제목

Assembly-CSharp AlertSound Update()

B – 6조 핫식스 카드 맞추기

{

팀장 김창연

팀원 박성준

팀원 박신환

팀원 이서영

팀원 윤정빈

}

제목 목차

Assembly-CSharp AlertSound Update()

목차

{

시연 영상
역할 분담
스크립팅 전략 및 해설
느낀 점
개선할 점

}

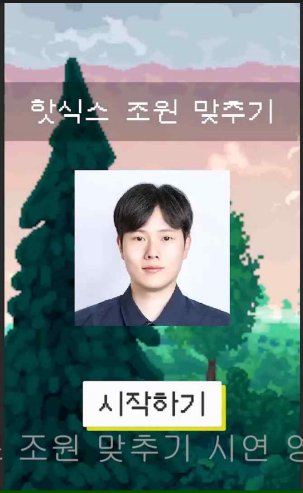
제목 목차 시연 영상

Assembly-CSharp AlertSound Update()

1

시연 영상

2



6조 핫식스 조원 맞추기 시연 영상입니다.

제목 | 목차 | 시연 영상 | 역할 분담

Assembly-CSharp | AlertSound | Update()

역할 분담

기능 구현 목록:
{

이서영
· 카드 이벤트 및 카드 색상 변경 효과
· 효과음 및 배경 음악

윤정빈
· 카드 매치 시 이름 출력

박신환
· 카드 랜덤 배치

김창연
· 게임 시간 제한에 따라 경고
· 애니메이션

박성준
· 화면에 매칭 시도 횟수 출력

}

제목 | 목차 | 시연 영상 | 역할 분담

Assembly-CSharp | AlertSound | Update()

역할 분담

추가기능:

{

이서영

· 카드가 뒤집어지는 모습 연출하기

· 첫번째 카드를 뒤집고 5초 이후 다시 뒤집기

윤정빈

· 실패할 때 시간 감소 효과

박신환

· 랜덤 이벤트를 코드스니펫을 사용하지 않고 직접 만들기

김창연

· 카드 등장 효과 연출

· 매칭시간 시도 횟수 등을 점수로 환산해 결과 점수 출력

· 플레이 중 스테이지 최단 기록 띄우기

박성준

· 스테이지 선택과 해금한 스테이지가 구분되는 화면 만들기

}

제목 | 목차 | 시연 영상 | 역할 분담 | 매칭 시 이벤트

Assembly-CSharp | AlertSound | Update()

매칭 시 이벤트

{

- 매칭 성공 시 처리를 위해 게임매니저에서 Matched 메서드를 만든다.
- 매칭 시도 횟수 담을 변수가 필요하며, Matched 메서드에서 이를 카운트한다.
- 소리를 재생하기 위해 AudioSource와 각종 AudioClip들이 필요
- 실패했을 때 시간 감소 효과도 Matched 메서드에서 처리한다.

}

제목 | 목차 | 시연 영상 | 역할 분담 | 매칭 시 이벤트

Assembly-CSharp | AlertSound | Update()

매칭 시 이벤트

```
if(firstCard.idx == secondCard.idx) // 첫 번째 카드와 두 번째 카드 그림이 같다면
  {
    audioSource.PlayOneShot(matchedSound); // 매칭이 성공했음을 소리로 알리고
    firstCard.DestroyCard(); // 첫 번째 카드를 삭제하고
    secondCard.DestroyCard(); // 두 번째 카드도 삭제하고
    cardCount -= 2; // 카드 개수를 2만큼 줄이고
    SettingNameTxt(); // 그림에 있는 조원 이름을…
    NameTxt.color = Color.white; // 흰색으로 표시하고

    if (cardCount == 0) // 카드가 다 떨어졌다면
    {
      BestTime(); // 최고점수를 기록하고
      endTxt.SetActive(true); // “끝”이라는 텍스트를 띄우고
      score = time / (1 + openCount) * 100; // 점수를 계산하고
      scoreTxt.text = score.ToString( " N2 " ); // 점수를 띄우고
```

매칭 시 이벤트

```
        if (PlayerPrefs.GetString("StageLevel") == "Easy") // 현재 난이도가 쉬움이라면
        {
            PlayerPrefs.SetInt("Easy", 1); // 보통 난이도를 해금하고
        }
        else if (PlayerPrefs.GetString("StageLevel") == "Normal") // 현재 난이도가 보통이라면
        {
            PlayerPrefs.SetInt( " Normal " , 1); // 어려움 난이도를 해금하고
        }

        Time.timeScale = 0.0f; // 시간을 멈춘다.
    }
}
```

타이머 이벤트

{

· 경고 소리와 경고 애니메이션
· 경고 소리를 다루는 게임 오브젝트를 오디오 매니저의 하위 오브젝트로 생성

}

{

어려웠던 점
· 경고 소리를 Update 메서드에서 실행하면 매 프레임마다 겹쳐서 재생하게 됨

}

{

해결 방안
· 경고 소리가 중복 재생되지 않게 하기 위해 Update 메서드에서 직접 실행하면 안됨
· 코루틴과 조건문을 이용해 경고 소리가 중복되지 않게 해결

}

{

· 텍스트 애니메이션은 별도의 스크립트 없이 게임매니저에서 bool 조건값을 통해 transition

}

제목 | 목차 | 시연 영상 | 역할 분담 | 매칭 시 이벤트 | 타이머 이벤트

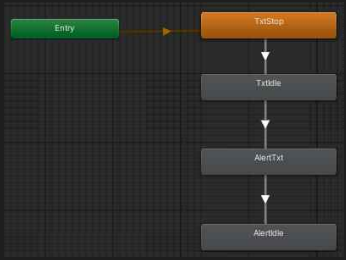
Assembly-CSharp | AlertSound | Update()

타이머 이벤트

```
public class AlertSound : MonoBehaviour
{
    AudioSource audioSource;

    public AudioClip alertSound;

    private bool audioPlayed = false;

    // Start is called before the first frame update
    void Start()
    {
        audioSource = GetComponent<AudioSource>(); // 오디오를 담당하는 컴포
넌트를 얻는다.
        audioSource.clip = this.alertSound; // 경고음이 그 오디오이다.
    }

    private void Update()
    {
        if (GameManager.instance.time <= 10.0f) // 시간이 10초 이하 남았을때
        {
            StartCoroutine(PlayAudio()); // 오디오를 즉시 재생한다.
        }
    }
```

타이머 이벤트

```
    private Ienumerator PlayAudio()
    {
        if(!audioPlayed) // 오디오가 이미 재생되고 있지 않다면
        {
            audioSource.Play(); // 오디오를 재생하고
            audioPlayed = true; // 오디오가 이미 재생되고 있음을 알린다.
        }
        yield return null; // 코루틴 일시정지.
    }
}
```



카드 선택 시 이벤트

{

- Card 클래스에서 다뤄야 한다.
- 하지만 실제로 카운트를 세는 것은 게임매니저에서 수행한다.
  이유는 Card 클래스 자체로는 다른 카드의 개방 유무를 알 수 없기 때문

}

{

어려웠던 점

- 같은 카드를 연속으로 클릭했을 때 게임매니저의 첫번째 카드와 두번째 카드에 같은 카드가 들어가는 현상이 있었다.

}

{

해결 방안

- 클릭 시 카드에 들어있는 버튼을 비활성화했다.

}

제목 | 목차 | 시연 영상 | 역할 분담 | 매칭 시 이벤트 | 카드 선택 시 이벤트

Assembly-CSharp | AlertSound | Update()

카드 선택 시 이벤트

```
public void OpenCard()
{
    if(GameManager.instance.isStart == true) // 게임이 진행중이라면
    {
        audioSource.PlayOneShot(flipSound); // 뒤집힐 때 나는 효과음 재생
        anim.SetBool("isOpen", true); // 뒤집는 애니메이터 활성화
        button.SetActive(false); // 카드에 들어 있는 버튼 비활성화

        if (GameManager.instance.firstCard == null) // 첫 번째 카드를 뒤집었다면
        {
            GameManager.instance.firstCard = this; // 참조가 이 카드를 가리키게 하고
            GameManager.instance.SetTimeAfterFirstCardFlip(MaxTimeAfterFirstCardFlip); // 5초 카운트를 센다. (그 동안 두 번째 카드를 안 뒤집으면 도로 뒤집기)
        }

```

제목 | 목차 | 시연 영상 | 역할 분담 | 매칭 시 이벤트 | 카드 선택 시 이벤트

Assembly-CSharp | AlertSound | Update()

카드 선택 시 이벤트

```
        }
        else // 두 번째 카드를 뒤집었다면
        {
            GameManager.instance.secondCard = this; // 참조가 이 카드를 가리키게 하고
            GameManager.instance.SetTimeAfterFirstCardFlip(ZeroTime); // 카운트를 0초로 초기화한다.
            GameManager.instance.Matched(); // 매칭이 성공했는지 확인하고
            GameManager.instance.openCount++; // 매칭 시도 횟수에 1을 더한다.
        }

        if (!isFlipedOnce) // 이미 한 번 뒤집어본 카드가 아니라면
        {
            isFlipedOnce = true; // 한 번 뒤집었음을 나타내고
            SpriteRenderer spriteRenderer = back.GetComponent<SpriteRenderer>(); // 렌더러 컴포넌트를 구한 다음
```

제목 | 목차 | 시연 영상 | 역할 | 매칭 시 이벤트 | 카드 선택 시 이벤트

Assembly-CSharp | AlertSound | Update()

카드 선택 시 이벤트

```
                spriteRenderer.color = new Color(0.7f, 0.7f, 0.7f, 1); // 회색을
입힌다.
            }
        }
    }
```

난이도 및 카드 랜덤 섞기

{
- Board 클래스를 사용한다.
- 먼저 배열의 원소를 최대로 넣은 다음, 난이도에 따라 앞 부분만 잘라낸다. (Array.Resize)

}

{

어려웠던 점
- 원래 Array.Resize 메서드를 몰라서 리스트를 활용하려 했으나, 이 방법은 기존 코드를 많이 변형시킨다.

}

{
- 랜덤하게 카드를 섞는데 피셔-예이츠(fischer-Yates) 알고리즘을 사용한다.
- 피셔-예이츠 알고리즘을 사용하면 각 순열이 나올 확률이 모두 같으므로, 보다 정확한 셔플을 만들어 준다.

}

난이도 및 카드 랜덤 섞기

Assembly-CSharp | AlertSound | Update()

난이도 및 카드 랜덤 섞기

```
void Start()
{
    // 난이도(1,2,3) 값으로 배열 크기 조정
    Array.Resize(ref arr, PlayerPrefs.GetInt("Difficulty") * 4 + 4); // 난이도에 따라 카드 개수 조절 (쉬움: 8, 보통: 12, 어려움: 16)


    for (int i = 0; i <arr.Length; i += 1) // 피셔-예이츠 알고리즘으로 카드 섞기
    {
        int j = UnityEngine.Random.Range(I, arr.Length); // 이 카드를 포함하여 오른쪽에 있는 카드들 중 하나를 골라…
        if (i != j) // 서로 다른 카드일 경우…
        {
            int temp = arr[i]; // 맞바꾼다.
            arr[i] = arr[j];
            arr[j] = temp;
        }
```

난이도 및 카드 랜덤 섞기 | 스테이지 선택 및 해금

Assembly-CSharp | AlertSound | Update()

스테이지 선택 및 해금

{

- 난이도 선택창은 별도의 씬으로 구현
- 버튼을 눌러 스테이지 돌입
- 난이도를 클리어할 시 다음 난이도 해금
- PlayerPrefs 활용

}

스테이지 선택 및 해금

```
public class StageManager : MonoBehaviour
{
    public Button NormalBtn; // 보통 난이도 버튼
    public Button HardBtn; // 어려움 난이도 버튼
    private void Start()
    {
        if (PlayerPrefs.GetInt("Easy") == 1) // 쉬움 난이도를 깼다면
        {
            NormalBtn.interactable = true; // 보통 난이도 버튼이 활성화되고
        }
        if (PlayerPrefs.GetInt("Normal") == 1) // 보통 난이도를 깼다면
        {
            HardBtn.interactable = true; // 어려움 난이도 버튼이 활성화된다.
        }
    }
}
```

최단 기록 표시

{

- 게임 매니저에서 작동
- PlayerPrefs 활용

}

최단 기록 표시

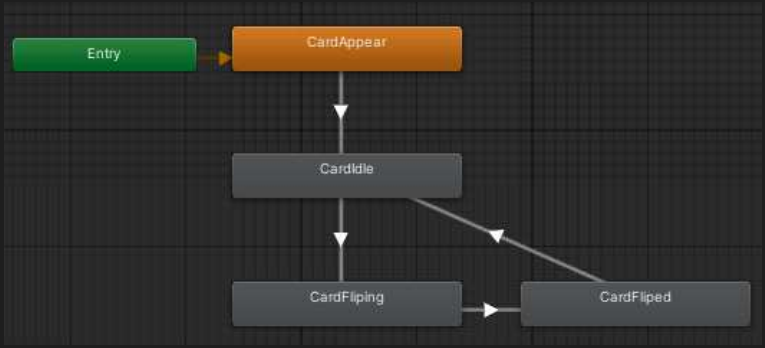
```
1
2   private void BestTime()
3   {
4       nowTime.text = time.ToString("N2"); // 깨는 데 걸린 시간을 표시하고
5       string key = GetKey(); // 현재 난이도의 최고 기록을 얻을 수 있는 키를 구하고
6
7       if (PlayerPrefs.HasKey(key)) // 최고 기록이 기록돼 있다면
8       {
9   경신했다면if (PlayerPrefs.GetFloat(key) < time) // 그 값을 구하고, 최고 기록을
10
11          {
12              PlayerPrefs.SetFloat(key, time); // 덮어쓰고
13              bestTime.text = time.ToString("N2"); // 표시한다.
14          }
15      }
16      else // 최고 기록이 기록돼 있지 않다면
17      {
18          PlayerPrefs.SetFloat(key, time); // 기록하고
19          bestTime.text = time.ToString("N2"); // 표시한다.
20      }
```

카드 뒤집기 연출

{

- 애니메이터 활용
- isOpen이라는 bool 조건값에 따라 transition

}



느낀 점

{

- 객체 지향 프로그래밍이라는게 쉽지 않음을 느꼈다.

- 깃허브를 쓰면서 소통의 중요성을 깨닳았고, 역할 분배가 잘 되어야 프로젝트 진행이 수월하겠다고 느꼈다.

- 혼자 작업할때보다 디버그를 더 자주하게 되고 가독성을 높이려고 노력했던 것 같다.

- 내가 몰랐던 알고리즘 및 팀원들의 코드를 보면서 새로운 것을 배울 수 있었다.

- 프로젝트를 진행하며 해결하는 경험이 좋았다. 강의를 따라가는 것보다 개선방안 같은 고민하는 과정이 필요했다.

- 피곤해서 소통을 잘 못하고 할 일만 했다. 하루에 12시간 과정을 진행면서 체력의 필요성을 느꼈다. 말로 설명하는 법이 어려워 최대한 글로 잘 풀어야겠다.

- 조원들이 각자 구현한 결과를 보면서 내가 구상한 것보다 나은 방식과 가독성의 이점 등 배울 점들이 있었다.

- 개인적인 욕심으로 애니메이션 퀄리티를 올리는데 시간을 많이 소모했다. 결국 수정한 애니메이션을 구현하지 못해 유니티에 대한 이해도가 높았으면 하는 아쉬움이 남았다.

}

느낀 점

{

- 객체지향을 위반한 것 같은 부분이 있었다.
- 이런 문제는 이벤트를 통해 해결할 수 있을 것이다.
- Invoke는 지양되는 편으로 코루틴을 더 적극적으로 활용해야 한다.

}

Thank You