[Unity3D] 고스톱 게임 만들기 - 2 (로직 구현)

이 강좌는 로직 구현 -> 화면 처리의 순서로 진행 됩니다.

초반부에는 유니티 엔진에 종속적이지 않은 클래스들의 구현이 나옵니다.

이 클래스들로 게임의 로직을 구현한 뒤 화면 처리 부분을 만들어 볼 것입니다.

먼저 전체 소스코드를 받아서 돌려본 뒤 강좌를 한번 훑어보시면 이해하기 쉬울것이라 생각됩니다.

**소스코드는 첫번째 강좌 게시물에서 압축파일 또는 svn을 통해 받으실 수 있습니다.**  
첫번째 강좌 링크 : [http://lab.gamecodi.com/board/zboard.php?id=GAMECODILAB\_Lecture\_series&amp;no=91&amp;z=](http://lab.gamecodi.com/board/zboard.php?id=GAMECODILAB_Lecture_series&no=91&z=)

\* 1:1고스톱은 따로 부르는 명칭(MAT-GO)이 있지만 이 게시판에 그 글자가 등록되지 않아 고스톱이라고 수정했습니다.

chapter 2 고스톱 로직 구현

고스톱 게임의 개발을 위해서는 카드를 정의할 수 있는 클래스들과 그 카드들을 이용하여 게임의 룰을 구현하는 클래스들이 필요 합니다.

카드와 룰을 구현하는 클래스는 서버측(여기서는 실제 서버가 아니라 클라이언트 프로젝트 내에 존재하면서

서버 역할을 담당하는 클래스들을 의미한다) 구현에 해당 합니다.

클라이언트에도 게임 로직이 들어갈 수 있지만 서버측에 구현하게 되면

코드에서 로직과 UI처리를 구분할 수 있어 이해하기가 쉽고 작업 목표가 명확히 세워지는 장점이 있습니다.

따라서 게임의 룰은 클라이언트와 관계 없는 별도의 클래스에서 구현하게 될 것입니다.

그 클래스의 이름이 바로 CGostopEngine이라는 클래스이며 이 클래스는 유니티의 MonoBehaviour를 상속받지 않는

일반적인 클래스로 되어 있습니다.

화면에 보여지는 부분은 클라이언트에서 사용하는 엔진에 따라 달라질 수 있는 부분입니다.

하지만 게임의 룰은 어떤 클라이언트 엔진을 사용하던 변하지 않는 부분입니다.

여기서는 변할 수 있는 부분과 변하지 않는 부분을 구분하여 작업할 것인데

이렇게 설계해 놓으면 추후 다른 엔진에서도 보다 쉽게 재사용 할 수 있는 코드가 됩니다.

**2.1 CGostopEngine 클래스**

CGostopEngine클래스는 고스톱 게임의 규칙을 구현한 클래스입니다.

카드 데이터를 보관하고 있으며 플레이어들에게 카드를 분배하고 선택한 카드에 대해 알맞은 로직을 처리하는 부분이 구현되어 있습니다.

멤버 변수들을 살펴보겠습니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 고스톱의 룰을 구현한 클래스.
3. /// </summary>
4. public class CGostopEngine
5. {
6. // 전체 카드를 보관할 컨테이너.
7. CCardManager card\_manager;
8. Queue<CCard> card\_queue;
10. // 플레이어들.
11. byte first\_player\_index;
12. List<CPlayerAgent> player\_agents;
14. CFloorCardManager floor\_manager;
16. // 게임 진행시 카드 정보들을 저장해놓을 임시 변수들.
17. // 한턴이 끝나면 모두 초기화 시켜줘야 한다.
18. public TURN\_RESULT\_TYPE turn\_result\_type { get; private set; }
19. public CCard card\_from\_player { get; private set; }
20. public byte selected\_slot\_index { get; private set; }
21. public CCard card\_from\_deck { get; private set; }
22. public List<CCard> bomb\_cards\_from\_player { get; private set; }
23. public List<CCard> target\_cards\_to\_choice { get; private set; }
24. public byte same\_card\_count\_with\_player { get; private set; }
25. public byte same\_card\_count\_with\_deck { get; private set; }
26. public CARD\_EVENT\_TYPE card\_event\_type { get; private set; }
27. public List<CARD\_EVENT\_TYPE> flipped\_card\_event\_type { get; private set; }
28. public List<CCard> cards\_to\_give\_player { get; private set; }
29. public List<CCard> cards\_to\_floor { get; private set; }
30. public Dictionary<byte, List<CCard>> other\_cards\_to\_player { get; private set; }
31. public List<CCard> shaking\_cards { get; private set; }
32. // 두개의 카드중 하나를 선택하는 경우는 두가지가 있는데(플레이어가 낸 경우, 덱에서 뒤집은 경우),
33. // 서버에서 해당 상황에 맞는 타입을 들고 있다가
34. // 클라이언트로부터 온 타입과 맞는지 비교하는데 사용한다.
35. // 틀리다면 오류 또는 어뷰징이다.
36. PLAYER\_SELECT\_CARD\_RESULT expected\_result\_type;
38. public byte current\_player\_index { get; private set; }
40. // history.
41. public List<CCard> distributed\_floor\_cards { get; private set; }
42. public List<List<CCard>> distributed\_players\_cards { get; private set; }
43. ...
44. }

멤버 변수들이 꽤 많습니다만 하나하나 주석을 참고하시면 존재의 이유를 파악하실 수 있을 겁니다.

이보다 더 효율적인 방법도 물론 있습니다. 일단은 이렇게 코딩해 봤습니다.

다음은 초기화 부분입니다.

·미리보기 | 소스복사·

1. /// <summary>
2. /// 게임 한판 시작 전에 초기화 해야할 데이터.
3. /// </summary>
4. public void reset()
5. {
6. this.player\_agents.ForEach(obj => obj.reset());
7. this.first\_player\_index = 0;
8. this.current\_player\_index = this.first\_player\_index;
9. this.card\_manager.make\_all\_cards();
10. this.distributed\_floor\_cards.Clear();
11. this.distributed\_players\_cards.Clear();
12. this.floor\_manager.reset();
14. clear\_turn\_data();
15. }

18. /// <summary>
19. /// 매 턴 진행하기 전에 초기화 해야할 데이터.
20. /// </summary>
21. public void clear\_turn\_data()
22. {
23. this.turn\_result\_type = TURN\_RESULT\_TYPE.RESULT\_OF\_NORMAL\_CARD;
24. this.card\_from\_player = null;
25. this.selected\_slot\_index = byte.MaxValue;
26. this.card\_from\_deck = null;
27. this.target\_cards\_to\_choice.Clear();
28. this.same\_card\_count\_with\_player = 0;
29. this.same\_card\_count\_with\_deck = 0;
30. this.card\_event\_type = CARD\_EVENT\_TYPE.NONE;
31. this.flipped\_card\_event\_type.Clear();
32. this.cards\_to\_give\_player.Clear();
33. this.cards\_to\_floor.Clear();
34. this.other\_cards\_to\_player.Clear();
35. this.bomb\_cards\_from\_player.Clear();
36. this.expected\_result\_type = PLAYER\_SELECT\_CARD\_RESULT.ERROR\_INVALID\_CARD;
37. this.shaking\_cards.Clear();
38. }

초기화는 두 부분으로 나누어 구현하였습니다.

먼저 게임 한판이 시작되기 전에 초기화 해야  할 부분이 있습니다.

이 때 는 플레이어들을 초기화 해 주고 게임에서 사용하는 카드들을 생성하는 작업을 수행 합니다.

·미리보기 | 소스복사·

1. this.card\_manager.make\_all\_cards();

이 부분이 전체 카드를 생성 하는 코드입니다. 여기서는 보너스 카드는 구현하지 않으므로 총 48장에 해당하는 카드 객체를 생성해 놓습니다.

그 다음으로 매 턴 마다 초기화 해야 할 부분이 있습니다.

이 때는 지난 턴에서 사용했던 데이터들을 모두 초기화 해 줍니다.

플레이어가 낸 카드, 뒤집은 카드, 가져가야 할 카드등에 대한 변수들을 모두 초기화 한 뒤 다음 플레이어의 게임 진행을 준비 합니다.

**2.2 카드 분배**

게임 진행을 위한 초기화 과정이 완료되면 카드를 분배하는 작업을 진행 합니다.

·미리보기 | 소스복사·

1. // 게임 시작.
2. public void start(List<CPlayer> players)
3. {
4. this.player\_agents.Clear();
5. for (int i = 0; i < players.Count; ++i)
6. {
7. this.player\_agents.Add(players[i].agent);
8. this.player\_agents[i].reset();
9. this.distributed\_players\_cards.Add(new List<CCard>());
10. }
12. shuffle();
13. distribute\_cards();
15. for (int i = 0; i < this.player\_agents.Count; ++i)
16. {
17. this.player\_agents[i].sort\_player\_hand\_slots();
18. }
19. }

start매소드에서 게임이 시작 됩니다. 플레이어를 세팅한 뒤 shuffle매소드를 통해서 카드를 무작위로 섞어 줍니다.

·미리보기 | 소스복사·

1. // 카드 섞기.
2. void shuffle()
3. {
4. this.card\_manager.shuffle();
6. this.card\_queue.Clear();
7. this.card\_manager.fill\_to(this.card\_queue);
8. }

카드를 무작위로 섞은 뒤 fill\_to매소드를 통해서 card\_queue라는 큐에 집어 넣습니다. 이 card\_queue는 고스톱 게임에서 중앙에 쌓여있는 카드 더미를 뜻합니다. 어떤 카드던지 제일 위에서부터 한 장씩 꺼내게 되므로 큐 자료구조를 이용하여 구현하였습니다. 스택으로 구현해도 비슷하겠죠. 단, 중간에 있는 카드를 꺼낼 수는 없으니 리스트와 같은 자료구조보다는 큐나 스택을 사용하는 것이 좋을 것입니다.

·미리보기 | 소스복사·

1. using System;
2. using System.Collections;
3. using System.Collections.Generic;
5. public static class CHelper
6. {
7. public static void Shuffle<T>(List<T> list)
8. {
9. Random rng = new Random();
10. int n = list.Count;
11. while (n > 1)
12. {
13. n--;
14. int k = rng.Next(n + 1);
15. T value = list[k];
16. list[k] = list[n];
17. list[n] = value;
18. }
19. }
20. }

CGostopEngine클래스의 멤버는 아니지만 카드를 섞는 로직을 보여드리기 위해 Shuffle매소드를 잠시 소개합니다.

List의 내용을 랜덤하게 섞는 역할을 하는 매소드입니다. 구글에 검색하면 바로 나오는 코드입니다.



카드가 분배된 모습. 같은 숫자의 카드는 겹쳐 놓는다.

카드를 섞은 뒤에는 바닥과 플레이어들에게 카드를 분배 합니다.

고스톱 게임에서 카드 분배는 바닥에 8장, 플레이어들에게 각각 10장씩 이루어집니다.

바닥 -> 1P -> 2P -> 바닥 -> 1P -> 2P 이런 순서로 카드를 분배 합니다.

·미리보기 | 소스복사·

1. // 카드 분배.
2. void distribute\_cards()
3. {
4. byte player\_index = this.first\_player\_index;
5. byte floor\_index = 0;
6. // 2번 반복하여 바닥에는 8장, 플레이어에게는 10장씩 돌아가도록 한다.
7. for (int count = 0; count < 2; ++count)
8. {
9. // 바닥에 4장.
10. for (byte i = 0; i < 4; ++i)
11. {
12. CCard card = pop\_front\_card();
13. this.distributed\_floor\_cards.Add(card);
15. this.floor\_manager.put\_to\_begin\_card(card);
16. ++floor\_index;
17. }
19. // 1p에게 5장.
20. for (int i = 0; i < 5; ++i)
21. {
22. CCard card = pop\_front\_card();
23. this.distributed\_players\_cards[player\_index].Add(card);
25. this.player\_agents[player\_index].add\_card\_to\_hand(card);
26. }
28. player\_index = find\_next\_player\_index\_of(player\_index);
30. // 2p에게 5장.
31. for (int i = 0; i < 5; ++i)
32. {
33. CCard card = pop\_front\_card();
34. this.distributed\_players\_cards[player\_index].Add(card);
36. this.player\_agents[player\_index].add\_card\_to\_hand(card);
37. }
39. player\_index = find\_next\_player\_index\_of(player\_index);
40. }
42. check\_bonus\_cards();
43. this.floor\_manager.refresh\_floor\_cards();
44. if (!this.floor\_manager.validate\_floor\_card\_counts())
45. {
46. //todo:fatal!!
47. UnityEngine.Debug.LogError("Invalid floor card count!!");
48. return;
49. }
50. }

pop\_front\_card매소드는 중앙의 카드 덱에서 한 장을 뽑아 오는 매소드입니다. 실제로 고스톱을 칠 때도 가운데 카드를 쌓아놓고 한 장씩 빼서 사용하는데 그 기능을 구현한 매소드입니다.

·미리보기 | 소스복사·

1. CCard pop\_front\_card()
2. {
3. return this.card\_queue.Dequeue();
4. }

pop\_front\_card매소드의 구현은 위와 같이 매우 단순합니다. 큐에서 제일 위에 있는 카드를 한 장 꺼내는 코드입니다.

카드 분배가 끝나면 CFloorCardManager의 validate\_floor\_card\_counts매소드를 호출하여 카드가 제대로 분배 되었는지를 체크 합니다.

바닥에 분배된 카드가 8장이 정확하다면 다음으로 넘어갑니다.

만약 코딩에 실수가 있어 8장이 아니라면 오류 메시지를 출력하도록 하였습니다. 이 오류 메시지는 절대 나와서는 안되겠죠.

이번 강좌는 여기 까지 입니다.

다음 강좌에서는 플레이어가 카드를 선택한 뒤의 로직 처리를 구현해 보도록 하겠습니다.

감사합니다.