**REPORT**

**[프로젝트]**



**과 목 : 심화프로그래밍02**

**담당교수 : 윤성림 교수님**

**학 과 : 컴퓨터공학과**

**학 번 : 2021111971**

**이 름 : 이재혁**

**제 출 일 : 2024.06.18**

텍스트, 클립아트이(가) 표시된 사진

자동 생성된 설명

**프로그램 설계 및 알고리즘**

블랙잭을 진행하면서 다른 플레이어가 어떤 카드를 받았는지 모두 알 수 있어 남은 카드 개수에 따라 HIT, STAND 전략을 세웁니다.

기본적으로 현재 카드의 합에서 21을 넘지 않을 확률이 높다면 HIT, 그렇지 않다면 STAND 하는 것이 합리적입니다.

따라서 현재 나의 카드의 합, 남은 카드의 수의 정보를 알고 있어야 합니다.

따라서 Student 클래스에 아래의 정보를 멤버 변수를 추가합니다.

sum: 내가 가진 카드 숫자의 합을 저장합니다. 라운드가 종료될 때 마다 초기화합니다.

MyCount [14]: 내가 가진 카드의 개수를 저장합니다. 11로 계산되는 Ace, 1로 계산되는 Ace를 분류해서 저장합니다.

PlayingCard[13]: 전체 카드의 개수를 저장하는 배열입니다. 남은 카드에서 특정 카드가 나올 확률을 계산하기 위해 사용합니다. notifyCardReset() 매소드에서 초기화합니다.

rSize: 현재 플레잉 카드 전체의 개수를 저장합니다. 남은 카드에서 특정 카드가 나올 확률을 계산하기 위해 사용합니다. notifyCardReset() 매소드에서 초기화합니다.

dealerCard: 딜러가 공개한 카드를 저장합니다.

checkAction 구현

처음 2장의 카드 중 11 Ace를 가지고 있는 경우와 그렇지 않은 경우로 나뉩니다.

11로 계산되는 Ace를 가지고 있을 경우 21을 넘어가도 Ace가 1로 바뀌어 10만큼 감소돼 버스트를 피할 수 있습니다. 따라서 보다 과감하게 HIT를 시도할 수 있습니다.

반대로 Ace가 이미 1로 계산되거나 없는 경우, 다시 Ace가 나올 확률은 낮고, 자칫하면 바로 버스트가 발생하기 때문에 합리적으로 HIT를 시도할 필요가 있습니다.

남은 모든 카드의 개수를 알고 있기 때문에, STAND를 하기 전 버스트가 되지 않고 카드의 합이 올라가는 확률이 0.5이상이면 HIT를 시도합니다.

1) 11로 계산되는 Ace를 가지고 있을 때

카드 합 19 이상 -> STAND

카드 합 18 -> 딜러의 공개 카드의 value가 9, 10, Ace 일 때, 딜러의 나머지 한 장의 카드와의 합이 18보다 클 확률이 높기 때문에 HIT

이외에는 HIT

2) Ace가 1로 계산되거나, 없을 때

카드 합 17 이상 -> STAND

카드 합 13 ~ 16 -> 딜러의 공개 카드가 2~6일 경우 나머지 한 장의 카드와의 합이 17보다 낮을 확률이 크고, 이때 딜러가 HIT를 시도하면, 버스트가 발생할 확률이 크기 때문에 STAND.

공개 카드가 7~A인 경우 나머지 한 장의 카드와의 합이 17을 넘을 확률이 크기 때문에, HIT

카드 합 12 -> 딜러의 공개 카드가 4~6일 경우 13~16과 같은 이유로 STAND,

2, 3을 뽑으면 13~16의 범위에 들어가기 때문에 HIT

이외에는 HIT

**소스코드 및 주석**

#pragma once

#include "player.h"

//아래의 코드만 수정할것.

class Student : public Player {

private:

string name;

int sum; // 내가 가진 카드 숫자의 합을 저장할 멤버 변수

int MyCount[14]; // 내가 가진 카드의 개수를 저장하는 배열, index 0은 1로 계산되는 Ace, 1은 11로 계산되는 Ace

int PlayingCard[13]; // 전체 플레잉 카드의 개수를 저장하는 배열, 전체 카드를 생성할 때 마다 동적할당

int rSize; // 현재 카드의 개수

Card dealerCard; // 딜러가 공개한 카드

public:

Student() {

name = "2021111971 이재혁"; // 플레이어 이름 설정

sum = 0; // 내가 받은 카드의 value의 합을 저장하는 멤버 변수

for (int i = 0; i < 14; i++) {

MyCount[i] = 0;

} // 나의 카드 개수 0으로 초기화

for (int i = 0; i < 13; i++) {

PlayingCard[i] = 0;

} // 전체 플레잉 카드 개수 초기화

}

/\*

\* 어떤 행동을 할지 반환하는 매소드

\* HIT = 카드를 계속 받겠다.

\* STAND = 카드 받기를 멈추겠다.

\*/

Action checkAction() {

double nextCard = 0; // 다음에 나올 카드의 기댓값을 계산하는 변수

// 카드 개수 저장

if (MyCount[1] >= 1) { // 내 핸드에 11로 계산되는 Ace가 있을 때

if (sum >= 19) {

nextCard = 0;

for (int i = 0; i < 2; i++) {

nextCard += PlayingCard[i];

} // Ace 2 -> 20 21

nextCard /= rSize; // 남은 카드에서 위 카드가 나올 확률

if (nextCard >= 0.5) return Action::HIT; // 확률이 0.5 이상일 때 HIT

return Action::STAND; // 그렇지 않으면 STAND

}

else if (sum == 18) { // Ace와 7일 때

if (dealerCard.getValue() == 9 || dealerCard.getValue() == 10 || dealerCard.getValue() == 11 || dealerCard.getValue() == 12 || dealerCard.getValue() == 13 || dealerCard.getValue() == 1) {

// 내 패의 합이 18, 딜러가 공개한 카드가 9, 10으로 계산되는 카드, Ace일 때

// 딜러의 카드 합이 18이상일 확률은 6/13 대략 1/2

// Ace가 존재하므로 한장을 더 받았을 때 합이 올라갈 확률 3/13

return Action::HIT; // 승률을 높이기 위해 카드를 더 받는 것이 합리적

}

else { // 딜러가 공개한 카드가 낮을 때 HIT시 카드합이 올라가는 확률을 계산

nextCard = 0;

for (int i = 0; i < 3; i++) {

nextCard += PlayingCard[i];

} // Ace 2 3 -> 19 20 21

nextCard /= rSize; // 남은 카드에서 위 카드가 나올 확률

if (nextCard >= 0.5) return Action::HIT; // 확률이 0.5 이상일 때 HIT

return Action::STAND;

}

}

return Action::HIT; // 이 외에는 HIT

}

else {

// 내 핸드에 Ace가 없거나 1로 계산되는 Ace가 있을 때

// 21초과시 바로 버스트

if (sum >= 17) {

nextCard = 0;

for (int i = 0; i < 4; i++) {

nextCard += PlayingCard[i];

} // Ace 2 3 4 -> 18 19 20 21

nextCard /= rSize; // 남은 카드에서 위 카드가 나올 확률

if (nextCard >= 0.7) return Action::HIT; // 확률이 0.7 이상일 때 HIT

// Ace가 없어 21초과 시 바로 버스트이기 때문에 4개의 경우가 나올 확률이 충분히 클 때

// HIT를 시도

return Action::STAND; // 그렇지 않으면 STAND

}

else if (sum >= 13 && sum <= 16) {

if (dealerCard.getValue() >= 2 && dealerCard.getValue() <= 6) {

// 내 카드의 합이 13~16 사이일 때 딜러가 공개한 카드가 낮다면

// 딜러가 합 17이상을 맞추기 위해 카드를 여러번 뽑을 확률이 커지고,

// 버스트가 발생확률도 커진다

// 따라서 STAND하는 것이 합리적

switch (sum) {

// sum의 케이스 별로 버스트가 나지 않을 확률을 계산해

// 그 확률이 절반 이상이면 HIT한다

case 13:

nextCard = 0;

for (int i = 0; i < 8; i++) {

nextCard += PlayingCard[i];

}

nextCard /= rSize;

// 버스트가 나지 않는 카드가 나올 경우의 수

if (nextCard >= 0.5) return Action::HIT;

// 버스트가 나지 않을 경우가 0.5 이상이면 HIT

return Action::STAND; // 그렇지 않으면 STAND

case 14:

nextCard = 0;

for (int i = 0; i < 7; i++) {

nextCard += PlayingCard[i];

}

nextCard /= rSize;

// 버스트가 나지 않는 카드가 나올 경우의 수

if (nextCard >= 0.5) return Action::HIT;

// 버스트가 나지 않을 경우가 0.5 이상이면 HIT

return Action::STAND; // 그렇지 않으면 STAND

case 15:

nextCard = 0;

for (int i = 0; i < 6; i++) {

nextCard += PlayingCard[i];

}

nextCard /= rSize;

// 버스트가 나지 않는 카드가 나올 경우의 수

if (nextCard >= 0.5) return Action::HIT;

// 버스트가 나지 않을 경우가 0.5 이상이면 HIT

return Action::STAND; // 그렇지 않으면 STAND

case 16:

nextCard = 0;

for (int i = 0; i < 5; i++) {

nextCard += PlayingCard[i];

}

nextCard /= rSize;

// 버스트가 나지 않는 카드가 나올 경우의 수

if (nextCard >= 0.5) return Action::HIT;

// 버스트가 나지 않을 경우가 0.5 이상이면 HIT

return Action::STAND; // 그렇지 않으면 STAND

}

}

else return Action::HIT; // 딜러의 카드가 충분히 큰 경우 HIT

}

else if (sum == 12) {

if (dealerCard.getValue() >= 4 && dealerCard.getValue() <= 6) {

nextCard = 0;

for (int i = 0; i < 9; i++) {

nextCard += PlayingCard[i];

} // Ace ~ 9 -> 13 ~ 21

nextCard /= rSize;

if (nextCard >= 0.5) return Action::HIT;

return Action::STAND;

}

else {

return Action::HIT;

}

}

return Action::HIT; // 합이 12이하일 경우 무조건 HIT

}

}

/\*

\* 최종 결과 출력에 사용할 이름을 반환하는 매소드

\*/

string getName() {

return name;

}

/\*

\* 딜러가 받은 카드가 뭔지 알 수 있는 매소드

\*/

void notifyDealerCard(Card card) {

//cout << "딜러 카드 공개" << "(" << getName(card.getShape()) << ", " << card.getValue() << ")" << endl;

dealerCard = card; // 딜러가 공개한 카드 저장

PlayingCard[dealerCard.getValue() - 1]--; // 전체 플레이 카드에서, 딜러 카드 제거

rSize--; // 플레이 카드 개수 -1

}

/\*

\* 다른 플레이어가 받은 카드가 뭔지 알 수 있는 매소드

\*/

void notifyOtherPlayerCard(Card card) {

//cout << "다른 플레이어 카드" << "(" << getName(card.getShape()) << ", " << card.getValue() << ")" << endl;

PlayingCard[card.getValue() - 1]--; // 다른 플레이어가 받은 카드 제거

rSize--; // 플레이 카드 개수 -1

}

/\*

\* 내가 받은 카드가 뭔지 알 수 있는 매소드

\*/

void notifyMyCard(Card card) {

//cout << "받은 카드" << "(" << getName(card.getShape()) << ", " << card.getValue() << ")" << endl;

PlayingCard[card.getValue() - 1]--; // 내가 받은 카드 제거

rSize--; // 플레이 카드 개수 -1

if (card.getValue() == 1) {

sum += 11; // Ace를 11로 먼저 계산

MyCount[1]++; // 11로 계산 되는 Ace 개수 + 1

}

else if (card.getValue() >= 10) {

sum += 10; // K, Q, J 는 10으로 계산

MyCount[card.getValue()]++; // 카드 개수 + 1

}

else {

sum += card.getValue(); // value 만큼 합에 추가

MyCount[card.getValue()]++; // 카드 개수 + 1

}

while (sum > 21 && MyCount[1] > 0) {

// Ace를 갖고 있고, 합이 21이상일 때

sum -= 10; // Ace를 1로 계산

MyCount[1]--; // 11로 계산되는 Ace - 1

MyCount[0]++; // 1로 계산되는 Ace + 1

}

}

/\*

\* 딜러가 사용하는 카드 더미를 새로 만들어낼때

\* 몇개의 플레잉 카드로 만들어졌는지 알려주는 매소드

\*/

void notifyCardReset(int cardDeck) {

//cout << "--- 플레잉 카드 초기화 ---" << endl;

for (int i = 0; i < 13; i++) {

PlayingCard[i] = 4 \* cardDeck;

} // 카드문양 \* 포커 덱의 개수 = 숫자 개수

rSize = 4 \* cardDeck \* 13; // 13개의 숫자 존재

}

/\*

\* 한번의 라운드가 끝났음을 알려주는 매소드

\*/

void notifyCompletedRound() {

//cout << "---------- 라운드 종료 ----------" << endl;

sum = 0; // 합 초기화

}

/\*

\* 카드의 모양을 문자열로 리턴하는 메소드

\*/

string getName(Shape shape) {

switch (shape) {

case 0:

return "SPADE";

case 1:

return "HEART";

case 2:

return "DIAMOND";

case 3:

return "CLOVER";

}

}

};

**결과 및 결과분석**

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

내가 받은 카드, 다른 플레이어가 받은 카드, 딜러의 카드는 1장만 먼저 오픈 하는 것을 확인할 수 있습니다.

텍스트, 스크린샷, 블랙, 흑백이(가) 표시된 사진

자동 생성된 설명

checkAction 알고리즘은 43.2%의 승률을 기록하는 것을 확인할 수 있습니다.

**소감**

주어진 과제를 해결하기 위해 제공해주신 코드를 이해하는데 가장 많은 시간을 할애한 것 같습니다.

Readme 파일을 참고했지만, 변수, 함수명을 확인하고 어디서 어떻게 동작하는지 이해하기 위해 코드를 많이 읽었습니다.

실무에서는 다른 사람들과 협업으로 작업하는 것으로 알고 있습니다. 다른 사람의 코드를 이해하고, 내 코드를 작성하는 것을 조금이나마 연습할 수 있던 시간이었습니다.

교수님이 강조하신 함수나 변수명에 의미를 부여하는 것이 코드를 이해하는데 큰 역할을 했던 것 같습니다.

제공해주신 코드를 확인하며, 클래스의 구현 부, 선언 부의 분리, vector container의 사용법 등 C++ 프로그래밍의 응용을 확인할 수 있었습니다. 객체지향 언어인 C++이 C언어보다 코드를 이해하고 다루기 더 편하다고 느꼈습니다. C언어로 공부한 자료구조를 C++로 구현하면 더 쉽게 다룰 수 있을 것 같다는 생각이 들었습니다.