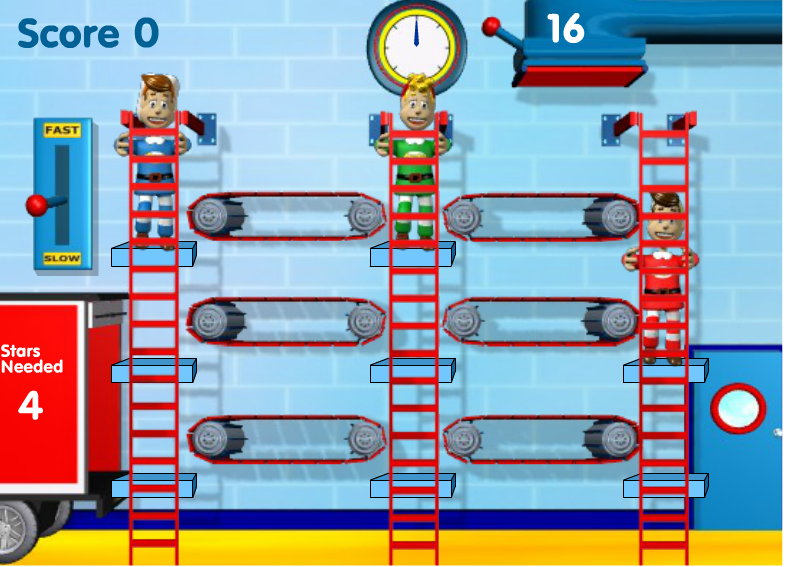
게임프로그래밍 프로젝트 보고서

1.게임 구상



<그림1-원작게임>

본 프로젝트는 <그림1-원작게임>의 테마인 과자공장이 주제인 운반 게임이다. 각 캐릭터를 움직일 때 마우스로 조작하고, 운반을 할 땐 키보드 z,x,c를 이용하도록 구성했다.

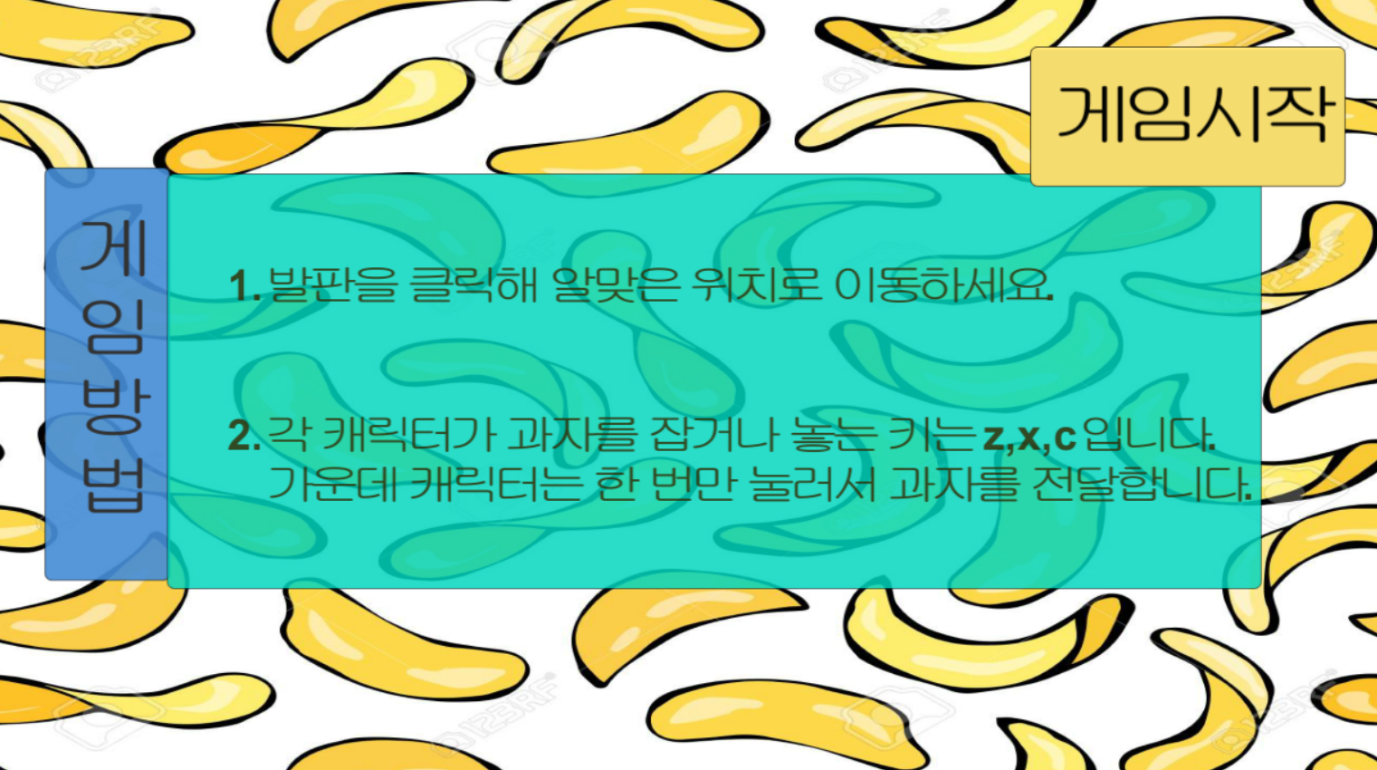
2.장면구성

게임오버

게임실행

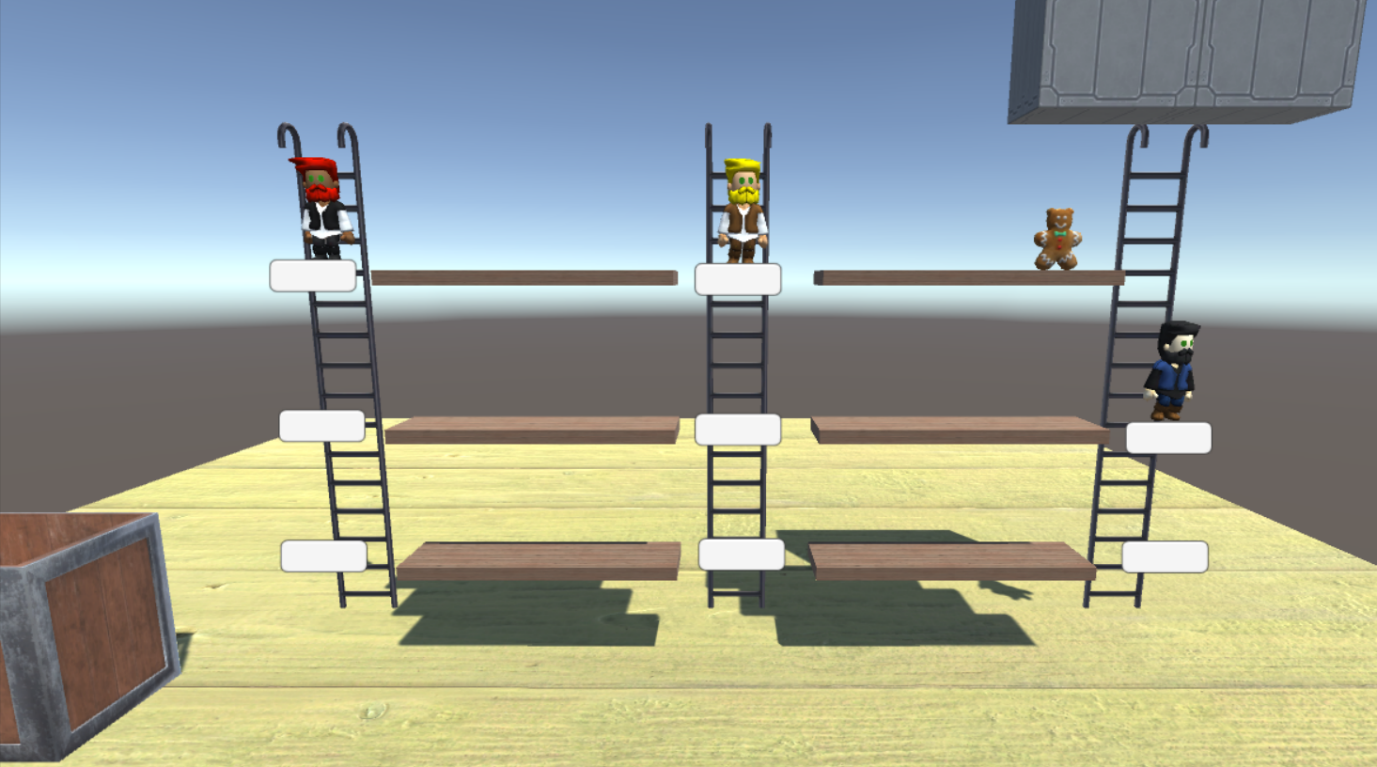
게임방법

a.게임방법



게임조작방법을 안내하고 게임시작 버튼을 눌러 실행하는 화면이다.

b.게임실행



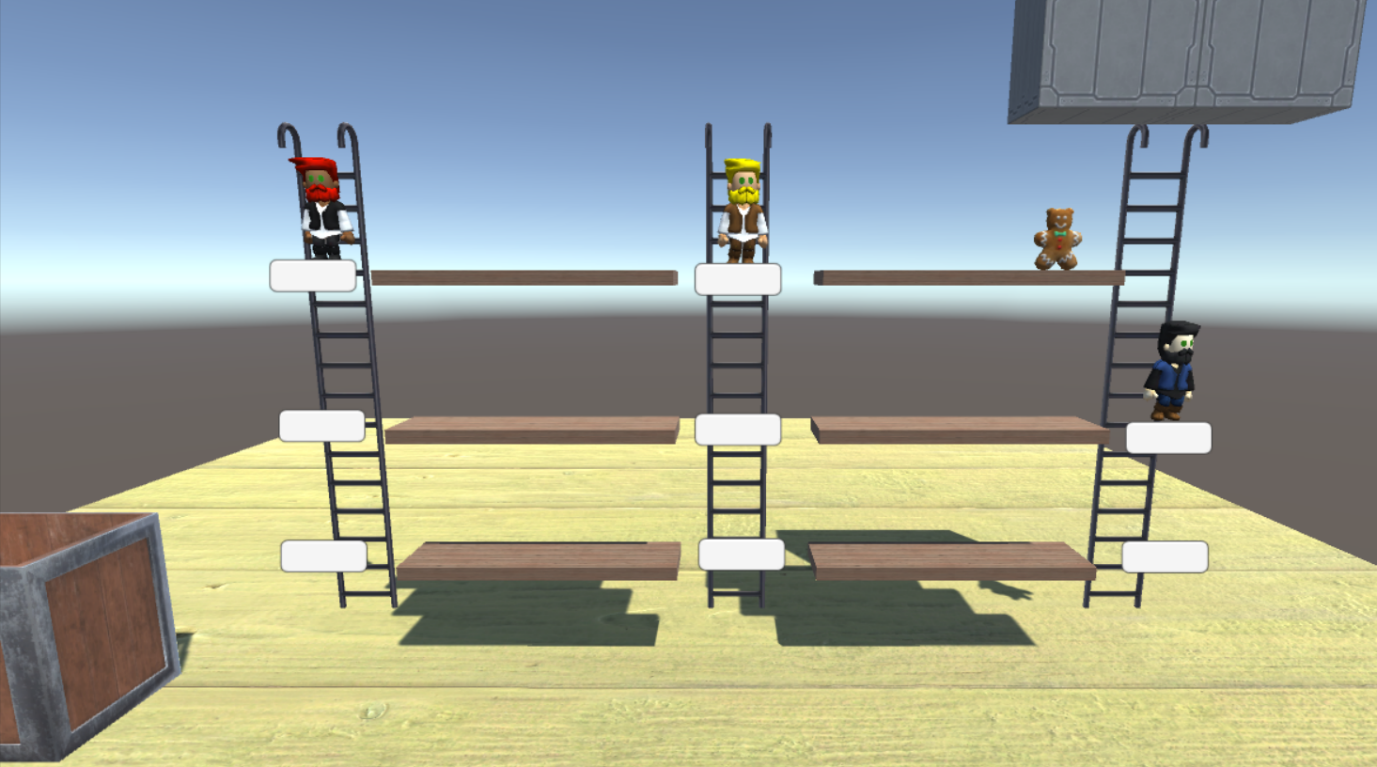
오른쪽 상단의 인풋박스에서 과자가 생성되어 떨어지고, 세 캐릭터들이 과자를 컨베이어벨트 6개를 거쳐 왼쪽 상단의 박스로 옮기도록 한다.

c.게임오버



과자를 놓쳐 바닥으로 떨어지면 게임오버가 된다. 다시시작 버튼을 눌러 b.게임실행 장면으로 돌아갈 수 있다.

3.게임규칙



과자가 오른쪽 상단의 컨베이어 벨트로 떨어지면 가운데 캐릭터가 x를 눌러 과자를 집어서 왼쪽 상단 컨베이어 벨트로 옮긴다. 왼쪽 캐릭터는 과자가 가까워지면 z를 눌러 과자를 집고 사용자는 마우스로 왼쪽 중앙 발판을 클릭해 캐릭터를 옮기고, z를 다시 눌러 과자를 내려놓는다. 그리고 가운데 캐릭터를 가운데 중앙 발판을 클릭해 과자를 받을 위치로 옮기고, x를 눌러 오른쪽 컨베이어벨트로 옮긴다. 오른쪽 캐릭터는 c를 눌러 과자를 집고, 오른쪽 아래 발판을 클릭해 캐릭터를 옮기고 다시 c를 눌러 가운데 캐릭터에게 과자를 전달한다.가운데 캐릭터는 가운데 아래 발판으로 이동해 과자를 왼쪽으로 넘겨줘야 하고, 왼쪽 캐릭터가 왼쪽 아래 발판으로 이동해 과자를 집고, 왼쪽 하단의 운반박스에 던지면 과자 하나의 전달이 완료된다. 과자를 중간에 놓쳐서 바닥과 과자가 충돌하게 된다면 게임오버가 된다. 과자가 생성되어 인풋박스에서 떨어지는 간격은 10초마다 1f씩 줄어든다. 캐릭터를 적재적소에 옮겨놓고, z,x,c로 픽업과 드랍을 헷갈리지 않고 빨리 하는 것이 관건인 게임이다.

4.구현 설명

a.컨베이어 벨트

public class ConveyorBelt : MonoBehaviour {

public GameObject belt;

public Transform endpoint;

public float speed;

void OnTriggerStay(Collider other)

{

other.transform.position = Vector3.MoveTowards(other.transform.position,endpoint.position,

speed\*Time.deltaTime);

}

}

각각의 벨트의 박스컬라이더에 isTrigger를 설정하고, 과자와 닿았을 시에 OnTriggerStay를 실행시켜 과자를 빈게임오브젝트인 endpoint를 향해 이동하도록 설정한다.

b.과자 생성

public class CreateSnack : MonoBehaviour {

public Transform inputPosition;

public GameObject item;

private float period;

private float timer;

void Start () {

period = 13f;

InvokeRepeating("NextPeriod", 10f,10f );

Instantiate(item, inputPosition.position, inputPosition.rotation);

timer = 0f;

}

void Update () {

timer += Time.deltaTime;

if (timer >= period)

{

timer = 0f;

Create();

}

}

void Create()

{

Instantiate(item, inputPosition.position, inputPosition.rotation);

}

void NextPeriod()

{

period -= 1f;

}

}

과자모양의 프리팹은 로테이션에 프리즈를 걸어 과자의 앞모습만 화면에 나오도록 한다.

inputPosition은 과자가 생성되는 위치이다. item은 과자 프리팹을 설정할 변수이다. period는 난이도를 위해 설정한, 생성 간격 변수이다. timer는 period에 따라 생성 시간이 달라지도록 update함수에서 쓸 변수이다.

start함수에서 period를 13f로 설정한다. 과자를 맨 처음 생성하고, 반복함수를 호출해 NextPeriod에서 10초 후-그 후 10초마다 period를 1씩 줄이도록 한다. timer를 0으로 설정한다.

Update함수에서 timer가 period와 같게 되면 타이머를 초기화시키고 Create()를 호출해 과자를 생성한다. period가 10초마다 줄어듦에 따라 Create()가 호출되는 속도 역시 빨라진다.

c.캐릭터의 픽업/드랍과 발판 버튼

public class PickAtCenter : MonoBehaviour {

private GameObject item;

public GameObject tempParent;

public Transform guide;

public bool twoToOne;

public bool threeToFour;

public bool sixToFive;

public Transform oneStart;

public Transform fourStart;

public Transform fiveStart;

void Start()

{

twoToOne = true;

threeToFour =false;

sixToFive = false;

}

void Update()

{

findNearestItem();

if ((Input.GetKeyDown("x") && Vector3.Distance(item.transform.position,guide.transform.position)<=1.5f))

{

pickup(); drop();

}

}

void pickup()

{

item.GetComponent<Rigidbody>().useGravity = false;

item.GetComponent<Rigidbody>().isKinematic = true;

item.transform.position = guide.transform.position;

item.transform.parent = tempParent.transform;

}

void drop()

{

item.GetComponent<Rigidbody>().useGravity = true;

item.GetComponent<Rigidbody>().isKinematic = false;

item.transform.parent = null;

if (twoToOne == true)

{

item.transform.position = oneStart.transform.position;

}

else if (threeToFour == true)

{

item.transform.position = fourStart.transform.position;

}

else if (sixToFive == true)

{

item.transform.position = fiveStart.transform.position;

}

}

void findNearestItem()

{

item=GameObject.FindGameObjectsWithTag("snack").Aggregate((o1, o2) =>

Vector3.Distance(o1.transform.position, this.transform.position) >

Vector3.Distance(o2.transform.position, this.transform.position) ? o2 : o1);

}

}

각 캐릭터에 적용되는 PickAtLeft,PickAtCenter,PickAtRight가 있지만 기능은 유사하니 PickAtCenter를 대표적으로 보면 item은 캐릭터가 픽업/드랍해야 할 과자를 의미한다. tempParent는 캐릭터객체이다. 픽업했을 시 과자를 tempParent의 자식으로 설정한다.guide는 과자가 픽업돼서 캐릭터의 자식이 되었을 때 어디에 위치할 것인지 빈게임오브젝트로 설정한 변수이다. bool 변수 세 개는, 캐릭터의 현재 위치가 두번째에서 첫번째로 과자를 옮길 수 있는 상태인가-세번째에서 네번째로 옮길 수 있는 상태인가-여섯번째에서 다섯번째로 옮길 수 있는 상태인가를 알려주는 변수이다. 가운데 캐릭터는 처음에 맨위에 있으므로 start()에서 twoToOne을 true로 설정했다. Update()에서는 findNearestItem()를 호출하는데, 이것은 과자 프리팹이 두개 이상 게임실행화면에 있을 시에, 캐릭터가 어떤 오브젝트를 집어야 할 지 모르기 때문에 캐릭터와 제일 가까이 있는 오브젝트를 설정하기 위해 만든 함수이다.

캐릭터와 과자가 1.5f이하로 가까이 있고, 사용자가 픽업과 드랍을 위한 x버튼을 눌렀을 시 가운데 캐릭터는 픽업과 드랍을 동시에 하게 된다. pickup()에서 과자가 들리는 것을 표현하기 위해 중력을 무력화 하고 키네마틱을 true로 했다.그리고 캐릭터와 동일한 위치에 있는 guide의 위치에 과자를 위치시키고, 들린 순간은 캐릭터의 자식이 되도록 했다. drop()에서는 중력과 키네마틱을 반대로 만들어 물리법칙을 다시 받도록 하고, 과자가 부모로부터 독립되게 하였다. 그리고 bool 변수를 보고 이쪽에서 저쪽으로 옮길 수 있는 상태인지 검사한다. 캐릭터가 알맞은 곳에 있는지 확인하는 과정이다. 맞는 위치에 있다면 과자를 다음 컨베이어벨트의 시작위치인 -start 위치로 옮겨놓는다.

public class CenterTop : MonoBehaviour {

public PickAtCenter pick;

public GameObject centerMan;

public Transform ctPosition;

public void OnClickCenterTop () {

pick = GameObject.Find("CenterMan").GetComponent<PickAtCenter>();

centerMan.transform.position = ctPosition.position;

pick.twoToOne = true;

pick.threeToFour = false;

pick.sixToFive = false;

}

}

각 발판 버튼에 적용되는 이벤트함수이다. 대표적으로 가운데 위쪽 버튼의 스크립트를 보면, PickAtCenter클래스의 pick변수를 불러오고있다. 이것은 어떤 버튼을 클릭하는게 곧 특정 캐릭터가 자신이 집은 과자를 다른 컨베이어벨트로 옮길 수 있는 상태가 되는 것을 의미하기 때문에, PickAtCenter의 bool함수 3개의 상태를 변경시켜야 하기 때문이다.

캐릭터의 위치를 변경시키고 그 캐릭터의 pick 현황을 설정한다.

d.바닥 충돌이벤트

public class GameOver : MonoBehaviour {

void OnCollisionEnter(Collision collision)

{

Destroy(collision.gameObject, 0f);

GetComponent<SceneTrans1>().SceneGameover();}}

바닥에 충돌이벤트를 설정한다. 플레이어가 과자를 놓쳐서 과자가 바닥에 떨어지게 되면, 과자 프리팹을 없애는 동시에 SceneTrans의 장면전환 함수를 호출해 게임오버 화면으로 넘어간다.