DIVISION OF
DIGITAL CONTENTS
DONGSEO UNIVERSITY

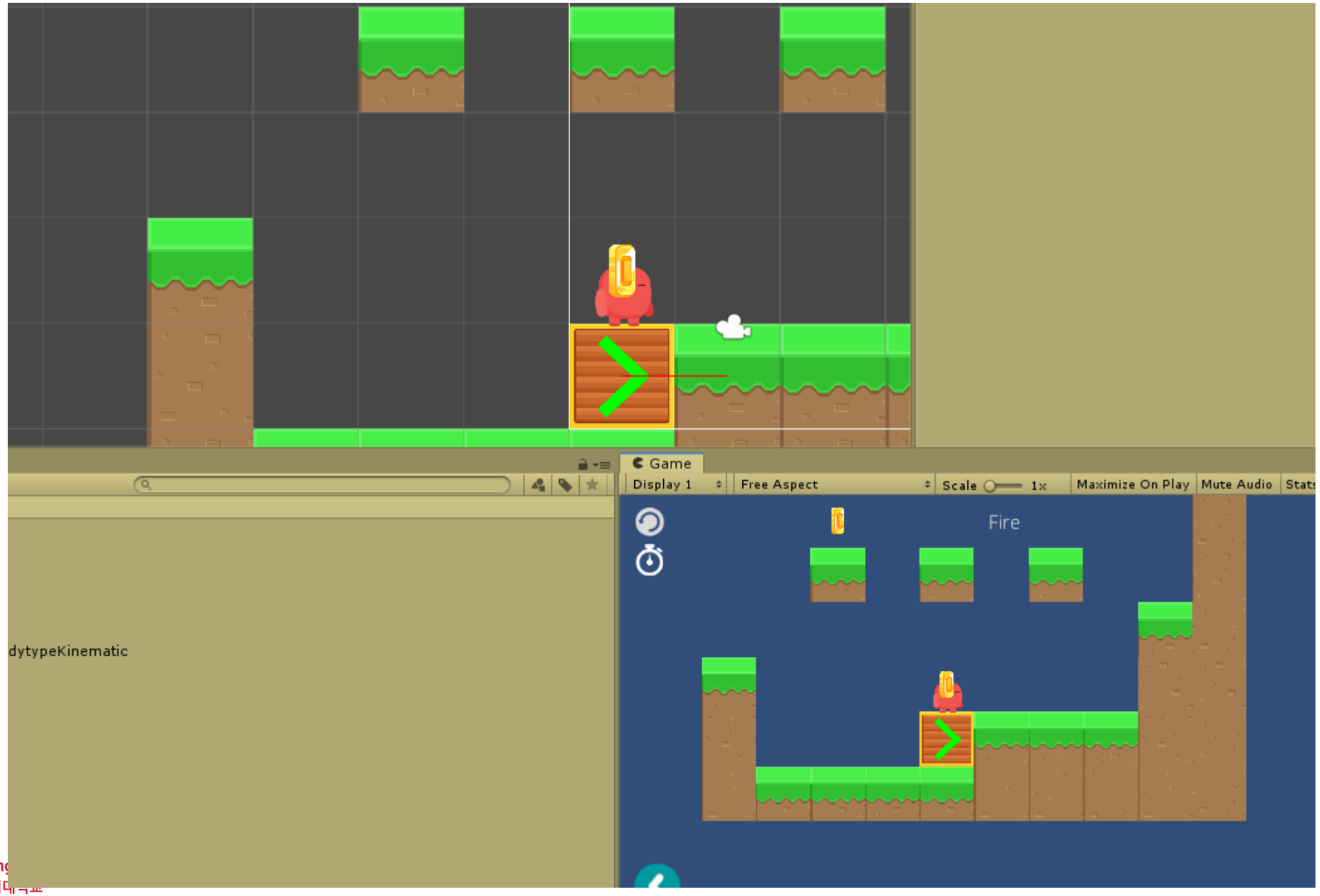
Dangerous Kave

Coin Acquisition Bug and More

jintaeks@dongseo.ac.kr

April, 2020

DSU Dongseo University 동서대학교


Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Fire
dytypeKinematic

```
180            _numCollision = hits.Length;
181
182            foreach (Collider2D hit in hits)
183            {
184                // Ignore our own collider.
185                if (hit.transform == transform)
186                    continue;
187
188                if (hit.transform.CompareTag("Player") || hit.transform.CompareTag("Box") || hit.tra
189                {
190                    stateTimer = 0.0f; // initialize timer when there is a collision with 'Player'
```

100 %

Watch 1

Name	Value	Type
hits	UnityEngine.Collider2D[3]	UnityEngine.Col...
[0]	"Box (2) (UnityEngine.BoxCollider2D)"	UnityEngine.Col...
[1]	"Tilemap (UnityEngine.TilemapCollider2D)"	UnityEngine.Col...
[2]	"Blob (UnityEngine.BoxCollider2D)"	UnityEngine.Col...

Breakpoints

New | Show Columns

Name	Labe
BoxController.cs, line 142 character 1	
BoxController.cs, line 182 character 1	
CharacterController2D.cs, line 116 character 1	

```
114        _hitCount = hits.Length;
115
116        foreach (Collider2D hit in hits)
117        {
118            // Ignore our own collider.
119            if (hit.transform == transform)
120                continue;
121
122            isJumping = false;
123            ColliderDistance2D colliderDistance = hit.Distance(_boxCollider);
124
```

100 %

Watch 1

Name	Value	Type
hits	UnityEngine.Collider2D[2]	UnityEngine.Col...
[0]	"Box (2) (UnityEngine.BoxCollider2D)"	UnityEngine.Col...
[1]	"Blob (UnityEngine.BoxCollider2D)"	UnityEngine.Col...

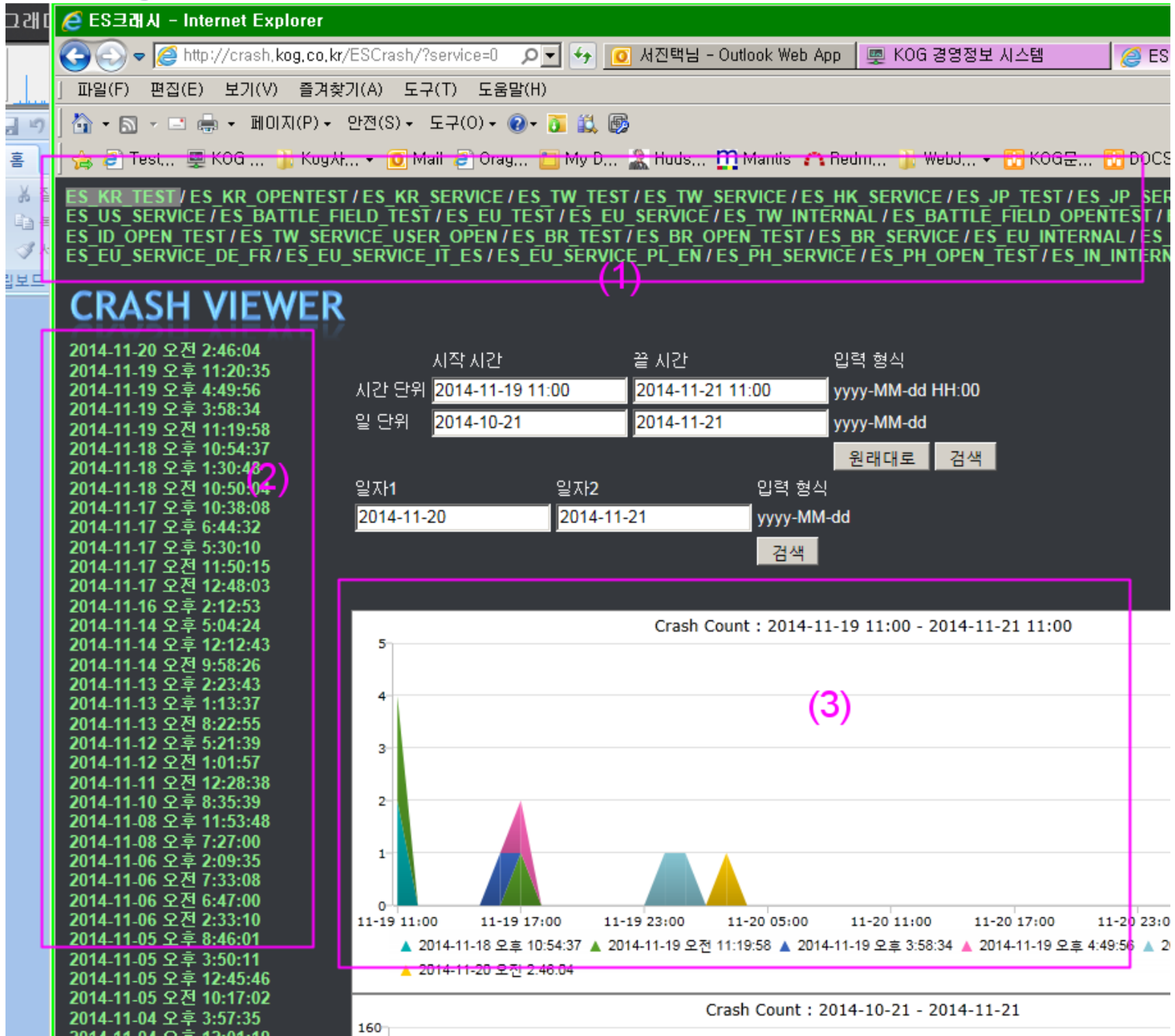
Breakpoints

New | Show Columns

Name | Lal

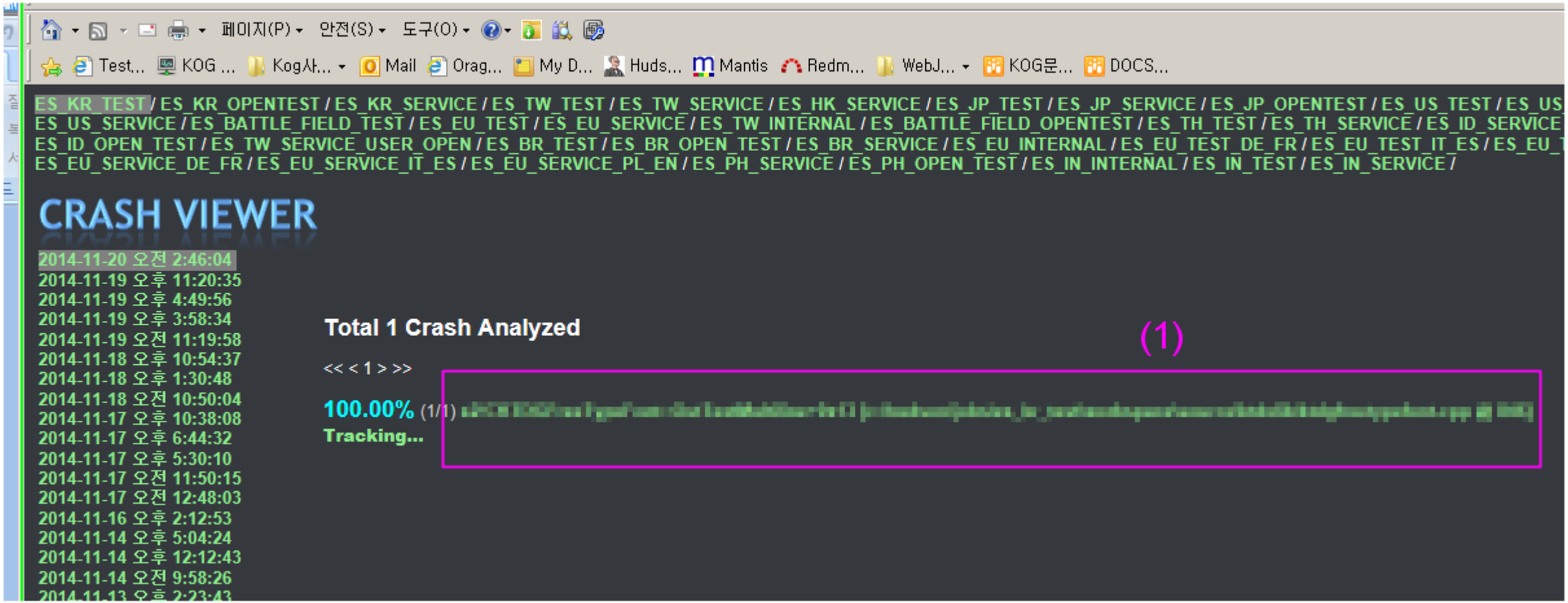
- BoxController.cs, line 142 character 1
- BoxController.cs, line 182 character 1
- **CharacterController2D.cs, line 116 character 1**

Crash Report System



DSU Dongseo University 동서대학교

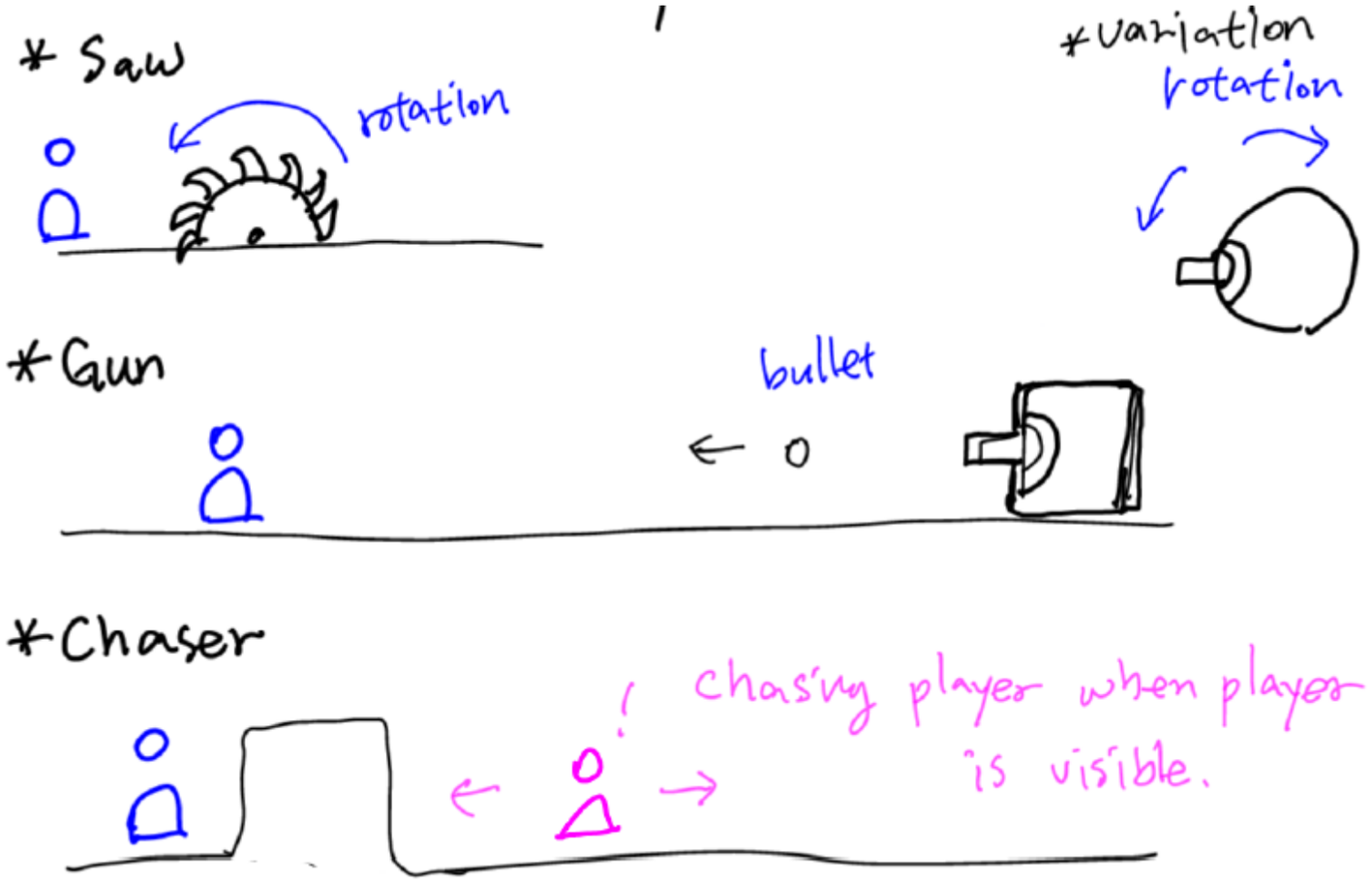
Crash Report System: Source Level Check



CI(continuous integration) : Hudson, Jenkins

Issue Management System : Mantis, Redmine

Mechanics can move around.



Need to maintain existing mechanics

```
[Range(0, 360)]
   public float _velocityDegree;
   [Range(0, 360)]
   public float[] _velocityDegrees;
   private int _currentVelocityIndex = 0;
```

```
if (dist <= 0.2f)
    {
        if (_velocityDegrees.Length >= 1)
        {
            _currentVelocityIndex += 1;
            _currentVelocityIndex %= _velocityDegrees.Length;
            //if (_currentVelocityIndex >= _velocityDegrees.Length)
            //    _currentVelocityIndex = 0;
            _velocityDegree = _velocityDegrees[_currentVelocityIndex];
            _curVelocity = new Vector2(0, 0);
            UpdateMaxVelocity();
            _stateTimer = 0.0f;
            _posQueue.ClearAll();
        }
        else
```

Need common code for the Saw and Box

```
if (hit.transform.CompareTag("Player") || hit.transform.CompareTag("Box"))
    //if(hit.gameObject.IsMovingObject())
    {
        isMovingState = true;
        break;
    }
```

Using the Bookmark window in Visual Studio

```
128         // Retrieve all colliders we have intersected after velocity has bee
129         Collider2D[] hits = Physics2D.OverlapBoxAll(transform.position, _box
130         foreach (Collider2D hit in hits)
131         {
132             // Ignore our own collider.
133             if (hit.transform == transform)
134                 continue;
135
136             if (hit.transform.CompareTag("Player") || hit.transform.CompareTag("Box"
137             //if(hit.gameObject.IsMovingObject())
138             {
139                 isMovingState = true;
140                 break;
141             }
142
143             ColliderDistance2D colliderDistance = hit.Distance(_boxCollider);
144             if (colliderDistance.isOverlapped)
```

Bookmarks

Bookmark	File Location	Line Number
Bookmark1	D:\github\DangerousKave\UnityDangerousKave\Assets\Scripts\BoxController.cs	251
Bookmark2	D:\github\DangerousKave\UnityDangerousKave\Assets\Scripts\BoxController.cs	188
Bookmark3	D:\github\DangerousKave\UnityDangerousKave\Assets\Scripts\BoxController.cs	136
Bookmark4	D:\github\DangerousKave\UnityDangerousKave\Assets\Scripts\BoxController.cs	177

```
public class ObjectProperty : MonoBehaviour
{
    public bool isMoving = false;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }
}
```

How to extend the GameObject?

```
namespace UnityEngine
{
    //
    // Summary:
    //     Base class for all entities in Unity Scenes.
    [ExcludeFromPreset]
    [NativeHeader("Runtime/Export/GameObject.bindings.h")]
    [UsedByNativeCode]
    public sealed class GameObject : Object
    {
        public GameObject();
        public GameObject(string name);
```

How to extend the GameObject?

```
public static class GameObjectExtensions
{
    public static bool IsMovingObject(this GameObject go)
    {
        ObjectProperty prop = go.GetComponent<ObjectProperty>();
        if (prop)
            return prop.isMoving;
        return false;
    }
}//public static class GameObjectExtensions
```

Generalized Box Behaviour

BoxBehaviour

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
using KaveUtil;

[RequireComponent(typeof(BoxCollider2D))]
public class BoxBehaviour : MonoBehaviour
{
    public enum EState
    {
        IDLE,
        PREMOVING,
        MOVING
    }

    public float _speed = 1;
    [Range(0, 360)]
    public float _velocityDegree;
    [Range(0, 360)]
    public float[] _velocityDegrees;
    private int _currentVelocityIndex = 0;

    private GameObject _player;

    [SerializeField]
    private BoxCollider2D _boxCollider;
    [SerializeField]
    private Transform _boxArrow;
    private SpriteRenderer _arrowSprite;
```

```
private float _acceleration = 1;
    private Vector2 _curVelocity = new Vector2(0, 0);
    private Vector2 _maxVelocity = new Vector2(1, -1);
    private int _numCollision = 0;
    private bool _isGrounded = false;
    private bool _isInAir = false;

    private EState _movingState = EState.IDLE;
    private float _stateTimer = 0.0f;
    private CircularQueue<Vector2> _posQueue = new CircularQueue<Vector2>(10);
    private float _posQueueInsertTimer = 0.0f; // insert position for every 0.1 second.

    private void Awake()
    {
        _player = GameObject.FindGameObjectWithTag("Player");
        CharacterController2D cc2d = _player.GetComponent<CharacterController2D>();
    }
    void Start()
    {
        _boxCollider = GetComponent<BoxCollider2D>();
        Debug.Assert(_boxCollider != null);
        _boxArrow = transform.Find("Box Arrow");
        if (_boxArrow)
        {
            _arrowSprite = _boxArrow.gameObject.GetComponent<SpriteRenderer>();
        }
        OnStart();
        UpdateMaxVelocity();
    }
```

```
void Update()
    {
        _stateTimer += Time.deltaTime;
        if (_movingState == EState.IDLE)
        {
            _Update_StateIDLE();

            SetArrowSpriteColor(Color.white);

        }
        else if (_movingState == EState.PREMOVING)
        {
            SetArrowSpriteColor(Color.black);
            if (_stateTimer >= 1.0f)
            {
                _movingState = EState.MOVING;
                _stateTimer = 0.0f;
            }
        }
        else if (_movingState == EState.MOVING)
        {
            _Update_StateMOVING();
            //c.g = Mathf.PingPong(_stateTimer, 0.5f);
            SetArrowSpriteColor(Color.green);
        }
        /*virtual*/OnUpdate(_movingState, _stateTimer);
    }
```

```
void _Update_StateIDLE()
    {
        bool isMovingState = false;

        // Retrieve all colliders we have intersected after velocity has been applied.
        RaycastHit2D[] hits = Physics2D.BoxCastAll(transform.position, _boxCollider.size, 0, new
Vector2(0, 0));
        if (hits.Length >= 1)
        {
            foreach (RaycastHit2D hit in hits)
            {
                // Ignore our own collider.
                if (hit.transform == transform)
                    continue;

                if (hit.transform.gameObject.IsMovingObject())
                {
                    isMovingState = true;
                    break;
                }
            }//foreach
        }

        if (isMovingState)
        {
            //_movingState = EState.PREMOVING;
            _movingState = EState.MOVING; // test _20200328_jintaeks
            _stateTimer = 0.0f;
        }
    }
```

```
void _StateMOVING_UpdateCollision()
    {
        /*virtual*/ OnPreCollision();

        _curVelocity = Vector2.MoveTowards(_curVelocity, _maxVelocity, _acceleration * Time.deltaTime);
        transform.Translate(_curVelocity * Time.deltaTime);

        _isGrounded = false;
        bool isInAir = true;

        // Retrieve all colliders we have intersected after velocity has been applied.
        Collider2D[] hits = Physics2D.OverlapBoxAll(transform.position, _boxCollider.size, 0);

        _numCollision = hits.Length;

        foreach (Collider2D hit in hits)
        {
            // Ignore our own collider.
            if (hit.transform == transform)
                continue;

            if( hit.gameObject.IsMovingObject())
            {
                _stateTimer = 0.0f; // initialize timer when there is a collision with 'Player' or 'Box'
            }

            isInAir = false;
            ColliderDistance2D colliderDistance = hit.Distance(_boxCollider);
```

```
 if (colliderDistance.isOverlapped)
            {
                /*virtual*/ OnOverlapped( hit, colliderDistance);

                // If we intersect an object beneath us, set grounded to true.
                if (Vector2.Angle(colliderDistance.normal, Vector2.up) < 90 && _curVelocity.y
< 0)
                {
                    _isGrounded = true;
                }
            }
        }

        if (isInAir != _isInAir)
        {
            _isInAir = isInAir;
        }

        /*virtual*/ OnPostCollision();
    }
```

```
void _Update_StateMOVING()
    {
        _StateMOVING_UpdateCollision();

        // check possible next state
        _posQueueInsertTimer += Time.deltaTime;
        if (_posQueueInsertTimer >= 0.1f)
        {
            Vector2 pos = transform.position;
            _posQueue.Insert(pos);
            _posQueueInsertTimer -= 0.1f;
        }
        if (_stateTimer >= 1.0f)
        {
            Vector2 vFront;
            Vector2 vRear;
            bool bFront = _posQueue.GetFront(out vFront);
            bool bRear = _posQueue.GetRear(out vRear);
            if (bFront && bRear)
            {
                float dist = Vector2.Distance(vFront, vRear);
                if (dist <= 0.2f)
                {
                    if (_velocityDegrees.Length >= 1)
                    {
                        _currentVelocityIndex += 1;
                        _currentVelocityIndex %= _velocityDegrees.Length;
                        _velocityDegree = _velocityDegrees[_currentVelocityIndex];
                        _curVelocity = new Vector2(0, 0);
```

```
UpdateMaxVelocity();
                        _stateTimer = 0.0f;
                        _posQueue.ClearAll();
                    }
                    else
                    {
                        _movingState = EState.IDLE;
                        _posQueue.ClearAll();
                        _stateTimer = 0.0f;
                    }//if.. else..
                }
            }
        }
    }
```

```
void UpdateMaxVelocity()
    {
        _maxVelocity = Util.Rotate(Vector2.right, _velocityDegree * Mathf.Deg2Rad) * _speed;
        Quaternion q = Quaternion.AngleAxis(_velocityDegree, Vector3.forward);
        if( _boxArrow )
            _boxArrow.transform.SetPositionAndRotation(_boxArrow.transform.position, q);
    }

    public void SetArrowSpriteColor(Color c)
    {
        if( _arrowSprite )
            _arrowSprite.color = c;
    }
    public void SetState(EState state)
    {
        _movingState = state;
        _stateTimer = 0.0f;
    }
    public void OnExternalCollision(GameObject gameObject)
    {
        if( gameObject.CompareTag("Player"))
        {
            if (_movingState == EState.IDLE)
            {
                _movingState = EState.MOVING; // test _20200328_jintaeks
                _stateTimer = 0.0f;
            }//if
        }
    }
```

```
 virtual public void OnUpdate(EState movingState, float stateTimer)
    {
    }

    virtual public void OnPreCollision()
    {
    }

    virtual public void OnOverlapped(Collider2D hit, ColliderDistance2D
colliderDistance)
    {
    }

    virtual public void OnPostCollision()
    {
    }

    virtual public void OnStart()
    {
    }
}//class BoxController
```

BoxController

```
[RequireComponent(typeof(BoxCollider2D))]
public class BoxController : BoxBehaviour
{
    private bool _isContactSaw = false;
    private float _sawContactTimer = 0f;
    [SerializeField]
    private GameObject _grindEffect;
    private GameObject _grindEffectInstance = null;
    private Vector2 _grindContactPos;
    private bool isContactSawTemp = false;

    override public void OnUpdate(EState movingState, float stateTimer)
    {
        if (movingState == EState.MOVING)
        {
            if (_isContactSaw)
            {
                _sawContactTimer += Time.deltaTime;
                SetArrowSpriteColor(Color.red);
                if (_sawContactTimer >= 1.0f)
                {
                    LevelManager.CreateEffect(LevelManager.EffectType.BigImpact, transform.position,
transform.rotation);
                    Destroy(gameObject);
                    Destroy(_grindEffectInstance);
                }
            }
        }
    }
```

```
    override public void OnPreCollision()
    {
        isContactSawTemp = false;
    }
    override public void OnOverlapped(Collider2D hit, ColliderDistance2D colliderDistance)
    {
        if (hit.transform.CompareTag("Saw"))        {
            isContactSawTemp = true;
            _grindContactPos = colliderDistance.pointB;
        }

        if (hit.transform.CompareTag("Player"))        {
            hit.transform.Translate(colliderDistance.pointA - colliderDistance.pointB);
        }
        else        {
            transform.Translate(colliderDistance.pointA - colliderDistance.pointB);
        }
    }

    override public void OnPostCollision()
    {
        if (_isContactSaw != isContactSawTemp)        {
            _isContactSaw = isContactSawTemp;
            if (_isContactSaw && _grindEffectInstance == null)
                _grindEffectInstance = Instantiate(_grindEffect, _grindContactPos, transform.rotation);
            if (_grindEffectInstance)
                _grindEffectInstance.SetActive(_isContactSaw);
        }
    }
}//class BoxController
```

Moving Saw Mechanic

```
public class SawController : BoxBehaviour
{
    override public void OnStart()
    {
        SetState(EState.MOVING);
    }

    override public void OnOverlapped(Collider2D hit, ColliderDistance2D
colliderDistance)
    {
        if (hit.gameObject.IsMovingObject())
        {
            hit.transform.Translate(colliderDistance.pointA -
colliderDistance.pointB);
        }
        else
        {
            transform.Translate(colliderDistance.pointA - colliderDistance.pointB);
        }
    }
}
```

MY **BRIGHT** FUTURE

DSU Dongseo University 동서대학교