[Unity3D] 고스톱 게임 만들기 - 3 (카드 선택 로직 구현)

**2.3 카드 선택**

카드 분배 이후에는 선이 되는 플레이어가 낼 카드를 선택하는 과정입니다.

이 부분은 CGostopEngine클래스의 player\_put\_card매소드에 구현되어 있습니다.

그런데 이 매소드가 어떤 과정을 거쳐 호출되는지 아직 알려드리지 않았습니다.

사실 서버측에는 CGostopEngine클래스 말고 CGameRoom이라는 클래스가 존재하는데

이 클래스에서 플레이어의 요청을 받고 CGostopEngine클래스로 넘겨주는 역할을 합니다.

이 부분은 고스톱 게임의 로직과는 관계 없으며 프로젝트의 설계와 관계 있는 부분이라

이 장에서는 언급을 하지 않도록 하겠습니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 플레이어가 카드를 낼 때 호출된다.
3. /// </summary>
4. /// <param name="player\_index"></param>
5. /// <param name="card\_number"></param>
6. /// <param name="pae\_type"></param>
7. /// <param name="position"></param>
8. /// <returns></returns>
9. public PLAYER\_SELECT\_CARD\_RESULT player\_put\_card(byte player\_index,
10. byte card\_number,
11. PAE\_TYPE pae\_type,
12. byte position,
13. byte slot\_index,
14. byte is\_shaking)
15. {
16. this.selected\_slot\_index = slot\_index;
18. //UnityEngine.Debug.Log(string.Format("recv {0}, {1}, {2}",
19. //  card\_number, pae\_type, position));
21. // 클라이언트가 보내온 카드 정보가 실제로 플레이어가 들고 있는 카드인지 확인한다.
22. CCard card = this.player\_agents[player\_index].pop\_card\_from\_hand(
23. card\_number, pae\_type, position);
24. if (card == null)
25. {
26. UnityEngine.Debug.LogError(string.Format("invalid card! {0}, {1}, {2}",
27. card\_number, pae\_type, position));
28. // error! Invalid slot index.
29. return PLAYER\_SELECT\_CARD\_RESULT.ERROR\_INVALID\_CARD;
30. }
32. this.card\_from\_player = card;
34. // 바닥 카드중에서 플레이어가 낸 카드와 같은 숫자의 카드를 구한다.
35. List<CCard> same\_cards = this.floor\_manager.get\_cards(card.number);
36. if (same\_cards != null)
37. {
38. this.same\_card\_count\_with\_player = (byte)same\_cards.Count;
39. }
40. else
41. {
42. this.same\_card\_count\_with\_player = 0;
43. }
45. //UnityEngine.Debug.Log("same card(player) " + this.same\_card\_count\_with\_player);
46. switch (this.same\_card\_count\_with\_player)
47. {
48. case 0:
49. {
50. if (is\_shaking == 1)
51. {
52. byte count\_from\_hand =
53. this.player\_agents[player\_index].get\_same\_card\_count\_from\_hand(this.card\_from\_player.number);
54. if (count\_from\_hand == 2)
55. {
56. this.card\_event\_type = CARD\_EVENT\_TYPE.SHAKING;
57. this.player\_agents[player\_index].plus\_shaking\_count();
59. // 플레이어에게 흔든 카드 정보를 보내줄 때 사용하기 위해서 리스트에 보관해 놓는다.
60. this.shaking\_cards =
61. this.player\_agents[player\_index].find\_same\_cards\_from\_hand(
62. this.card\_from\_player.number);
63. this.shaking\_cards.Add(this.card\_from\_player);
64. }
65. }
66. }
67. break;
69. case 1:
70. {
71. // 폭탄인 경우와 아닌 경우를 구분해서 처리 해 준다.
72. byte count\_from\_hand =
73. this.player\_agents[player\_index].get\_same\_card\_count\_from\_hand(this.card\_from\_player.number);
74. if (count\_from\_hand == 2)
75. {
76. this.card\_event\_type = CARD\_EVENT\_TYPE.BOMB;
78. get\_current\_player().plus\_shaking\_count();
80. // 플레이어가 선택한 카드와, 바닥 카드, 폭탄 카드를 모두 가져 간다.
81. this.cards\_to\_give\_player.Add(this.card\_from\_player);
82. this.cards\_to\_give\_player.Add(same\_cards[0]);
83. this.bomb\_cards\_from\_player.Add(this.card\_from\_player);
84. List<CCard> bomb\_cards =
85. this.player\_agents[player\_index].pop\_all\_cards\_from\_hand(this.card\_from\_player.number);
86. for (int i = 0; i < bomb\_cards.Count; ++i)
87. {
88. this.cards\_to\_give\_player.Add(bomb\_cards[i]);
89. this.bomb\_cards\_from\_player.Add(bomb\_cards[i]);
90. }
92. take\_cards\_from\_others(1);
93. this.player\_agents[player\_index].add\_bomb\_count(2);
94. }
95. else
96. {
97. // 뒤집이서 뻑이 나오면 못가져갈 수 있으므로 일단 임시변수에 넣어 놓는다.
98. this.cards\_to\_give\_player.Add(this.card\_from\_player);
99. this.cards\_to\_give\_player.Add(same\_cards[0]);
100. }
101. }
102. break;
104. case 2:
105. {
106. if (same\_cards[0].pae\_type != same\_cards[1].pae\_type)
107. {
108. // 카드 종류가 다르다면 플레이어가 한장을 선택할 수 있도록 해준다.
109. this.target\_cards\_to\_choice.Clear();
110. for (int i = 0; i < same\_cards.Count; ++i)
111. {
112. this.target\_cards\_to\_choice.Add(same\_cards[i]);
113. }
115. this.expected\_result\_type = PLAYER\_SELECT\_CARD\_RESULT.CHOICE\_ONE\_CARD\_FROM\_PLAYER;
116. return PLAYER\_SELECT\_CARD\_RESULT.CHOICE\_ONE\_CARD\_FROM\_PLAYER;
117. }
119. // 같은 종류의 카드라면 플레이어가 선택할 필요가 없으므로 첫번째 카드를 선택해 준다.
120. this.cards\_to\_give\_player.Add(this.card\_from\_player);
121. this.cards\_to\_give\_player.Add(same\_cards[0]);
122. }
123. break;
125. case 3:
126. {
127. //todo:자뻑인지 구분하여 처리하기.
128. this.card\_event\_type = CARD\_EVENT\_TYPE.EAT\_PPUK;
130. // 쌓여있는 카드를 모두 플레이어에게 준다.
131. this.cards\_to\_give\_player.Add(card);
132. for (int i = 0; i < same\_cards.Count; ++i)
133. {
134. this.cards\_to\_give\_player.Add(same\_cards[i]);
135. }
137. //todo:상대방 카드 한장 가져오기. 자뻑이었다면 두장 가져오기.
138. take\_cards\_from\_others(1);
139. }
140. break;
141. }
143. return PLAYER\_SELECT\_CARD\_RESULT.COMPLETED;
144. }

매소드의 길이가 꽤 길지만 하나 하나 살펴보면 별거 아닌 내용들입니다. 최대한 줄인다고 줄여봤는데도 이정도네요.

먼저 카드 검증 부분부터 시작 합니다.

선택한 카드가 정말 플레이어가 갖고 있는 카드인지 검사 합니다.

이런 부분은 온라인 게임에서 클라이언트의 입력을 검증할 때 들어가는 코드 형태입니다.

이 프로젝트는 클라이언트 내에서 모두 구현이 되고 있기 때문에 CGostopEngine클래스로 잘못된 카드를 전달하는 경우는 없을 것입니다.

하지만 이 클래스가 서버에서 돌아간다고 생각하고 작업한다면 위와 같은 검증 코드는

반드시 들어가야 하겠죠.

앞으로도 서버측에서 구현될 코드를 작성할 때에는 모두 저런 내용들을 구현할 것입니다.

조금 번거롭지만 사소한 코딩 실수로 로직이 꼬이는 경우도 미리 예방할 수 있다는 장점도 있습니다.

·미리보기 | 소스복사·

1. // 바닥 카드중에서 플레이어가 낸 카드와 같은 숫자의 카드를 구한다.
2. List<CCard> same\_cards = this.floor\_manager.get\_cards(card.number);
3. if (same\_cards != null)
4. {
5. this.same\_card\_count\_with\_player = (byte)same\_cards.Count;
6. }
7. else
8. {
9. this.same\_card\_count\_with\_player = 0;
10. }

다음으로 플레이어가 낸 카드와 바닥에 깔려 있는 카드를 비교하여 같은 숫자의 카드가 몇장 있는지 계산 합니다.

같은 숫자의 카드 장수에 따라서 고스톱 게임의 세부 규칙이 달라지기 때문에

간단한 코드지만 대단히 중요한 부분이라고 할 수 있습니다.

·미리보기 | 소스복사·

1. switch (this.same\_card\_count\_with\_player)
2. {
3. ...
4. }

이제 정말 본격적으로 고스톱 게임의 룰을 구현해 볼 시간입니다.

예전에 제가 신입 시절에 본 소스코드를 떠올려 보면 이 switch문의 코드가 천줄이 넘었던 것으로 기억합니다.

정말 그 당시에는 리팩토링 할 엄두도 못내었었죠.

다행이도 여기서는 그렇게 길진 않으니 너무 겁먹진 마시기 바랍니다.

먼저 같은 카드가 한 장도 없을 경우의 코드입니다. case 0 에 해당 합니다.

·미리보기 | 소스복사·

1. if (is\_shaking == 1)
2. {
3. byte count\_from\_hand =
4. this.player\_agents[player\_index].get\_same\_card\_count\_from\_hand(this.card\_from\_player.number);
5. if (count\_from\_hand == 2)
6. {
7. this.card\_event\_type = CARD\_EVENT\_TYPE.SHAKING;
8. this.player\_agents[player\_index].plus\_shaking\_count();
10. // 플레이어에게 흔든 카드 정보를 보내줄 때 사용하기 위해서 리스트에 보관해 놓는다.
11. this.shaking\_cards =
12. this.player\_agents[player\_index].find\_same\_cards\_from\_hand(
13. this.card\_from\_player.number);
14. this.shaking\_cards.Add(this.card\_from\_player);
15. }
16. }

플레이어가 낸 카드와 일치하는 바닥 카드가 한 장도 없는 경우라면 그냥 바닥에 내려 놓아야 합니다.

단, 플레이어가 흔들기를 선언 할 경우 선택한 카드를 상대방에게 보여줘야 하는 규칙이 있습니다.

같은 숫자의 카드 세장을 들고 있으면서 바닥에 같은 숫자의 카드가 한 장도 없을 때

흔들기를 선언할 수 있는데 이 때 들고 있는 세장의 카드를 상대방에게 보여줘야 합니다.

대신 점수 계산시 두배의 효과를 적용할 수 있습니다.

흔들기를 하지 않으면 두배의 효과도 없으며 상대방에게 카드를 보여줄 필요도 없습니다.

if (count\_from\_hand == 2)

여기서 3이 아니라 2와 비교한 이유는 앞선 코드에서

pop\_card\_from\_hand매소드를 통해 플레이어의 카드 한 장을 이미 빼내왔기 때문입니다.

다음은 같은 카드가 한 장 있을 경우의 코드입니다.

·미리보기 | 소스복사·

1. // 폭탄인 경우와 아닌 경우를 구분해서 처리 해 준다.
3. byte count\_from\_hand =
5. this.player\_agents[player\_index].get\_same\_card\_count\_from\_hand(this.card\_from\_player.number);
7. if (count\_from\_hand == 2)
9. {
11. this.card\_event\_type = CARD\_EVENT\_TYPE.BOMB;


15. get\_current\_player().plus\_shaking\_count();


19. // 플레이어가 선택한 카드와, 바닥 카드, 폭탄 카드를 모두 가져 간다.
21. this.cards\_to\_give\_player.Add(this.card\_from\_player);
23. this.cards\_to\_give\_player.Add(same\_cards[0]);
25. this.bomb\_cards\_from\_player.Add(this.card\_from\_player);
27. List<CCard> bomb\_cards =
29. this.player\_agents[player\_index].pop\_all\_cards\_from\_hand(this.card\_from\_player.number);
31. for (int i = 0; i < bomb\_cards.Count; ++i)
33. {
35. this.cards\_to\_give\_player.Add(bomb\_cards[i]);
37. this.bomb\_cards\_from\_player.Add(bomb\_cards[i]);
39. }


43. take\_cards\_from\_others(1);
45. this.player\_agents[player\_index].add\_bomb\_count(2);
47. }
49. else
51. {
53. // 뒤집이서 뻑이 나오면 못가져갈 수 있으므로 일단 임시변수에 넣어 놓는다.
55. this.cards\_to\_give\_player.Add(this.card\_from\_player);
57. this.cards\_to\_give\_player.Add(same\_cards[0]);

60. }

바닥에 같은 숫자의 카드가 한 장 깔려 있을 때는 두가지 경우가 있습니다.

첫 번째는 이전에 처리 했던 흔들기와 비슷한 경우로 폭탄이라는 규칙입니다.

같은 숫자의 카드를 세장 들고 있고 바닥에 같은 숫자의 카드가 한 장 깔려 있다면

해당 카드를 낼 때 폭탄처리가 됩니다.

폭탄은 세장의 카드를 한꺼번에 낼 수 있습니다.

그리고 상대방으로부터 피 한 장을 뺏어올 수 있습니다.

take\_cards\_from\_others매소드가 이 기능을 담당합니다.

폭탄으로 낸 카드와 바닥에 깔려 있는 카드는 모두 플레이어가 가져갈 수 있으므로

cards\_to\_give\_players리스트에 카드들을 모두 담아 놓고

나중에 턴 종료를 위해 카드를 정리할 때 한꺼번에 가져가도록 처리 합니다.

두 번째는 그냥 일반적인 경우입니다.

이 경우에는 낸 카드와 바닥에 깔린 카드를 플레이어가 가져갈 수 있습니다.

단, 가운데 카드를 뒤집은 후의 결과에 따라 뻑이 나올 수 도 있기 때문에

일단은 임시 변수에 넣어놓도록 구현합니다.

다음은 같은 숫자의 카드가 두장 있을 경우입니다.

·미리보기 | 소스복사·

1. if (same\_cards[0].pae\_type != same\_cards[1].pae\_type)
3. {
5. // 카드 종류가 다르다면 플레이어가 한장을 선택할 수 있도록 해준다.
7. this.target\_cards\_to\_choice.Clear();
9. for (int i = 0; i < same\_cards.Count; ++i)
11. {
13. this.target\_cards\_to\_choice.Add(same\_cards[i]);
15. }


19. this.expected\_result\_type = PLAYER\_SELECT\_CARD\_RESULT.CHOICE\_ONE\_CARD\_FROM\_PLAYER;
21. return PLAYER\_SELECT\_CARD\_RESULT.CHOICE\_ONE\_CARD\_FROM\_PLAYER;
23. }


27. // 같은 종류의 카드라면 플레이어가 선택할 필요가 없으므로 첫번째 카드를 선택해 준다.
29. this.cards\_to\_give\_player.Add(this.card\_from\_player);
31. this.cards\_to\_give\_player.Add(same\_cards[0]);

이 경우에는 카드의 타입을 비교해서 서로 다른 타입이라면 플레이어에게 선택권을 줍니다.

두장의 카드가 서로 같은 타입이라면 어떤 카드를 가져오던

상관 없으므로 첫 번째 카드를 자동으로 선택해 줍니다.

마지막으로 같은 카드가 세장 있을 경우입니다.

·미리보기 | 소스복사·

1. //todo:자뻑인지 구분하여 처리하기.
3. this.card\_event\_type = CARD\_EVENT\_TYPE.EAT\_PPUK;


7. // 쌓여있는 카드를 모두 플레이어에게 준다.
9. this.cards\_to\_give\_player.Add(card);
11. for (int i = 0; i < same\_cards.Count; ++i)
13. {
15. this.cards\_to\_give\_player.Add(same\_cards[i]);
17. }


21. //todo:상대방 카드 한장 가져오기. 자뻑이었다면 두장 가져오기.
23. take\_cards\_from\_others(1);

이 경우에는 바닥에 깔려 있는 카드와 자신이 낸 카드를 모두 가져올 수 있습니다.  
단, 이전에 본인이 뻑을 해서 쌓여있는 카드일 경우에는 자뻑이라고 해서  
상대방에게 피 두장을 뺏어올 수 있습니다.  
이 부분은 아직 구현이 안된 부분이라 주석으로 표시만 해놓고 일단 한 장만 뺏어오도록 코딩해놨습니다.

이번 강좌는 여기 까지 입니다.

다음 강좌에서는 카드를 뒤집은 후의 구현부분에 대해서 다루겠습니다.

감사합니다.