2021 Spring OOP Assignment Report

과제 번호 : assn3
학번 : 20200725
이름 : 윤승우
Povis ID : ysw1110

명예서약 (Honor Code)

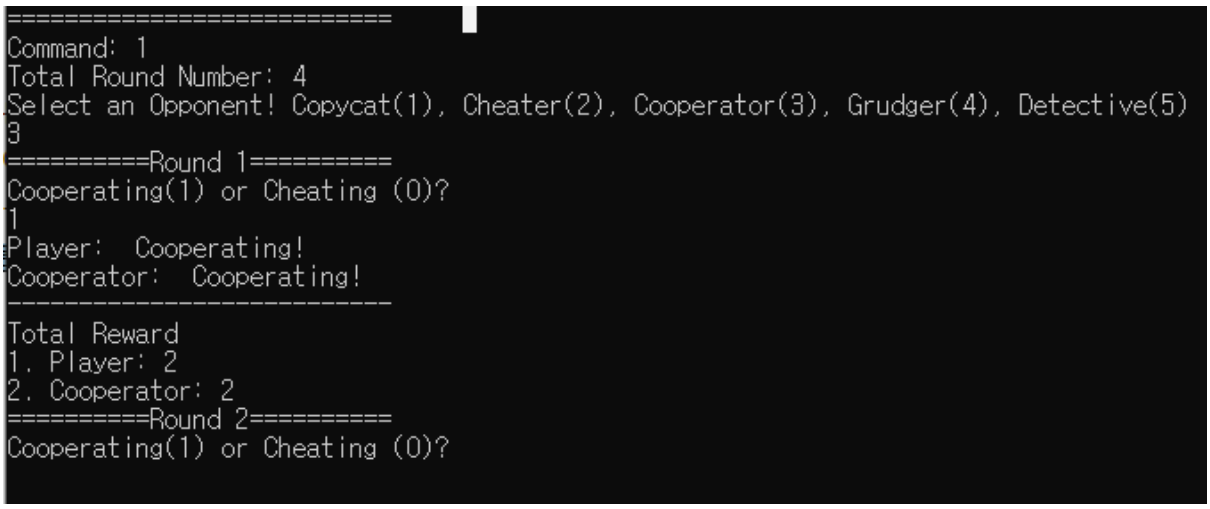
나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

1. 프로그램 개요

□ 신뢰의 진화 게임을 구현했다.

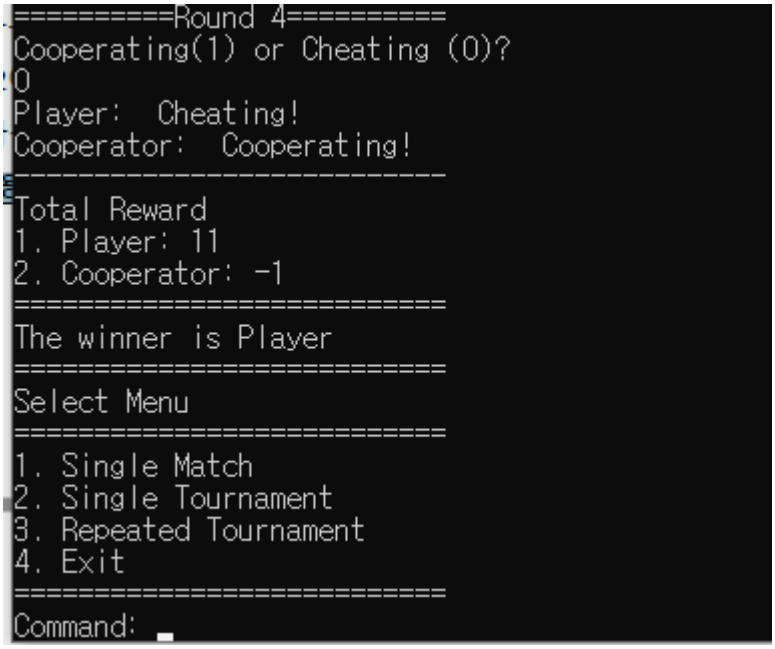
프로그램을 실행하면 메뉴화면이 나온다.



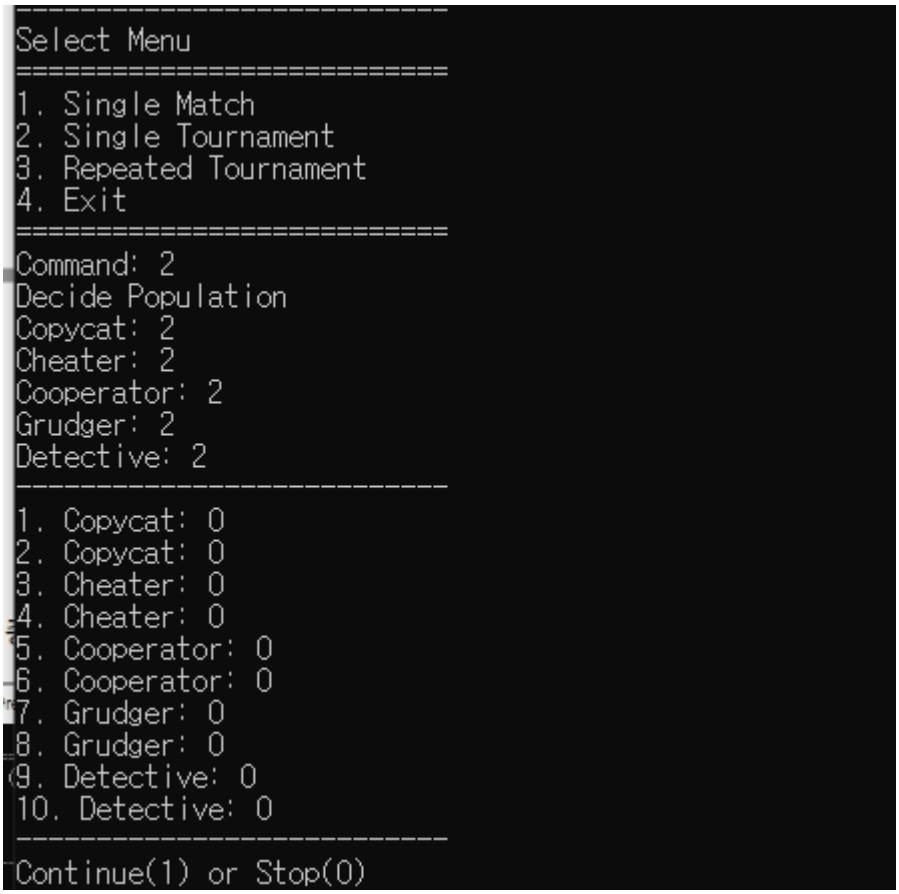
1번을 누르면 single match 실행한다. 플레이어는 total round 수를 입력하고 상대의 직업

을 선택할 수 있다. 플레이어는 cooperating(1) 혹은 cheating(0)을 선택한다.

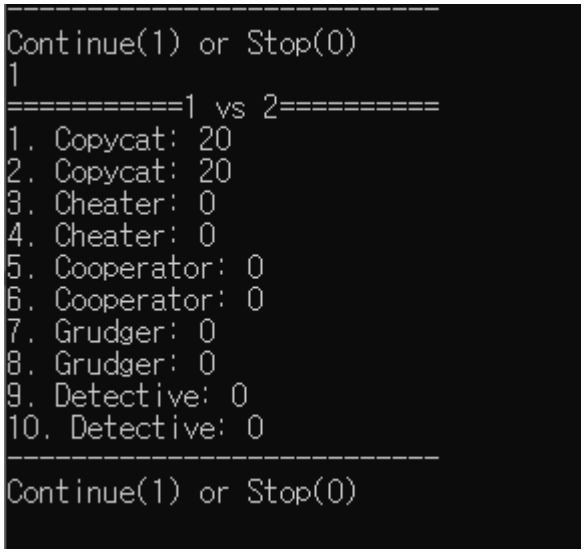
round진행 후 플레이어와 상대의 점수가 출력된다.



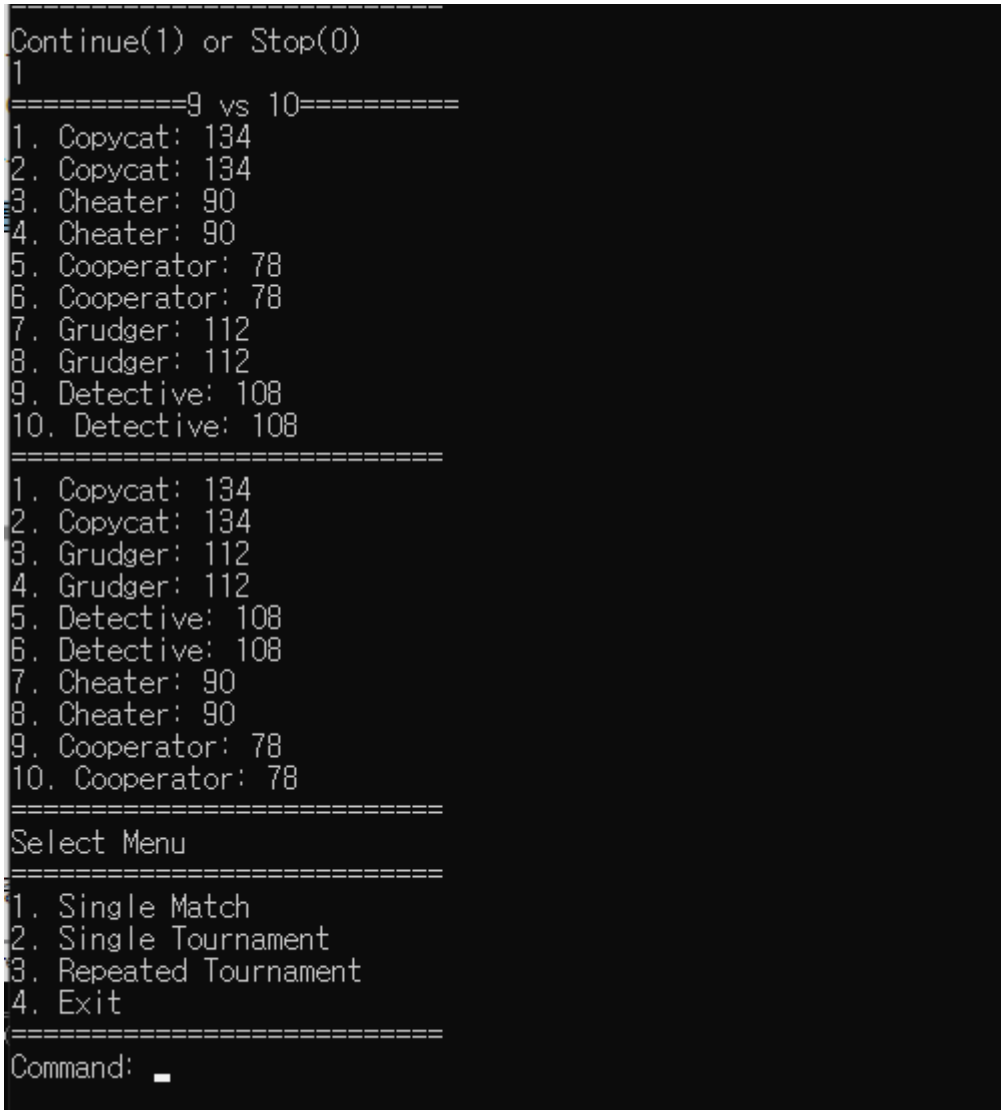
마지막 라운들르 실행하면 승자를 출력하고 시작메뉴로 되돌아간다.



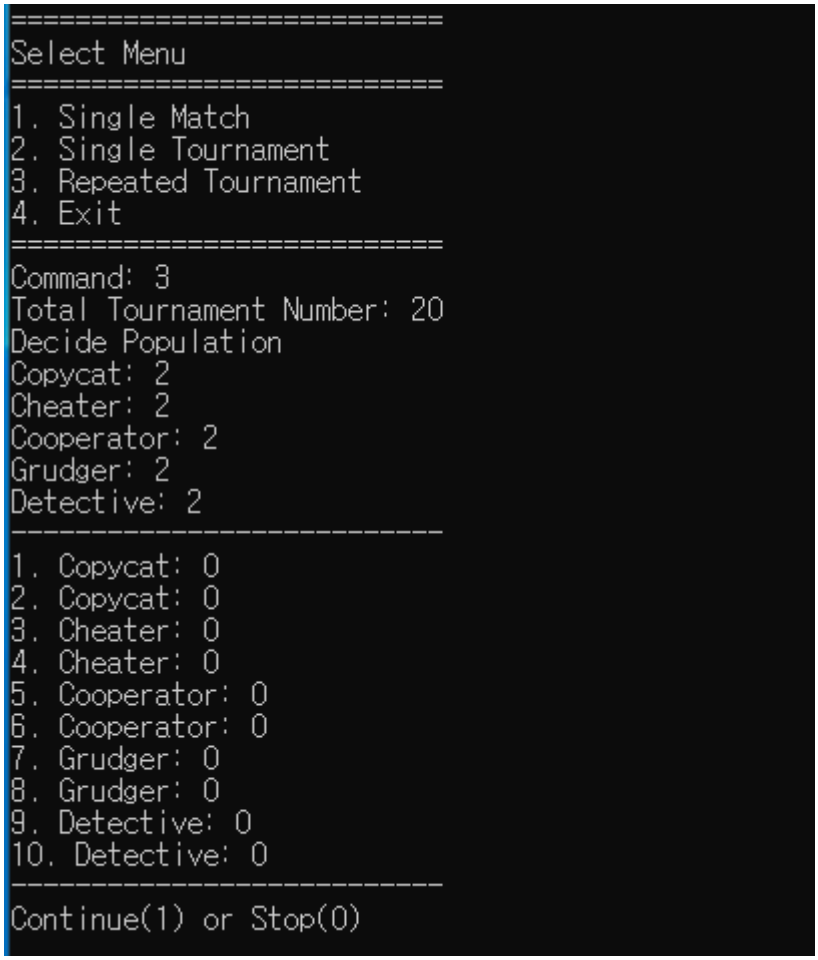
메뉴의 2번은 single tournament이다. Single tournament를 실행하면 토너먼트를 실행시킬 각 직업들의 수를 입력받는다. 입력을 완료하면 각 직업의 reward를 일단 내림차순으로 출력한다. 1을 눌러 continue하면 tournament를 계속하고, 0을 누르면 시작메뉴로 되돌아간다.



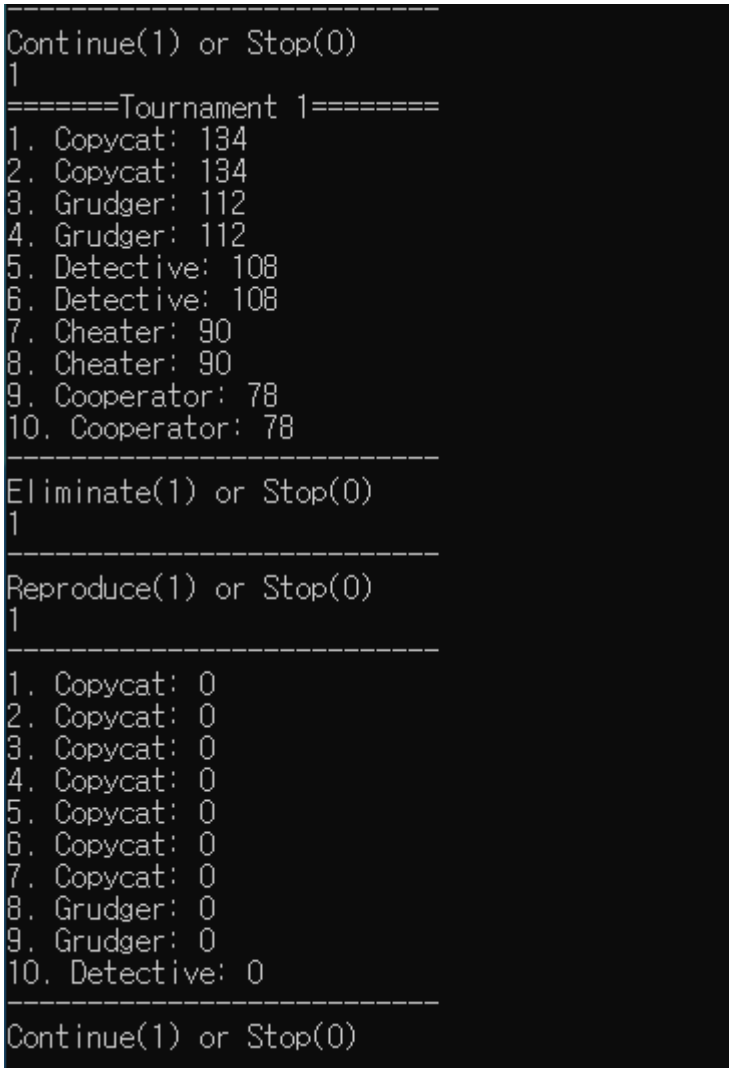
토너먼트를 실행시키면 몇번끼리 토너먼트를 돌렸는지 첨자로 표시된다. 그리고 토너먼트의 실행결과(reward)를 출력한다.



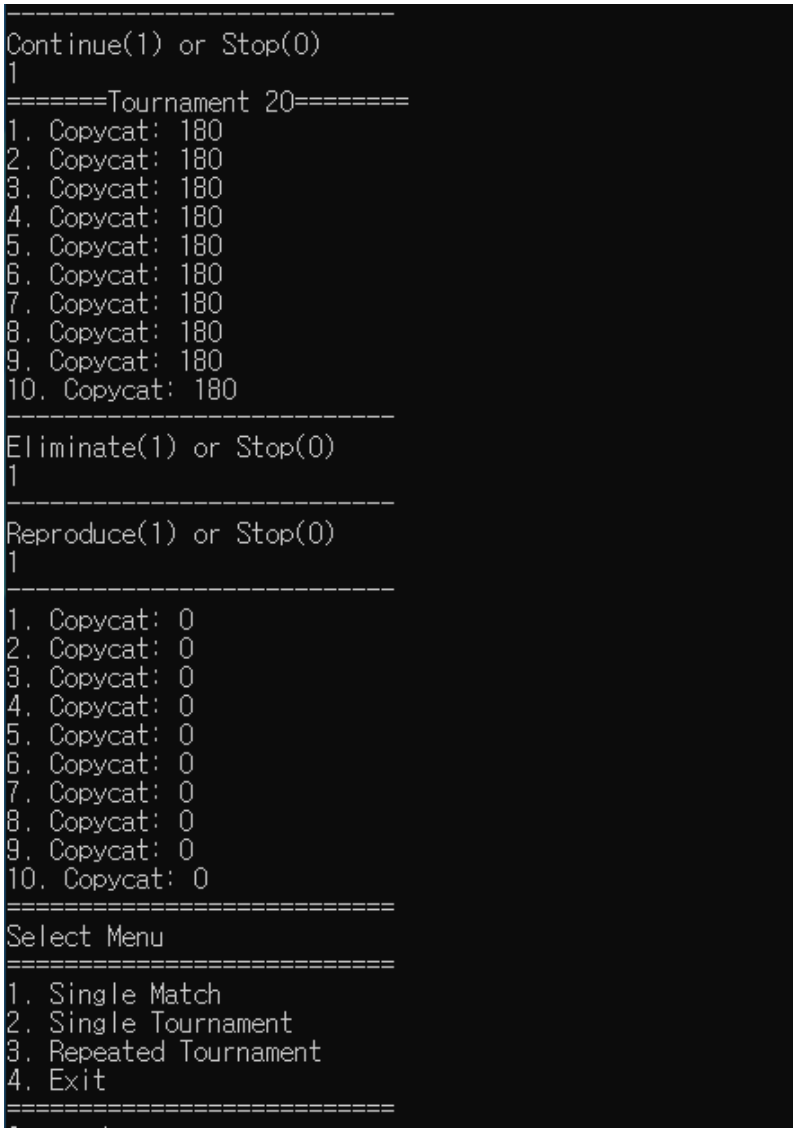
토너먼트의 마지막 match를 실행하고 나면 점수를 내림차순으로 정렬하여 출력하고 시작메뉴로 되돌아간다.



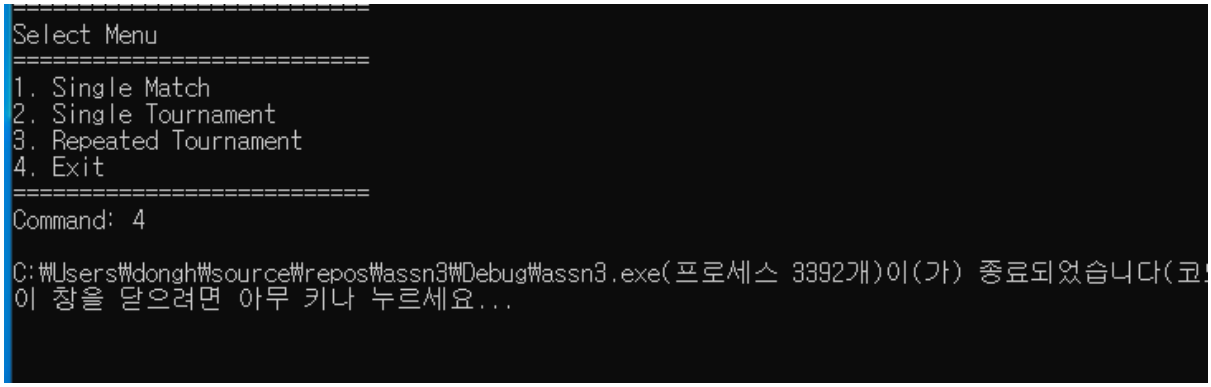
시작 메뉴에서 3번을 누르면 Repeated tournament를 실행한다. 총 tornament 입력과 repeated tournament를 진행시킬 직업들의 수를 입력받는다. 그 이후 각 직업들의 reward를 출력한다.



Repeated tournament는 3단계의 과정으로 이루어진다. Continue를 하면 tournament를 진행시켜 점수들을 내림차순으로 출력한다. 그 이후 eliminate를 거치면 하위 5명을 삭제시킨다. Reproduce를 하면 1등의 직업을 복사하여 5개 생성하고 직업 목록을 출력한다.



마지막 tournament가 끝나면 시작 메뉴로 되돌아간다. Stop을 누르면 얼마든지 종료하고 시작메뉴로 되돌아갈 수 있다.



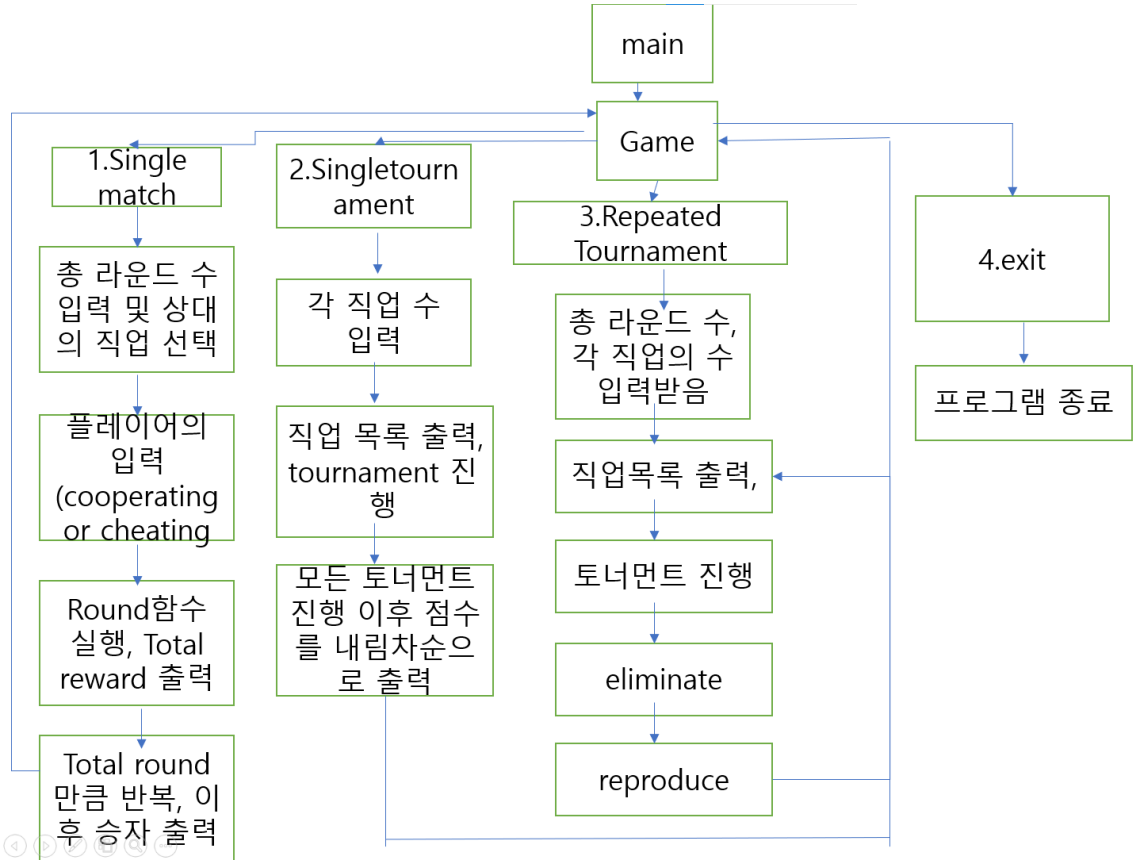
시작 메뉴의 4번을 누르면 프로그램을 종료시킨다.

- 프로그램 디렉토리의, 헤더파일들은 각각 헤더파일의 이름에 해당하는 클래스의 선언이 들어있는 파일들이다. main.cpp를 제외한 cpp파일들은 파일이름에 해당하는 클래스의 구현부분이 들어있다.

2. 프로그램의 구조 및 알고리즘

- Agent클래스: 각 직업들의 부모 클래스
- Cheater클래스: Cheater직업을 구현
- Cooperator클래스: Cooperator직업을 구현
- Copyycat클래스: Copycat직업을 구현
- Detective클래스: Detective직업을 구현
- DoublyLinkedList클래스: Tournament, repeated Tournament에서 이용할, Agent들을 저장하는 자료구조, sorted doubly linked list이다. 각 노드는 저장할 데이터와 더불어 이전 노드와 다음 노드에 대한 포인터를 가지고 있다. 그리고 add로 노드를 추가할 때 자동으로 reward순으로 내림차순으로 삽입되게 한다. Population class가 상속하게 될 자료구조 클래스
- Game클래스: 시작메뉴, single tournament, repeated tournament, single match를 진행시키는, 게임을 총괄하는 클래스
- Grudger클래스:Grudger 직업을 구현하는 클래스
- Iterator클래스: Doubly Linked List에서 이용할, 임의의 노드에 접근하게끔 도와주는 클래스
- Match클래스: Match를 담당,
- Player클래스: single match에서의 Player직업을 담당
- Population클래스: 자료구조 DoyblyLinkedList를 상속받아 repeated tournament에서의 reproduce, eliminate 등, agent를 관리하는 기능을 하는 클래스
- Round클래스: round를 담당하는 클래스
- Tournament클래스: tournament를 담당하는 클래스

- 알고리즘:



.

변수 설명

Agent.h:

int reward=0; //점수, default로 0
int history; //이전에 했던 행동
int state;//cheating을 했는지, 협력을 했는지 표시
//0이면 cheating 1이면 협력
int id; //직업을 표시

Detective.h

int cheathistory = 0;//상대방이 배신한 적이 없다면 0을 유지
int identity; //4라운드 이후 detective의 행동방향 결정

DoublyLinkedList.cpp

Iterator iter(*this); //임의의 노드에의 접근을 도와주는 iterator변수

NodeType* temp = new NodeType; //추가하려는 노드 한 개 생성, or 삭제하려는 노드의 주소를 임시로 저장하는 용도의 변수

DoublyLinkedList.h

NodeType* pFirst; //첫 노드의 주소
NodeType* pLast; //마지막 노드의 주소
int length; //리스트의 크기

Game.cpp

void Game::singleMatch() {
int TNR; //singleMAtch할 때의 Total Round Number
int Opponent; //상대 직업의 id를 나타내는 변수
int select; //cooperate인지cheat인지 입력받음

Player p; //플레이어 object
Agent* pOppo; //Opponent가 될 Agent를 가리키는 포인터

Round r(&p, pOppo); //라운드 오브젝트

}

void Game::singleTournament() {
Population p; //population 오브젝트

int nCopycat, nCheater, nCooperator, nGrudger, nDetective; //각 직업들의 개수

Tournament t(&p); //tournament 오브젝트

int selectinput; // continue, stop할지 입력

}

void Game::repeatedTournament() {
Population p; //population 오브젝트
int nCopycat, nCheater, nCooperator, nGrudger, nDetective; //각 직업들의 개수
int totalNumber;// 총 토너먼트 횟수

Tournament t(&p); //tournament 오브젝트

int selectinput;
int selectinput2;
int selectinput3; //각각 tournament 진행, eliminate 진행, reproduce 진행의 입력을 받음

}

void Game::runGame()
{
int reinput;// reinput=0이면 정상 입력, 1이면 잘못된 입력, 다시 입력받음
}

Game.cpp

int selectMenu;//메뉴화면에서 뭘 선택했는지(command)

Grudger.h

int cheathistory=0;//상대방이 배신한 적이 없다면 0을 유지

Iterator.h

const DoublyLinkedList& dList; //iterator를 작동시킬 링크드 리스트
NodeType* pCurPointer; //iterator가 현재 가리키는 노드의 주소

main.cpp

Game g1; //게임 object

Match.h

int roundLimit; //진행할 라운드 개수
int curRound = 1;//현재 라운드 수
int winner=1;// 1이면 ptr1이 가리키는 agent가 승자,2이면 ptr2가 가리키는 agent가 승자
Agent* ptr1;
Agent* ptr2; //match를 돌릴 두 agent를 각각 가리키는 포인터

Player.h

int input; //player의 입력

Population.cpp

void Population::Reproduce() {
Agent** newarr = new Agent * [5]; //새로 생성할 Agent의 주소들을 담을 배열
}
void Population::showReward() {
NodeType* pointer=getPFirst();//현재 출력할 노드를 가리키는 포인터, 맨 처음에는 first로 초기화시킴
}
void Population::sort() {
Agent** newarr = new Agent * [getLength()]; //Agent의 포인터를 담는 배열
NodeType* ptr = getPFirst(); //노드에 접근하기 위한 포인터

void Population::resetAll() { //모든 agent들의 내용물(reward포함)을 0으로 초기화
NodeType* pointer = getPFirst();//현재 reset시킬 노드를 가리키는 포인터
}

void Population::resetAll_without_reward(){//reward를 제외한 history, identity등을 초기화
NodeType* pointer = getPFirst();//현재 reset시킬 노드를 가리키는 포인터
}

Population.h

```
Agent** arr; //Agent*들의 배열의 시작주소, 자세한건 Alloc구현부분 참고
int nCopycat; //각 직업들의 개수
int nCheater;
int nCooperator;
int nGrudger;
int nDetective;
```

Round.h

```
int curRound=1;//현재 라운드 수
int history1;
int history2;//라운드를 돌리는 두 agent들의 history
Agent* ptr1;
Agent* ptr2; //match를 돌릴 두 agent의 포인터
```

Tournament.h

```
int roundT = 1; //현재 몇 번째 Tournament인지
Match m;
int i = 0;
int j=1;//다음에 진행될 match에 참가하는 AGent의 첨자들.
Population* pPopulation; //토너먼트에 참가하게 될 녀석의 데이터들
```

3. 토론 및 개선

- 동적할당을 Switch case문을 이용하여 하고, 그 이후에 동적할당받은 포인터를 이용하려고 하면 컴파일 에러가 난다. 반드시 switch-case문에 default 구문을 집어넣어줘야 컴파일 에러가 나오지 않는다.
- 클래스를 설계할 때, 공통되는 겹치는 부분이 많으면 그 내용물들을 따로 base class로 만들어 놓고, derived class가 그 내용물들을 상속받는 방식으로 설계하면 코드도 줄일 수 있고 설계하기도 편리하다.
- Virtual 함수의 경우, derived class에서 반드시 재정의해줘야 한다. 그러지 않으면 컴파일 에러가 난다.
- Linked list 자료 구조를 이용할 때는 배열과 다르게 정렬하기가 힘들다. 그래서 정렬을 하고싶다면, add할 때 자동으로 정렬된 위치를 찾아서 노드를 삽입하도록 하고, 정렬할 때는 그냥 노드들을 삭제했다가 다시 추가하는 방식으로 한다.
- 다른 직업들을 추가할 수 있을 것 같다. 그리고 실제 게임 이론 모델을 위해서라면, 직업들이 올바른 선택을 하지 않고 실수의 선택을 할 확률도 포함하면 좋을 것 같다.

- Single match에서 랜덤 모드를 추가하여 상대방의 직업을 모르는 상태에서 최고점을 얻기 위해 플레이어가 노력하는 모드를 추가하면 좋을 것 같다.

4. 참고 문헌

- 본인이 제출했던 assn2에서 doubly linked list와 iterator의 코드를 참고함.