**[Unity3D] 고스톱 게임 만들기 - 7 (카드 움직임)**

**소스코드 링크 :**[**https://github.com/sunduk/freegostop**](https://github.com/sunduk/freegostop)

■ 카드 분배 장면 구현

카드 움직임 구현은 몇단계로 나눠서 살펴볼것입니다.

첫 번째로 알아볼 것은 선이 정해진 이후 플레이어와 바닥에 카드를 분배하는 장면입니다.

CPlayRoomUI클래스의 distribute\_cards매소드에서 이 부분을 구현합니다.

그럼 distribute\_cards매소드의 코드를 보겠습니다.

·미리보기 | 소스복사·

1. IEnumerator distribute\_cards(Queue<CCard> floor\_cards, Dictionary<byte, Queue<CCard>> player\_cards)
2. {
3. yield return new WaitForSeconds(1.0f);
5. List<CCardPicture> begin\_cards\_picture = new List<CCardPicture>();
7. // [바닥 -> 1P -> 2P 나눠주기] 를 두번 반복한다.
8. for (int looping = 0; looping < 2; ++looping)
9. {
10. // 바닥에는 4장씩 분배한다.
11. for (int i = 0; i < 4; ++i)
12. {
13. CCard card = floor\_cards.Dequeue();
14. CCardPicture card\_picture = this.deck\_cards.Pop();
15. card\_picture.update\_card(card, get\_hwatoo\_sprite(card));
16. begin\_cards\_picture.Add(card\_picture);
18. card\_picture.transform.localScale = SCALE\_TO\_FLOOR;
19. move\_card(card\_picture, card\_picture.transform.position, this.floor\_slot\_position[i + looping \* 4]);
21. yield return new WaitForSeconds(0.02f);
22. }
24. yield return new WaitForSeconds(0.1f);
26. // 플레어이의 카드를 분배한다.
27. foreach(KeyValuePair<byte, Queue<CCard>> kvp in player\_cards)
28. {
29. byte player\_index = kvp.Key;
30. Queue<CCard> cards = kvp.Value;
32. byte ui\_slot\_index = (byte)(looping \* 5);
33. // 플레이어에게는 한번에 5장씩 분배한다.
34. for (int card\_index = 0; card\_index < 5; ++card\_index)
35. {
36. CCardPicture card\_picture = this.deck\_cards.Pop();
37. card\_picture.set\_slot\_index(ui\_slot\_index);
38. this.player\_hand\_card\_manager[player\_index].add(card\_picture);
40. // 본인 카드는 해당 이미지를 보여주고,
41. // 상대방 카드(is\_nullcard)는 back\_image로 처리한다.
42. if (player\_index == this.player\_me\_index)
43. {
44. CCard card = cards.Dequeue();
45. card\_picture.update\_card(card, get\_hwatoo\_sprite(card));
46. card\_picture.transform.localScale = SCALE\_TO\_MY\_HAND;
47. move\_card(card\_picture, card\_picture.transform.position,
48. this.player\_card\_positions[player\_index].get\_hand\_position(ui\_slot\_index));
49. }
50. else
51. {
52. card\_picture.update\_backcard(this.back\_image);
53. card\_picture.transform.localScale = SCALE\_TO\_OTHER\_HAND;
54. move\_card(card\_picture, card\_picture.transform.position,
55. this.player\_card\_positions[player\_index].get\_hand\_position(ui\_slot\_index));
56. }
58. ++ui\_slot\_index;
60. yield return new WaitForSeconds(0.02f);
61. }
62. }
63. }
65. sort\_floor\_cards\_after\_distributed(begin\_cards\_picture);
66. sort\_player\_hand\_slots(this.player\_me\_index);
68. CPacket msg = CPacket.create((short)PROTOCOL.DISTRIBUTED\_ALL\_CARDS);
69. CNetworkManager.Instance.send(msg);
70. }

분배 순서는 다음과 같습니다.

바닥 4장 -> 1P 5장 -> 2P 5장 -> 바닥 4장 -> 1P 5장 -> 2P 5장 -> 끝

distribute\_cards매소드의 첫 부분에 바닥에 4장을 분배하는 코드가 구현되어 있습니다.

·미리보기 | 소스복사·

1. // [바닥 -> 1P -> 2P 나눠주기] 를 두번 반복한다.
2. for (int looping = 0; looping < 2; ++looping)
3. {
4. // 바닥에는 4장씩 분배한다.
5. for (int i = 0; i < 4; ++i)
6. {
7. CCard card = floor\_cards.Dequeue();
8. CCardPicture card\_picture = this.deck\_cards.Pop();
9. card\_picture.update\_card(card, get\_hwatoo\_sprite(card));
10. begin\_cards\_picture.Add(card\_picture);
12. card\_picture.transform.localScale = SCALE\_TO\_FLOOR;
13. move\_card(card\_picture, card\_picture.transform.position, this.floor\_slot\_position[i + looping \* 4]);
15. yield return new WaitForSeconds(0.02f);
16. }
18. yield return new WaitForSeconds(0.1f);
20. ...
21. }

[바닥 -> 1P -> 2P] 나눠주기를 두 번 반복해야 하기 때문에 먼저 for루프로 들어갑니다.

그리고 카드를 한 장 꺼내옵니다. 눈에 보이는 카드는 하나의 객체이지만 코드에서는 두가지 부분으로 나눠서 처리 합니다.

카드 자체의 정보만을 갖고 있는 CCard클래스와

이미지를 표현하는데 필요한 CCardPicture클래스로 나눠져 구현되어 있습니다.

CCard클래스는 카드 정보만 갖고 있는 클래스입니다. 카드 번호, 광, 피, 열끗, 띠등의 타입과 특수 속성만을 갖고 있습니다.

여기에는 이미지 정보는 없는 순수한 카드 자체의 정보만을 보관하고 있습니다.

따라서 화면에 출력해주기 위해서는 카드 이미지를 갖고 있는 객체도 필요한데

그것이 바로 다음줄에 있는 CCardPicture클래스로 표현 됩니다.

이렇게 두 개로 구분한 것은 이미지 정보가 필요 없는 서버측 코딩에도 카드 클래스를 같이 사용하기 위해서입니다.

CCard card = floor\_cards.Dequeue();

카드를 한 장 꺼내오는 부분입니다. 여기서는 바닥 카드를 꺼내와야 하니 바닥카드의 정보를 담고 있는 컨테이너인 floor\_cards라는 큐에서 한 장 뽑아오겠습니다.

앞서 말씀드렸듯이 CCard클래스에는 카드 자체의 정보값들로만 구성되어 있다고 했습니다.

클라이언트에 표시하려면 카드 정보를 토대로 실제 카드 이미지를 표현할 수 있는 객체도 필요합니다.

그부분이 바로 다음줄에 나오는 CCardPicture클래스입니다.

CCardPicture card\_picture = this.deck\_cards.Pop();

모든 카드는 deck\_cards라는 곳에서 꺼내오게 되는데 이 deck\_cards는 게임에서 사용되는 48장의 카드 객체를 갖고 있는 스택입니다.

고스톱에서 카드를 꺼낼때는 무조건 제일 윗장부터 꺼낼 수 있으므로 스택 자료구조를 사용하여 표현한 것입니다.

만약 중간에 꺼낼 수 있는 룰을 적용시킨다면 스택대신 리스트를 사용해야 겠죠.

여기서 꺼내온 CCardPicture객체에 카드 이미지를 설정해야 해당 카드에 맞는 이미지가 화면에 출력됩니다.

카드 자체의 정보는 먼저 꺼내온 CCard객체에 담겨져 있으니 그 객체로부터 이미지를 설정해 보겠습니다.

card\_picture.update\_card(card, get\_hwatoo\_sprite(card));

update\_card매소드를 통해서 카드 자체의 정보와 이미지를 설정하는 부분입니다.

CCardPicture클래스는 카드 이미지를 표현하는 클래스이지만 카드 자체의 정보인 CCard클래스를 멤버로 담고 있는다면

코딩할 때 한결 편해지기 때문에 첫 번째 파라미터로 CCard클래스도 같이 넣어준 것입니다.

두 번째 파라미터로 get\_hwatoo\_sprite(card)를 통해서 이미지를 넘겨줍니다.

get\_hwatoo\_sprite매소드는 카드 정보에 맞는 이미지를 찾아주는 일을 담당 합니다.

·미리보기 | 소스복사·

1. Sprite get\_hwatoo\_sprite(CCard card)
2. {
3. int sprite\_index = card.number \* 4 + card.position;
4. return CSpriteManager.Instance.get\_card\_sprite(sprite\_index);
5. }

매소드의 내용은 이렇게 생겼는데 카드 번호와 이미지상의 위치에 따라서 스프라이트 인덱스를 구하고 해당 스프라이트를 리턴해주는 형식으로 구현되어 있습니다.

지금은 카드 분배로직을 살펴보는 시간이므로 여기서 사용된 CSpriteManager클래스는 다음시간에 자세히 알아보도록 하겠습니다.

클라이언트에서 카드 객체를 표현하는데 필요한 정보들을 다시 정리해보겠습니다.

카드 자체의 정보를 갖고 있는 CCard 클래스.

카드 이미지를 표현할 수 있는 CCardPicture 클래스.

위 두 개의 클래스들은 클라이언트내에서 자주 사용되므로 그 의미를 확실히 파악하시는게 중요합니다.

카드 한 장의 이미지 설정이 완료된 후에는 화면상에서 카드를 움직여야 합니다.

가운데 덱의 위치부터 바닥 위치까지 이동시키는 부분을 알아보겠습니다.

·미리보기 | 소스복사·

1. begin\_cards\_picture.Add(card\_picture);
3. card\_picture.transform.localScale = SCALE\_TO\_FLOOR;
4. move\_card(card\_picture, card\_picture.transform.position, this.floor\_slot\_position[i + looping \* 4]);

첫줄에서 begin\_cards\_picture리스트에 card\_picture객체를 담고 있는데 이부분은 일단 건너뛰도록 하겠습니다.

간단히 설명드리자면 카드를 다 분배한 후 같은 번호의 카드끼리 정렬하는데 사용하기 위한 부분입니다.

카드 이동의 첫 번째 단계는 SCALE\_TO\_FLOOR값으로 카드의 스케일을 조절합니다.

바닥 카드는 약간 작게 표현되도록 만들었기 때문에 바닥카드에 맞는 스케일값으로 설정하는 부분입니다. 원하신다면 제거해도 상관없는 코드입니다.

그 다음에 move\_card매소드를 통해서 카드의 움직임을 시작합니다.

넘겨주는 파라미터로는 카드 이미지를 갖고 있는 card\_picture와

움직임을 시작할 위치, 도착할 위치값이 있습니다.

move\_card매소드가 어떻게 구현되어 있는지 보겠습니다.

·미리보기 | 소스복사·

1. void move\_card(CCardPicture card\_picture, Vector3 begin, Vector3 to, float duration = 0.1f)
2. {
3. if (card\_picture.card != null)
4. {
5. card\_picture.update\_image(get\_hwatoo\_sprite(card\_picture.card));
6. }
7. else
8. {
9. card\_picture.update\_image(this.back\_image);
10. }
12. CMovingObject mover = card\_picture.GetComponent<CMovingObject>();
13. mover.begin = begin;
14. mover.to = to;
15. mover.duration = duration;
16. mover.run();
17. }

제일 먼저 카드이미지를 설정 합니다. 게임을 돌려보면 빨라서 안보일 수도 있지만

자세히 살펴보면 현재 위치에서 카드 이미지를 먼저 교체한 후 도착위치로 이동시키도록 구현되어 있습니다.

이 때 카드 정보가 없다면(null이라면) 뒷면으로 가정하고 back\_image를 설정해 줍니다.

여기서는 바닥 카드를 이미 설정해놓았으니 해당 카드에 맞는 이미지가 들어가겠죠.

(사실 null로 카드 존재 유무를 체크하는 방법은 그리 좋은 설계는 아니지만 일단 이렇게 구현하도록 하겠습니다. 추후에 여건이 되면 리팩토링 하면 되니까요)

그 다음줄을 보시면 CMovingObject라는 클래스가 나타납니다.

이 클래스는 CCardPicture의 게임 오브젝트에 붙어있는 클래스입니다.

시작과 끝 위치를 정해주면 실제로 화면상에서 객체를 움직이는 일을 수행합니다.

mover.run()매소드가 어떻게 구현되어 있는지 보겠습니다.

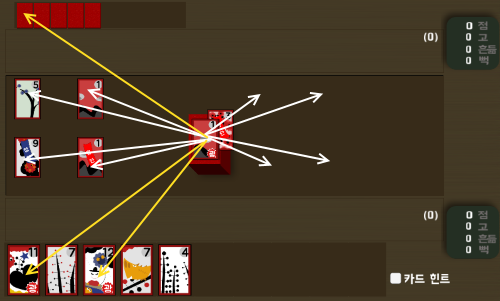
·미리보기 | 소스복사·

1. public void run()
2. {
3. StopAllCoroutines();
4. StartCoroutine(run\_moving());
5. }

이전에 혹시라도 코루틴이 진행되고 있었다면 모두 중지한 후 다시 시작하도록 되어 있습니다.

·미리보기 | 소스복사·

1. IEnumerator run\_moving()
2. {
3. this.sprite\_renderer.sortingOrder = CSpriteLayerOrderManager.Instance.Order;
5. float begin\_time = Time.time;
6. while (Time.time - begin\_time <= duration)
7. {
8. float t = (Time.time - begin\_time) / duration;
10. float x = EasingUtil.easeInExpo(begin.x, to.x, t);
11. float y = EasingUtil.easeInExpo(begin.y, to.y, t);
12. transform.position = new Vector3(x, y, begin.z);
14. yield return 0;
15. }
17. transform.position = to;
18. }



실제로 움직이는 부분인데요, 제일 첫줄에 sortingOrder를 설정하는 코드가 있습니다.

움직이는 카드는 화면에서 제일 위로 출력되게 하기 위한 부분입니다.

추후에 SCSpriteLayerOrderManager클래스에 대해서도 설명드리도록 하겠습니다.

다음줄을 보시면 while루프 안에서 transform.position을 갱신해주며 이동을 구현하는 코드가 보입니다.

객체를 지정된 시간(duration)동안 시작(begin)위치에서 끝(to)위치 까지 보간시켜 주는 코드입니다.

2D게임이기 때문에 z값을 건드리지 않고 x,y좌표만 변화시킵니다.

이 때 사용된 보간 방법은 easeInExpo라는 매소드를 사용합니다.

그냥 일정하게 linear로 해도 되지만 조금 더 움직이는 맛을 내기 위해서 가속도가 들어간 보간 방법을 사용한 것입니다.

EasingUtil클래스를 보면 linear부터 여러 가지 함수들이 있는데 바꿔가면서 적용시켜보면 카드의 움직임이 약간씩 다른 느낌을 내게 될겁니다.

이 클래스에 대해서는 iTween을 사용해 보셨다면 금방 이해할 수 있는 부분입니다.

사용법은 유니티 엔진에 포함되어 있는 Mathf.Lerp매소드와 같이 (시작, 끝, 시간) 값으로 표현됩니다.

이제 지금까지의 작업을 4번 반복하면 바닥 카드 4장이 분배됩니다.

그 다음에는 플레이어에게 카드를 나눠주는 장면을 살펴보겠습니다.

카드 이미지를 설정하고 움직이는 부분은 바닥 카드와 동일한 로직으로 처리됩니다.

한가지 다른 것이 있다면 상대방 카드와 본인 카드를 구분하여 출력해준다는 점입니다.

상대방 카드는 볼 수 없으므로 뒷면으로 표시해 주고

본인의 카드는 실제 카드 이미지를 표시해주도록 구현해야 합니다.

그렇지 않으면 상대방이 어떤 카드를 갖고 있는지 다 알게 되기 때문이죠.

·미리보기 | 소스복사·

1. // 플레어이의 카드를 분배한다.
2. foreach(KeyValuePair<byte, Queue<CCard>> kvp in player\_cards)
3. {
4. byte player\_index = kvp.Key;
5. Queue<CCard> cards = kvp.Value;
7. byte ui\_slot\_index = (byte)(looping \* 5);
8. // 플레이어에게는 한번에 5장씩 분배한다.
9. for (int card\_index = 0; card\_index < 5; ++card\_index)
10. {
11. CCardPicture card\_picture = this.deck\_cards.Pop();
12. card\_picture.set\_slot\_index(ui\_slot\_index);
13. this.player\_hand\_card\_manager[player\_index].add(card\_picture);
15. // 본인 카드는 해당 이미지를 보여주고,
16. // 상대방 카드(is\_nullcard)는 back\_image로 처리한다.
17. if (player\_index == this.player\_me\_index)
18. {
19. CCard card = cards.Dequeue();
20. card\_picture.update\_card(card, get\_hwatoo\_sprite(card));
21. card\_picture.transform.localScale = SCALE\_TO\_MY\_HAND;
22. move\_card(card\_picture, card\_picture.transform.position,
23. this.player\_card\_positions[player\_index].get\_hand\_position(ui\_slot\_index));
24. }
25. else
26. {
27. card\_picture.update\_backcard(this.back\_image);
28. card\_picture.transform.localScale = SCALE\_TO\_OTHER\_HAND;
29. move\_card(card\_picture, card\_picture.transform.position,
30. this.player\_card\_positions[player\_index].get\_hand\_position(ui\_slot\_index));
31. }
33. ++ui\_slot\_index;
35. yield return new WaitForSeconds(0.02f);
36. }
37. }

플레이어들의 카드를 분배하는 코드입니다.

상대방의 카드일 경우에는 update\_backcard매소드로 카드 뒷면을 나타내는 back\_image를 넣어주는 부분이 보입니다.

카드 움직임은 바닥카드에서 사용한 move\_card매소드를 그대로 사용합니다.

나눠준 카드들은 게임 진행시 계속 사용할 것이기 때문에 player\_hand\_card\_manager라는 객체에 설정해줘야 합니다.

여기서는 카드의 움직임 부분만을 살펴보는 것이므로 저 객체가 어떻게 사용되는지는 추후에 설명드리도록 하겠습니다.

·미리보기 | 소스복사·

1. sort\_floor\_cards\_after\_distributed(begin\_cards\_picture);
2. sort\_player\_hand\_slots(this.player\_me\_index);
4. CPacket msg = CPacket.create((short)PROTOCOL.DISTRIBUTED\_ALL\_CARDS);
5. CNetworkManager.Instance.send(msg);

카드를 다 나눠준 후에는 바닥 카드 정렬과 플레이어 카드 정렬을 수행 합니다.

바닥에 같은 번호의 카드가 두장 이상 있을 경우 서로 겹쳐서 놓도록 좌표 이동을 진행하며,

플레이어의 카드는 번호 순서대로 오도록 정렬 해줍니다.

정렬 작업이 모두 끝나면 완료했다는 메시지를 서버에 전송하여 카드 분배를 끝냅니다.

다음 강좌에서는 카드 선택과 뒤집기 부분의 구현에 대해서 알아보도록 하겠습니다.

감사합니다.