	16	16	16
	Q	48	48	24	16	16	16	16	16	12	12	16	16	16
	J	48	48	48	24	16	16	16	16	12	12	16	16	16
	T	48	48	48	48	24	16	16	16	12	12	16	16	16
	9	48	48	48	48	48	24	16	16	12	12	16	16	16
	8	48	48	48	48	48	48	24	16	12	12	16	16	16
	7	48	48	48	48	48	48	48	24	12	12	16	16	16
	6	36	36	36	36	36	36	36	36	12	12	12	12	12
	5	36	36	36	36	36	36	36	36	24	12	12	12	12
	4	48	48	48	48	48	48	48	48	36	36	24	16	16
	3	48	48	48	48	48	48	48	48	36	36	48	24	16
	2	48	48	48	48	48	48	48	48	36	36	48	48	24

Таблиця 3.3. Імовірність $T(r_1, r_2)$ (помножена на $\binom{52}{2}\binom{50}{2} = 1624350$) того, що гравці мають певні діапазони, де $r_1 = 113$ ({6,5} однієї масті).

3.2 Результати для двох гравців

Маючи таблиці значень P і T, ми можемо порахувати вартість гри E_i для кожного гравця, похідні $\frac{\partial E_i}{\partial x_{nr}}$ по кожній змінній x_{nr}, разом з якобіанами [8] (див. Додаток Г), і розвязати систему (2) використовуючи метод Ньютона [9].

Ми записуємо всі змінні x_{nr} у вигляді вектора X розміру $(2^N - 1) \cdot 169 = 507$. Суфікси n і r перетворюються в один індекс $i = (n-1) \cdot 169 + r$.

Ми ініціалізуємо X випадковими значеннями з рівномірного розподілу $[0,1]$. Спочатку $\epsilon = 10^{-4}$. Після 20 ітерацій методу Ньютона ми множимо ϵ на деяке значення τ. Ми починаємо з $\tau = 10$, і якщо метод Ньютона не збігається, ми ділимо τ навпіл і обчислюємо знову. Ми зупиняємося, коли ϵ досягає 10^9.

У наступних таблицях (3.4 – 3.7) ми представляємо деякі з порахованих нами стратегій.

		Однакові масті												
		A	K	Q	J	T	9	8	7	6	5	4	3	2
Різні масті	A	100	100	100	100	100	100	100	100	100	100	100	100	100
	K	100	100	100	100	100	100	100	100	100	100	100	100	100
	Q	100	100	100	100	100	100	100	100	100	100	100	100	100
	J	100	100	100	100	100	100	100	100	100	100	100	100	100
	T	100	100	100	100	100	100	100	100	100	100	100	0	0
	9	100	100	100	100	100	100	100	100	100	100	0	0	0
	8	100	100	100	100	100	100	100	100	100	100	0.2	0	0
	7	100	100	100	100	100	100	100	100	100	100	100	0	0
	6	100	100	100	0	0	0	0	100	100	100	100	0	0
	5	100	100	100	0	0	0	0	0	0	100	100	100	0
	4	100	100	4.5	0	0	0	0	0	0	0	100	0.1	0
	3	100	100	0	0	0	0	0	0	0	0	0	100	0
	2	100	100	0	0	0	0	0	0	0	0	0	0	100

Таблиця 3.4. Частоти x_{1r} з якими гравець 1 має пушити.

$$SB = 0.5, BB = 1,\ S = 14$$

		Однакові масті												
		A	K	Q	J	T	9	8	7	6	5	4	3	2
Різні масті	A	100	100	100	100	100	100	100	100	100	100	100	100	100
	K	100	100	100	100	100	100	100	100	100	100	100	100	100
	Q	100	100	100	100	100	100	100	100	100	100	100	0	0
	J	100	100	100	100	100	100	100	100	0	0	0	0	0
	T	100	100	100	100	100	100	100	0	0	0	0	0	0
	9	100	100	100	100	100	100	100	0	0	0	0	0	0
	8	100	100	100	0	0	0	100	0	0	0	0	0	0
	7	100	100	78.3	0	0	0	0	100	0	0	0	0	0
	6	100	100	0	0	0	0	0	0	100	0	0	0	0
	5	100	100	0	0	0	0	0	0	0	100	0	0	0
	4	100	100	0	0	0	0	0	0	0	0	100	0	0
	3	100	100	0	0	0	0	0	0	0	0	0	100	0
	2	100	100	0	0	0	0	0	0	0	0	0	0	100

Таблиця 3.5. Частоти x_{2r} з якими гравець 2 має пушити, після того, як гравець 1 запушив. $SB = 0.5, BB = 1, S = 14$

		Однакові масті												
		A	K	Q	J	T	9	8	7	6	5	4	3	2
Різні масті	A	100	100	100	100	100	100	100	100	100	100	100	100	100
	K	100	100	100	100	100	100	100	100	100	100	100	100	100
	Q	100	100	100	100	100	100	100	100	100	100	100	100	100
	J	100	100	100	100	100	100	100	100	100	100	100	0	0
	T	100	100	100	100	100	100	100	100	100	58.5	0	0	0
	9	100	100	100	100	100	100	100	100	100	100	0	0	0
	8	100	100	100	100	100	100	100	100	100	100	0	0	0
	7	100	100	0	0	0	0	100	100	100	100	100	0	0
	6	100	100	0	0	0	0	0	0	100	100	100	0	0
	5	100	100	0	0	0	0	0	0	0	100	100	100	0
	4	100	100	0	0	0	0	0	0	0	0	100	0	0
	3	100	100	0	0	0	0	0	0	0	0	0	100	0
	2	100	100	0	0	0	0	0	0	0	0	0	0	100

Таблиця 3.6. Частоти x_{1r} з якими гравець 1 має пушити.

$SB = 0.5, BB = 1, S = 21$

		Однакові масті												
		A	K	Q	J	T	9	8	7	6	5	4	3	2
Різні масті	A	100	100	100	100	100	100	100	100	100	100	100	100	100
	K	100	100	100	100	100	100	100	100	100	100	100	0.1	0
	Q	100	100	100	100	100	100	100	0	0	0	0	0	0
	J	100	100	100	100	100	100	0	0	0	0	0	0	0
	T	100	100	100	100	100	0.1	0	0	0	0	0	0	0
	9	100	100	93.2	0	0	100	0	0	0	0	0	0	0
	8	100	100	0	0	0	0	100	0	0	0	0	0	0
	7	100	100	0	0	0	0	0	100	0	0	0	0	0
	6	100	0	0	0	0	0	0	0	100	0	0	0	0
	5	100	0	0	0	0	0	0	0	0	100	0	0	0
	4	100	0	0	0	0	0	0	0	0	0	100	0	0
	3	100	0	0	0	0	0	0	0	0	0	0	100	0
	2	100	0	0	0	0	0	0	0	0	0	0	0	100

Таблиця 3.7. Частоти x_{2r} з якими гравець 2 має пушити, після того, як гравець 1 запушив. $SB = 0.5, BB = 1, S = 21$

ВИСНОВОК

В даній дипломній роботі ми розробили систему для обчислення рівноваги Неша для техаського пуш-фолд холдему для двох гравців.

Цікаво відзначити, що в стратегіях, які ми розрахували, більшість частот є тривіальними: або 100%, або 0%. Лише деякі з них лежать у $(0,1)$. Примітними прикладами є: частота пушу з $\{Q, 7\}$ різних мастей для гравця 2, після того, як гравець 1 запушив, коли $S = 14$: $78{,}3\%$ (табл. 3.5) і частота пушу для гравця 1 з $\{10{,}5\}$ з однакової масті, коли $S = 21$: $58{,}5\%$ (табл. 3.6).

З отриманих результатів можна також зробити висновок, що при збільшенні розміру стеку S, кількість діапазонів з якими вигідно пушити зменшується. Так, для порахованих нами стратегій ця кількість зменшилась від 116 до 106 для гравця 1 і від 82 до 62 для гравця 2, в той час, як розмір стеку змінився від 14 до 21.

У майбутньому ми плануємо розширити нашу систему та розрахувати оптимальні стратегії для трьох [5] і чотирьох гравців. Основна проблема полягає в обчисленні великих таблиць пошуку P і T очікуваних виплат для більш ніж двох гравців. Наприклад, для чотирьох гравців така таблиця міститиме 169^4 записів, що становитиме більше 3 Гб чисел з плаваючою комою. Ми припускаємо, що зможемо подолати ці проблеми, використовуючи більше обчислювальних ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Glossary of poker terms [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення: https://en.wikipedia.org/wiki/Glossary_of_poker_terms – дата звернення: 06.06.2022.
2. List of poker hands [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення: https://en.wikipedia.org/wiki/List_of_poker_hands – дата звернення: 06.06.2022.
3. Cactus Kev's Poker Hand Evaluator [Електронний ресурс] – Режим звернення: http://suffe.cool/poker/evaluator.html – дата звернення: 06.06.2022.
4. John Billingham. Equilibrium solutions of three player Kuhn poker with $N > 3$ cards: A new numerical method using regularization and arc-length continuation. [Електронний ресурс]: arXiv:1802.04670. – Режим звернення: https://arxiv.org/abs/1802.04670 – дата звернення: 06.06.2022.
5. Sam Ganzfried, Tuomas Sandholm. Computing an Approximate Jam/Fold Equilibrium for 3-player No-Limit Texas Hold'em Tournaments. Proc. Of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)
6. Eigen, a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms [Електронний ресурс] – http://eigen.tuxfamily.org/ – дата звернення: 06.06.2022.
7. Community card poker [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення: https://en.wikipedia.org/wiki/Community_card_poker – дата звернення: 06.06.2022.
8. Jacobian matrix and determinant [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення:

https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant – дата звернення: 09.06.2022.

9. Newton's method [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення: https://en.wikipedia.org/wiki/Newton%27s_method#Systems_of_equations – дата звернення: 09.06.2022.
10. All-In or Fold [Електронний ресурс]: GGPoker. – Режим звернення: https://en.ggpoker.com/poker-games/all-in-or-fold/ – дата звернення: 10.06.2022.

ДОДАТОК А – АЛГОРИТМ ОЦІНКИ ПОКЕРНИХ КОМБІНАЦІЙ

```
int hand_rank(std::vector<card> hand) {
  std::array<int, ranks.size()> rank_n = {};
  std::array<int, suits.size()> suit_n = {};
  std::array<std::array<bool, ranks.size()>, suits.size()> suit_ranks = {};
  for (auto& card : hand) {
    suit_n[card.suit()]++;
    rank_n[card.rank()]++;
    suit_ranks[card.suit()][card.rank()] = true;
  }

  bool straight_flush = false;
  rank_t straight_flush_rank;
  bool flush = false;
  int flush_rank;
  bool straight = false;
  rank_t straight_rank;
  bool quads = false;
  rank_t quads_rank;
  bool quads_kicker = false;
  rank_t quads_kicker_rank;
  bool trips = false;
  rank_t trips_rank;
  bool trips_kicker = false;
  rank_t trips_kicker_rank;
  bool pair = false;
  rank_t pair_rank;
  bool second_pair = false;
  rank_t second_pair_rank;
  bool pair_kicker = false;
  rank_t pair_kicker_rank;
  std::vector<rank_t> kickers;

  // Check for flush and stright flush
  for (auto suit : suits) {
    if (suit_n[suit] >= 5) {
      std::vector<rank_t> flush_kickers;
      int sequential = 0;
      rank_t first_sequential_rank;
      for (auto rank : straight_ranks) {
        if (suit_ranks[suit][rank]) {
          sequential += 1;
          if (sequential == 1)
            first_sequential_rank = rank;
          if (sequential == 5 && (!straight_flush || straight_flush_rank >
first_sequential_rank)) {
            straight_flush_rank = first_sequential_rank;
            straight_flush = true;
          }
          if (flush_kickers.size() < 5)
            flush_kickers.push_back(rank);
        }
        else {
          sequential = 0;
```

```
        }
      }
      auto new_flush_rank = five_rank({ flush_kickers[0], flush_kickers[1],
flush_kickers[2], flush_kickers[3], flush_kickers[4] });
      if (!straight_flush && (!flush || flush_rank > new_flush_rank)) {
        flush_rank = new_flush_rank;
        flush = true;
      }
    }
  }

  if (!straight_flush) {
    int sequential = 0;
    rank_t first_sequential_rank;
    // Check for straights
    for (auto rank : straight_ranks) {
      if (rank_n[rank]) {
        sequential += 1;
        if (sequential == 1)
          first_sequential_rank = rank;
        if (sequential == 5 && !straight) {
          straight_rank = first_sequential_rank;
          straight = true;
        }
      }
      else {
        sequential = 0;
      }
    }
    // Check for everything else
    for (auto rank : ranks) {
      auto n = rank_n[rank];
      if (n) {
        if (n == 4) {
          if (!quads) {
            quads_rank = rank;
            quads = true;
          }
          else if (!quads_kicker) {
            quads_kicker_rank = rank;
            quads_kicker = true;
          }
        }
        else {
          if (!quads_kicker) {
            quads_kicker_rank = rank;
            quads_kicker = true;
          }
        }
        if (n == 3) {
          if (!trips) {
            trips_rank = rank;
            trips = true;
          }
          else if (!trips_kicker) {
            trips_kicker_rank = rank;
```

```
            trips_kicker = true;
          }
        }
        if (n == 2) {
          if (!trips_kicker) {
            trips_kicker_rank = rank;
            trips_kicker = true;
          }
          if (!pair) {
            pair_rank = rank;
            pair = true;
          }
          else if (!second_pair) {
            second_pair_rank = rank;
            second_pair = true;
          }
          else if (!pair_kicker) {
            pair_kicker_rank = rank;
            pair_kicker = true;
          }
        }
        if (n == 1) {
          if (!pair_kicker) {
            pair_kicker_rank = rank;
            pair_kicker = true;
          }
          if (kickers.size() < 5) {
            kickers.push_back(rank);
          }
        }
      }
    }
  }

  int result = 0;
  if (straight_flush)
    return result + straight_flush_rank;
  else {
    result += 10;
    if (quads)
      return result + group_rank<1, 1>({ quads_rank },
{ quads_kicker_rank });
    else {
      result += binom(13, 1) * binom(12, 1);
      if (trips && trips_kicker)
        return result + group_rank<1, 1>({ trips_rank },
{ trips_kicker_rank });
      else {
        result += binom(13, 1) * binom(12, 1);
        if (flush)
          return result + flush_rank;
        else {
          result += binom(13, 5) - 10;
          if (straight)
            return result + straight_rank;
          else {
```

```
            result += 10;
            if (trips)
              return result + group_rank<1, 2>({ trips_rank }, { kickers[0],
kickers[1] });
            else {
              result += binom(13, 1) * binom(12, 2);
              if (second_pair)
                return result + group_rank<2, 1>({ pair_rank,
second_pair_rank }, { pair_kicker_rank });
              else {
                result += binom(13, 2) * binom(11, 1);
                if (pair)
                  return result + group_rank<1, 3>({ pair_rank },
{ kickers[0], kickers[1], kickers[2] });
                else {
                  result += binom(13, 1) * binom(12, 3);
                  return result + five_rank({ kickers[0], kickers[1],
kickers[2], kickers[3], kickers[4] });
                }
              }
            }
          }
        }
      }
    }
  }
  return result;
}
```

ДОДАТОК Б – ТЕСТУВАННЯ ОЦІНКИ ПОКЕРНИХ КОМБІНАЦІЙ

```
#include <iostream>
#include "poker.h"

int main() {
  std::cout
  << "hands ranks:\n"
  << "  straight flush\n"
  << "    As Ks Qs Js Ts: " << hand_rank({"As","Ks","Qs","Js","Ts"}) << "\n"
  << "    5s 4s 3s 2s As: " << hand_rank({"5s","4s","3s","2s","As"}) << "\n"
  << "  four of a kind\n"
  << "    As Ad Ac Ah Ks: " << hand_rank({"As","Ad","Ac","Ah","Ks"}) << "\n"
  << "    2s 2d 2c 2h 3s: " << hand_rank({"2s","2d","2c","2h","3s"}) << "\n"
  << "  full house\n"
  << "    As Ad Ac Kh Ks: " << hand_rank({"As","Ad","Ac","Kh","Ks"}) << "\n"
  << "    2s 2d 2c 3h 3s: " << hand_rank({"2s","2d","2c","3h","3s"}) << "\n"
  << "  flush\n"
  << "    As Ks Qs Js 9s: " << hand_rank({"As","Ks","Qs","Js","9s"}) << "\n"
  << "    7s 5s 4s 3s 2s: " << hand_rank({"7s","5s","4s","3s","2s"}) << "\n"
  << "  straight\n"
  << "    As Kd Qc Jh Ts: " << hand_rank({"As","Kd","Qc","Jh","Ts"}) << "\n"
  << "    5s 4d 3c 2h As: " << hand_rank({"5s","4d","3c","2h","As"}) << "\n"
  << "  three of a kind\n"
  << "    As Ad Ac Kh Qs: " << hand_rank({"As","Ad","Ac","Kh","Qs"}) << "\n"
  << "    2s 2d 2c 3h 4s: " << hand_rank({"2s","2d","2c","3h","4s"}) << "\n"
  << "  two pair\n"
  << "    As Ad Kc Kh Qs: " << hand_rank({"As","Ad","Kc","Kh","Qs"}) << "\n"
  << "    2s 2d 3c 3h 4s: " << hand_rank({"2s","2d","3c","3h","4s"}) << "\n"
  << "  pair\n"
  << "    As Ad Kc Qh Js: " << hand_rank({"As","Ad","Kc","Qh","Js"}) << "\n"
  << "    2s 2d 3c 4h 5s: " << hand_rank({"2s","2d","3c","4h","5s"}) << "\n"
  << "  high card\n"
  << "    As Kc Qh Jd 9s: " << hand_rank({"As","Kc","Qh","Jd","9s"}) << "\n"
  << "    2s 3d 4c 5h 7s: " << hand_rank({"2s","3d","4c","5h","7s"}) << "\n";
}
```

```
hands ranks:
  straight flush
    As Ks Qs Js Ts: 0
    5s 4s 3s 2s As: 9
  four of a kind
    As Ad Ac Ah Ks: 10
    2s 2d 2c 2h 3s: 165
  full house
    As Ad Ac Kh Ks: 166
    2s 2d 2c 3h 3s: 321
  flush
    As Ks Qs Js 9s: 322
    7s 5s 4s 3s 2s: 1598
  straight
    As Kd Qc Jh Ts: 1599
    5s 4d 3c 2h As: 1608
  three of a kind
    As Ad Ac Kh Qs: 1609
    2s 2d 2c 3h 4s: 2466
  two pair
    As Ad Kc Kh Qs: 2467
    2s 2d 3c 3h 4s: 3324
  pair
    As Ad Kc Qh Js: 3325
    2s 2d 3c 4h 5s: 6184
  high card
    As Kc Qh Jd 9s: 6185
    2s 3d 4c 5h 7s: 7461
```

ДОДАТОК В – СТВОРЕННЯ ТАБЛИЦІ ПОШУКУ ЗНАЧЕНЬ *P* ДЛЯ ДВОХ ГРАВЦІВ

```
#include <iostream>
#include <fstream>
#include <numeric>
#include "poker.h"
#include "combinatorics.h"

constexpr auto ranges = ranks.size() * ranks.size();

int equityfull[deck.size()][deck.size()][deck.size()][deck.size()];
fraction<int> equity2[ranges][ranges];
int equity3[ranges][ranges][ranges];

int main() {
  using card_id_t = uint_least8_t;
  std::array<card_id_t, deck.size()> deck_id;
  std::iota(deck_id.begin(), deck_id.end(), 0);
  std::array<std::array<int, deck.size()>, deck.size()> hole_id;
  std::array<std::array<int, deck.size()>, deck.size()> range;
  int id = 0;
  for (auto hole : combinations<2>(deck_id)) {
    hole_id[hole[0]][hole[1]] = id++;
    auto card1 = deck[hole[0]], card2 = deck[hole[1]];
    range[hole[0]][hole[1]] = ((card1.rank > card2.rank) ^ (card1.suit ==
card2.suit)) ? card1.rank * 13 + card2.rank : card2.rank * 13 + card1.rank;
  }

  for (auto community_ids : combinations<5>(deck_id)) {
    std::array<card, 5> community;
    for (int i = 0; i < 5; ++i) {
      community[i] = deck[community_ids[i]];
    }

    std::array<card_id_t, deck.size() - community.size()> hole_set;
    std::set_difference(deck_id.begin(), deck_id.end(), community_ids.begin(),
community_ids.end(), hole_set.begin());

    std::array<std::array<int, 52>, 52> hole_rank;
    for (auto hole : combinations<2>(hole_set)) {
      hole_rank[hole[0]][hole[1]] = hand_rank({
        community[0],
        community[1],
        community[2],
        community[3],
        community[4],
        deck[hole[0]],
        deck[hole[1]]
      });
    }
    for (auto hole1 : combinations<2>(hole_set)) {
      auto rank1 = hole_rank[hole1[0]][hole1[1]];
      auto range1 = range[hole1[0]][hole1[1]];
      std::array<card_id_t, hole_set.size() - hole1.size()> hole2_set;
      std::set_difference(hole_set.begin(), hole_set.end(), hole1.begin(),
hole1.end(), hole2_set.begin());
      for (auto hole2 : combinations<2>(hole2_set)) {
        auto rank2 = hole_rank[hole2[0]][hole2[1]];
```

```
          auto range2 = range[hole2[0]][hole2[1]];
          auto& equity = equity2[range1][range2];
          equity.denominator += 2;
          if (rank1 < rank2) {
            equity.numerator += 2;
          }
          else if (rank1 == rank2) {
            equity.numerator += 1;
          }
        }
      }
    }


    std::ofstream file("equity2.bin", std::ios::binary);
    file.write(reinterpret_cast<char*>(&equity2), sizeof(equity2));

    return 0;
}
```

ДОДАТОК Г – ПІДРАХУНОК ВАРТОСТІ ГРИ, МАТРИЦІ ПОХІДНИХ ТА ЯКОБІАНІВ

```
template<int players = 2, typename Scalar>
void game_value(
  const parameter_t<Scalar, players> &X,
  const Scalar &e,
  std::array<parameter_t<Scalar, players>, 2> *X_prev,
  const Scalar &delta,
  value_t<Scalar, players> *value,
  derivative_t<Scalar, players> *derivative = nullptr,
  jacobian_t<Scalar, players> *jacobian = nullptr
) {
  constexpr Scalar BIG_BLIND = 1;
  constexpr Scalar SMALL_BLIND = BIG_BLIND*.5;
  constexpr Scalar JACK_POT = Scalar(757) / (50*49*48/5/4/3);
  Scalar prob;
  std::array<Scalar, players> dprob;
  std::array<decltype(dprob), players> ddprob;
  auto P = X(P_pos<players>);
  // The stack of a single player (must be >= BB)
  auto stack = P;
  std::unique_ptr<derivative_t<Scalar, players>> temp_derivative;
  if (value)
    value->setConstant(0);
  if (derivative || jacobian) {
    temp_derivative = std::make_unique<derivative_t<Scalar, players>>();
    temp_derivative->setConstant(0);
  }
  if (jacobian)
    jacobian->setConstant(0);
  nested_loops<ranges, players>([&](auto player_range) {
    //Scalar times_val = times<players>(player_range) * (1 / Scalar((52*51/2)
* (50*49/2)));
    Scalar times_val = times<players>(player_range);
    if (times_val == 0)
      return;
    for (int mask = 1, max_mask = 1 << players; mask < max_mask; ++mask) {
      prob = times_val;
      dprob.fill(times_val);
      ddprob.fill(dprob);
      Scalar dP = 0;
      Scalar pot = 0;
      std::array<int, players> betting, folding;
      int nbetting = 0, nfolding = 0;
      std::array<int, players> player_node;
      for (int node = 0, player = 0; player < players; ++player) {
        auto x = X(node * ranges + player_range[player]);
        player_node[player] = node;
        if ((mask >> player) & 1) {
          node = node * 2 + 1;
          prob *= x;
          for (int p1 = 0; p1 < players; ++p1) {
            if (p1 == player) continue;
            dprob[p1] *= x;
            for (int p2 = 0; p2 < players; ++p2) {
              if (p2 == player || p2 == p1) continue;
              ddprob[p1][p2] *= x;
            }
```

```
          }
          pot += stack;
          betting[nbetting++] = player;
        } else {
          node = node * 2 + 2;
          prob *= 1 - x;
          dprob[player] = -dprob[player];
          for (int p1 = 0; p1 < players; ++p1) {
            if (p1 == player) continue;
            dprob[p1] *= 1 - x;
            ddprob[player][p1] = -ddprob[player][p1];
            ddprob[p1][player] = -ddprob[p1][player];
            for (int p2 = 0; p2 < players; ++p2) {
              if (p2 == player || p2 == p1) continue;
              ddprob[p1][p2] *= 1 - x;
            }
          }
          if (player == players - 1)
            pot += BIG_BLIND;
          if (player == players - 2)
            pot += SMALL_BLIND;
          folding[nfolding++] = player;
        }
      }
      for (int player = 0; player < players; ++player) {
        auto id = player_node[player] * ranges + player_range[player];
        Scalar value_delta = -.05 * BIG_BLIND;
        Scalar dP = 0;
        if ((mask >> player) & 1) {
          if (nbetting > 1) {
            value_delta -= .05 * BIG_BLIND;
            auto rank1 = player_range[player] / 13;
            auto rank2 = player_range[player] % 13;
            if (rank2 == 12) {
              if (11 >= rank1 && rank1 >= 8) value_delta += 1 * JACK_POT;
              if (3 >= rank1 && rank1 >= 0) value_delta += 1 * JACK_POT;
            } else if (rank2 > rank1 && rank2 - rank1 <= 4) {
              if (rank2 == 11 && rank1 >= 9 || rank1 == 0 && rank2 <= 2) {
                value_delta += 2 * JACK_POT;
              } else if (rank2 == 10 && rank1 >= 9 || rank1 == 1 && rank2 <=
2) {
                value_delta += 3 * JACK_POT;
              } else {
                value_delta += (5 - (rank2 - rank1)) * JACK_POT;
              }
            }
          }
          if (nbetting == 1) {
            value_delta += pot - stack;
          } else {
            std::array<int, players> equity_index;
            int i = 0;
            equity_index[i++] = player_range[player];
            int ibetting = nbetting, ifolding = nfolding;
            // First go betting players
            while (ibetting--) {
              if (betting[ibetting] == player)
                continue;
              equity_index[i++] = player_range[betting[ibetting]];
            }
```

```
            // And then folding ones
            while (ifolding--) {
              equity_index[i++] = player_range[folding[ifolding]];
            }
            Scalar equity_val = equity<players>(nbetting, equity_index);
            value_delta += pot * equity_val - stack;
            dP = nbetting * equity_val - 1;
          }
        } else {
          if (player == players - 1)
            value_delta -= BIG_BLIND;
          if (player == players - 2)
            value_delta -= SMALL_BLIND;
        }
        if (dP != 0) {
          if (X_prev && jacobian) {
            jacobian->coeffRef(id, P_pos<players>) += dprob[player] * dP;
          }
        }
        if (value_delta == 0) continue;
        if (value)
          value->coeffRef(player) += prob * value_delta;
        if (derivative || jacobian)
          temp_derivative->coeffRef(id) += dprob[player] * value_delta;
        if (jacobian)
          for (int player2 = 0; player2 < players; ++player2) {
            if (player2 == player) continue;
            auto id2 = player_node[player2] * ranges + player_range[player2];
            jacobian->coeffRef(id, id2) += ddprob[player][player2] *
value_delta;
          }
      }
    }
  });
  if (derivative || jacobian) {
    *temp_derivative *= e;
    if (derivative) {
      *derivative = temp_derivative->array().atan() * Scalar(1 / EIGEN_PI) +
Scalar(.5) - X.array();
      if (!X_prev) {
        derivative->coeffRef(P_pos<players>) = 0;
      }
    }
    if (jacobian) {
      *temp_derivative = (temp_derivative->array().square() + 1).inverse() *
Scalar(e / EIGEN_PI);
      for (int i = 0; i < vars<players> - 1; ++i) {
        jacobian->row(i) *= temp_derivative->coeffRef(i);
        jacobian->coeffRef(i, i) = -1;
      }
    }
    if (X_prev) {
      auto &Xp = *X_prev;
      parameter_t<Scalar, 2> dX = Xp[0] - Xp[1];
      dX.normalize();
      if (derivative) {
        derivative->coeffRef(P_pos<players>) = (X - Xp[0]).dot(dX) - delta;
      }
      if (jacobian) {
        jacobian->row(P_pos<players>) = dX;
```

```
        }
      }
    }
  }
```