2021 Spring OOP Assignment Report

과제 번호 : Programming Assignment #3
학번 : 20200437
이름 : 김채현
Povis ID : kch3481

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

본 프로그램을 간략히 설명하면 다음과 같다.

- 사용자는 시작 메뉴에서 Single Match, Single Tournament, Repeated Tournament 중 하나를 선택하여 게임을 시작한다.
- Single Match의 경우 사용자가 진행할 Round 수를 입력하고 각 Round마다 행동을 선택한다. Single Tournament의 경우 사용자가 직업 별 인원을 입력하고 각 직업들끼리 10Round로 이루어진 Match를 진행한다. Repeated Tournament의 경우 사용자가 직업 별 인원과 진행할 Tournament를 입력하고 각 Tournament에 대해 Single Tournament와 같은 방식으로 게임을 진행한다. 이후 하위 5명을 탈락시키고 점수 1위를 5명 재생산한다. 게임이 끝나며 시작 메뉴로 돌아간다.
- Single Match를 제외한 경우 게임 중간에 시작 메뉴로 돌아갈 수 있으며 메뉴에서 4를 입력받으면 프로그램을 종료한다.

2. 프로그램 구조 및 알고리즘

■ Class에 대한 설명

1) LinkedList Class

먼저, 구현한 Linked List는 맨 처음 node와 맨 마지막 node는 유의미한 값을 가지지 않고 node를 이어주는 역할만 하는 일명 dummy node이다. 따라서 list가 비었다는 의미는 맨 처음 node와 맨 마지막 node만 존재하고 데이터를 가진 node는 존재하지 않는다는 의미로 구현하였다. 리스트를 빈 리스트로 만들 때도 맨 처음 node와 맨 마지막 node를 제외한 데이터를 가진 node 만을 삭제한다.

(1) NodeType

먼저 LinkedList의 node가 되는 struct NodeType는

① 이전 노드를 가리키는 pointer NodeType *prev

② Population에서 Agent들을 저장하는 용도로 사용되는 Agent* character
③ Match에서 Round 결과를 저장하는 용도와 Agent에서 행동을 저장하는 용도로 사용되는 int data
④ 다음 노드를 가리키는 pointer NodeType *next
로 이루어져 있다.

(2) member
list의 맨 처음 노드를 가리키는 포인터 NodeType *pFirst, list의 맨 마지막 노드를 가리키는 포인터 NodeType *pLast, 리스트에 저장된 node의 수(dummy node 제외) int length가 존재한다.

(3) method
① 생성자와 소멸자
리스트 생성시 맨 처음과 마지막에 들어가는 dummy node가 생성되어야 한다. 그렇기에 pFirst와 pLast에 동적할당을 통해 NodeType을 생성해주고 두 노드를 이어준다. length는 0으로 저장해준다. 소멸 시에도 동적할당 해준 pFirst와 pLast를 해체해준다.
② getter, setter
member에 대해 외부에서도 접근하게 해준다.
③ Add
Match에서 Round 결과를 저장하는 용도와 Agent에서 행동을 저장할 때 parameter로 int형 자료를 받아와 int data를 가진 node를 생성하고(character은 NULL로 저장) 마지막 dummy node 바로 앞에 삽입한다. length는 +1을 해준다.
④ reset
맨 처음과 마지막 dummy node를 제외한 node들을 삭제하여 LinkedList를 비운다.

2) Agent Class
(1) member
각 Round에서 행한 행동을 저장하는 LinkedList를 가리키는 LinkedList* history, 현재 Round에서 행하는 행동을 저장하는 int action (1,0으로 저장), total reward를 저장하는 int reward가 있다.

(2) method
① 생성자와 소멸자
history가 가리키는 LinkedList를 생성한다. 생성된 LinkedList의 dummy node의 data에는 -1을 저장한다. history가 가리키는 LinkedList의 node의 data에는 각 Round에서 행한 행동을 저장하는데 이는 유의미한 값으로 1, 0을 가지기 때문에 무의미한 값이라는 뜻으로 -1을 저장해준다. 같은 맥락으로 action에도 -1으로 초기화한다. reward에는 0을 초기화한다. 소멸 시 동적할당 해준 history를 해제해준다.

② getter, setter

member에 대해 외부에서도 접근하게 해준다. 단, history에 대한 setter함수로는 history가 가리키는 LinkedList에 action을 저장한 node를 추가하는 set_history_action이 존재한다.

③ reset_history

history를 비운다.

④ print_action

int형으로 저장된 member action값이 1이면 "Cooperating!"으로, 0이면 "Cheating!"으로 출력해준다.

⑤ act

pure virtual 함수이다.

3) Player, Copycat, Cheater, Cooperator, Grudger, Detective의 act function

다들 Agent class를 상속받는다.

(1) Player

사용자에게 입력을 받아 action에 저장한다.

(2) Copycat

parameter로 받아온 상대방의 history가 가리키는 LinkedList에서 마지막으로 행한 행동, 즉 마지막 node에 있는 data를 가져와 action에 저장한다.

(3) Cheater

상대방의 history에 상관없이 항상 action에 0을 저장한다.

(4) Cooperator

상대방의 history에 상관없이 항상 action에 1을 저장한다.

(5) Grudger

상대방이 한 번이라도 배신을 했는지 저장하기 위해 int check_op라는 member를 선언한다. 젤 처음에는, 즉 상대방의 history에 아무 node도 없을 때 action에 1을 저장하고 check_op에 1을 저장한다. 이후 check_op가 1일 때 parameter로 받아온 상대방의 history가 가리키는 LinkedList에서 마지막 node에 있는 data가 1이면 action에 1을 저장, 0이면 action에 0을 저장하고 check_op값을 0으로 바꾼다. check_op가 0이면 action에 0을 저장한다.

(6) Detective

Grudger과 같은 맥락으로 int check_op라는 member를 선언한다. 상대방의 history가 가리키는 LinkedList에서 마지막 node에 있는 data가 0이면 action에 0을 저장하고 check_op값을 0으로 바꾼다. 협력-배신-협력-협력으로 시작하므로 history의 LinkedList length가 0, 1, 2, 3일 때 action에 각각 1, 0, 1, 1을 저장한다. length가 4 이상부터는 check_op가 0이면 Copycat과 같은 알고리즘으로, 1이면 Cheater와 같은 알고리즘으로 action을 저장한다.

4) Population Class

LinkedList class를 상속받는다.

(1) member

상속받는 LinkedList class의 member를 제외하고 Population class만의 member는 따로 없다.

(2) method

① 생성자와 소멸자

별 다른 특이점은 없다.

② Add

parameter로 직업을 나타내는 숫자인 character_sort(1-Copycat, 2-Cheater, 3-Cooperator, 4-Grudger, 5-Detective)와 그 직업의 인원인 character_number을 받는다. character_sort 값에 따라 각 직업 class 객체를 가리키는 Agent*를 가진 node를 생성한다. node의 data에는 각 직업을 나타내는 숫자를 저장한다. 생성된 node를 맨 마지막 dummy node와 그 앞 node 사이에 이어주고 length를 +1 해준다. 반복문을 통해 character_number만큼 반복한다.

③ Eliminate

마지막 dummy node 앞에 있는 node를 삭제한다. 5회 반복하여 결과적으로 뒤에서부터 5개의 node를 삭제한다.

④ Reproduce

젤 앞 dummy node 다음의 node와 동일한 직업을 가리키는 node를 생성하여 그 다음에 삽입한다. 5회 반복한다. 모든 node의 Agent의 reward는 0으로 바꿔준다.

⑤ sort

선택 정렬 알고리즘을 이용해 LinkedList의 node를 내림차순으로 정렬한다.

⑥ swap

parameter로 받은 두 Iterator가 각각 가리키는 node의 character, data를 바꾼다. sort 과정에서 필요한 함수이다.

⑦ print_Tot_Reward_single

Single Match에서 Total Reward를 출력한다.

⑧ print_Tot_Reward

1.Single Match가 아닌 Tournament과정에서 이루어지는 regular Match에서 Total Reward를 출력한다.

5) Round Class

(1) method

① 생성자와 소멸자

별 다른 특이점은 없다.

② play_round

round를 진행하는 함수이다. parameter로 Round를 하는 Agent* a, Agent* b 두 개를 받아온다.

각 Agent에 상대방의 history를 넘겨주어 action을 결정하게 하고 이후 자신의 history에 action을 저장, 추가한다.

6) Match Class

(1) member

Match를 진행하는 두 직업에 대한 Round 결과를 LinkedList로 저장하기 위해 LinkedList* a_history, LinkedList* b_history가 있다.

(2) method

① 생성자와 소멸자

a_history, b_history가 가리키는 LinkedList를 생성한다. 소멸 시 동적할당 해준 history를 해제해준다.

② single_Match

게임 중 1. Single Match에서 사용되는 함수이다. parameter로 진행할 Round의 수인 int Round_number, Match를 진행하는 두 캐릭터인 Agent* a, Agent* b, b의 직업을 나타내는 숫자인 int character을 받는다. single Match의 특성상 Agent* a는 player로 고정되어 있다. 각 Round에서 player와 상대방의 행동과 Total reward를 출력하고, Match가 끝난 후에는 승자를 출력한다.

③ regular_Match

Tournament에서 하는 Match를 진행하는 함수이다. parameter로 Match를 진행하는 두 캐릭터인 Agent* a, Agent* b를 받아오고, 10회 Round를 진행한다. 각 Round의 결과는 Add_Round_data를 통해 history에 저장한다. Match가 끝나면 history를 reset한다.

④ Add_Round_data

parameter로 받아온 Agent* a, Agent* b의 action에 따라 결과를 a, b의 reward에 더해주고 history에도 저장해준다.

7) Tournament Class

(1) method

① 생성자와 소멸자

별 다른 특이점은 없다.

② single_Tournament

게임 중 2. Single Tournament에서 사용되는 함수이다. Population의 pointer를 parameter로 받아와 리그형식으로 Population의 node의 Agent*를 parameter로 보내 하나씩 Match를 진행시킨다. 각 Match마다 끝난 후 Total reward를 출력한 후 계속 진행할 것인지 여부를 입력받는다.

③ regular_Tournament

게임 중 3. Repeated Tournament에서 Tournament를 진행하는 함수이다. Single Match와 같은

알고리즘이나 각 Match가 끝난 후 Total reward를 출력하지 않고 계속 진행할 것인지 여부를 입력 받지 않는다.

8) Game Class

(1) member

사용자로부터 입력 받는 Command를 저장하는 변수 int Command가 있다.

(1) method

① 생성자와 소멸자

생성자에서 Command를 0으로 초기화해준다. 별 다른 특이점은 없다.

② print_menuscreen

메뉴 화면을 출력하고 사용자로부터 Command를 입력 받아 Command에 저장한다. 1, 2, 3, 4 외의 입력은 다시 입력받는다.

③ play_game

게임을 진행하는 함수이다. Command가 1인 경우 Single Match를 진행한다. 진행할 Round의 수를 입력 받고 상대방 직업을 입력 받아 Population의 LinkedList에 추가해준다. single_Match 함수를 호출하여 게임을 진행한다. 게임이 끝난 후에는 return 1을 해준다.

Command가 2인 경우 Single Tournament를 진행한다. 각 직업 별로 인원을 입력 받아 Population의 LinkedList에 추가해준다. 게임을 시작하기 전 Total reward를 출력해주고 계속 진행할 것인지 여부를 입력 받는다. single_Tournament 함수를 호출하여 게임을 진행하고 Tournament가 끝난 후 Total reward를 내림차순으로 sort 해주고 출력해준다. 게임이 끝난 후 혹은 계속 진행하지 않으면 return 1을 해준다.

Command가 3인 경우 진행할 Tournament의 수를 입력 받고 상대방의 직업을 입력 받아 Population의 LinkedList에 추가해준다. 게임을 시작하기 전 Total reward를 출력해주고 계속 진행할 것인지 여부를 입력 받는다. regular_Tournament 함수를 호출하여 게임을 진행하고 Tournament가 끝난 후 Total reward를 내림차순으로 sort 해주고 출력해준다. 하위 5명 제거와 재생산 여부를 입력 받고 계속 진행할 것인지 여부를 입력 받는다. 제거나 재생산을 하지 않거나 계속 진행하지 않는 경우 return 1을 해준다. 게임이 끝난 후에는 return 1을 해준다.

Command가 4인 경우 return 0을 해준다.

④ Decide_Population

각 직업별로 인원을 입력 받은 후 parameter로 받아온 Population의 LinkedList에 추가해주는 함수이다.

9) Iterator Class

Assignment2의 problem 3의 Iterator Class와 같은 맥락으로 LinkedList에 순차적으로 접근하는 기능을 가진 class이다.

■ 전체적인 알고리즘 설명

본 프로그램에서 Tournament에서는 Match를 불러오고 Match에서는 Round를 불러오는 형식으로 이루어져 있다. 하지만 프로그램 출력 특성 상 Single Match나 Tournament에서와 일반적으로 반복되는 Match와 Tournament에서 차이가 있기 때문에 class 내에서 method를 구분하였다. LinkedList는 총 세 군데에서 사용된다. Population에서 Agent를 관리하는 용도, Agent에서 action을 저장하는 history 용도, Match에서 Round를 저장하는 history 용도로 사용된다. main 함수에서는 play_game 함수의 return 값이 0이 아닐 경우 메뉴화면을 출력하고 게임을 하는 과정을 반복한다. Command로 4를 입력 받았을 경우에만 play_game함수에서 return 0을 하고 프로그램을 종료한다.

3) 토론 및 개선

본 프로그램은 "신뢰의 진화" 게임을 설계하고 구현하는 프로그램이다. 본 프로그램을 구현하는 과정에서 어려웠던 점은 어떤 Class에 어떤 Class를 사용해야 하는가 같은 포함관계가 어려웠던 것 같다. 이 외에 아쉬운 점은 Single Match나 Tournament에서와 일반적으로 반복되는 Match와 Tournament를 따로 함수로 구현해주었는데 통일된 함수로 구현해주었으면 좋았겠다는 아쉬움이 있다. 또한 Match나 Tournament에서 계속 진행할 것인지를 입력 받는 것을 함수로 구현해주었다면 코드에 반복되는 부분을 제거할 수 있었을 것이라는 생각이 든다. 이 외에도 한 게임이 끝나면 Population의 LinkedList의 node를 다 삭제해주어 동적할당을 없앴어야 하는데 그러지 못해 아쉬움이 남는다. 그럼에도 이번 과제는 Inheritance와 Pure virtual function, Abstract Class 등을 익힐 수 있는 좋은 기회였다.