DIVISION OF
DIGITAL CONTENTS
DONGSEO UNIVERSITY

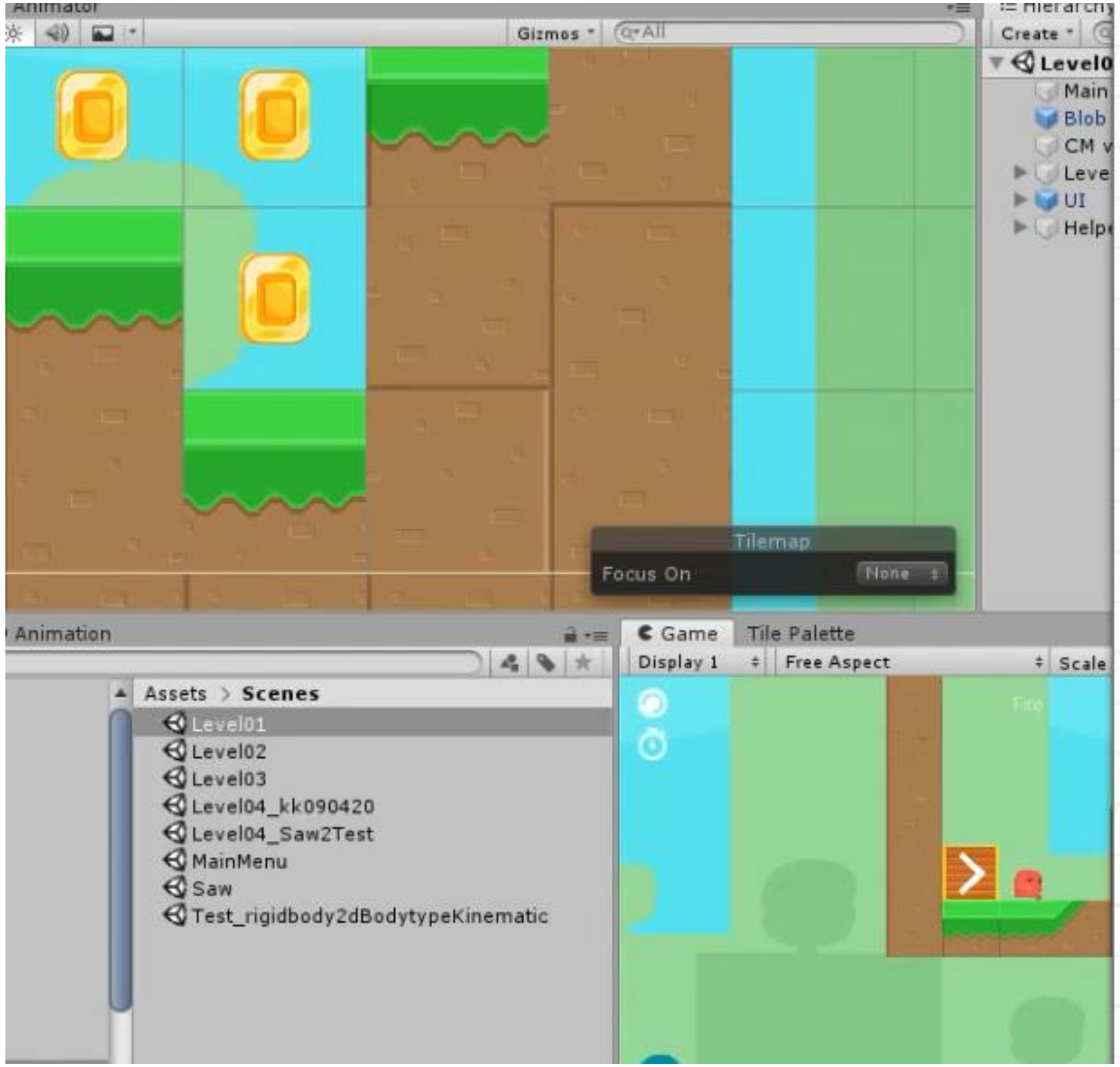
Dangerous Kave 11

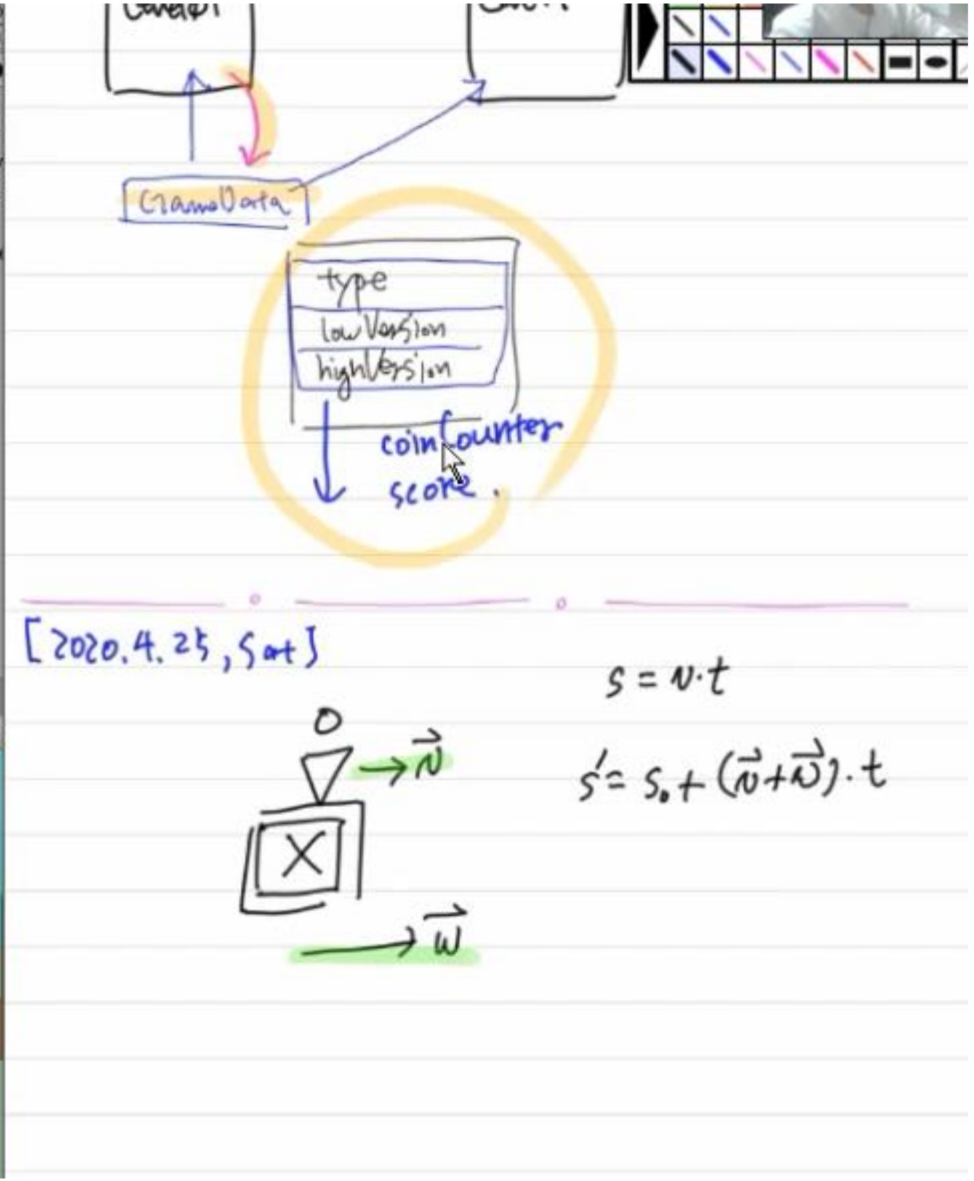
Movement on a moving platform

jintaeks@dongseo.ac.kr

May, 2020

Dongseo University
동서대학교





[2020.4.25, Sat]

$s = v \cdot t$

$s' = s_0 + (\vec{v}+\vec{w}) \cdot t$

BoxBehaviour

```
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; // index to '_velocityDegrees[]'

    private Vector2 _instantaneousVelocity = Vector2.zero;
    public Vector2 Velocity
    {
        get { return _instantaneousVelocity; }
    }

    private GameObject _player; // reference to the player character
```

Dongseo University
동서대학교

```
void Update()
    {
        Vector3 oldPos = transform.position;

        _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;
...
        Vector3 newPos = transform.position;
        _instantaneousVelocity = (newPos - oldPos)/ Time.deltaTime;
        /*virtual*/OnUpdate(_movingState, _stateTimer);
    }
```

Dongseo University
동서대학교

```
 void _StateMOVING_UpdateCollision()
    {
...

        // 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;
```

BoxController

```
 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);
                }
            }
        }
    }
```

KaveUtil

```
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 Vector2 GetVelocity(this GameObject go)
    {
        if (go.CompareTag("Box") || go.CompareTag("Saw"))
        {
            BoxBehaviour boxBehav = go.GetComponent<BoxBehaviour>();
            if (boxBehav)
                return boxBehav.Velocity;
        }
        return Vector2.zero;
    }
}//public static class GameObjectExtensions
```

LevelManager

```
public static bool IsOverlapWithWorld(Vector2 p, Transform owner, ref Collider2D hitOut )
   {
       bool isOverlap = false;
       if (_tilemap2d != null)
           isOverlap = _tilemap2d.OverlapPoint(p);
       if (isOverlap)
       {
           hitOut = null; // set [ref] parameter
           return true;
       }

       Collider2D[] hits = Physics2D.OverlapPointAll(p);
       foreach (Collider2D hit in hits)
       {
           if (hit.transform == owner)
               continue;
```

Dongseo University
동서대학교

CharacterController2D

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

[RequireComponent(typeof(BoxCollider2D))]
public class CharacterController2D : MonoBehaviour
{
    enum CornerId
    {
        Left,
        Right,
        Top,
        Bottom, // 3
        LeftBottom, // 4
        RightBottom, // 5
        MAX
    }

    enum InternalEventType
enum InternalEventType
   {
       DestroyCharacter,
       CornerCollision,
       MAX
   }

   struct InternalEvent
   {
       public InternalEventType eventType;
       public GameObject go;
       public int iParam;
   }
```

```
private bool  isGrounded = false;
   private Vector2 _groundVelocity = new Vector2(0, 0);
   private bool _isJumping = false;
   private bool _isFacingRight = true;
   private int _hitCount = 0;
   private CornerData[] _cornerData = new CornerData[6];
   private int _numBottomColl = 0;
   private Queue<InternalEvent> _internalEvents = new Queue<InternalEvent>();

   void Awake()
   {
       _boxCollider = GetComponent<BoxCollider2D>();
       _InitializeCornerData();
       _maxFallingVelocity = _speed * _walkAcceleration;
   }
```

```
    if ((moveInput > 0 && _isFacingRight == false) || (moveInput < 0 && _isFacingRight ==
true))
            Flip(moveInput);

        if (_isGrounded  )
        {
            _velocity.y = 0;

            if (Input.GetButtonDown("Jump") && _isJumping == false)
            {
                _velocity.x += _groundVelocity.x;
                // Calculate the velocity required to achieve the target jump height.
                _velocity.y = Mathf.Sqrt(2 * _jumpHeight * Mathf.Abs(Physics2D.gravity.y));
            }
        }

        float acceleration = _isGrounded ? _walkAcceleration : _airAcceleration;
```

```
_UpdatePointCollInfo();
        if (_cornerData[( int )CornerId.Top].isCornerColl && _velocity.y > 0)
        {
            //Collider2D coll2d = _cornerData[( int )CornerId.Top].cornerCollider2D;
            //if (coll2d.transform.CompareTag("Box"))
            //{
            //    BoxBehaviour boxBehavior = coll2d.gameObject.GetComponent<BoxBehaviour>();
            //    if (boxBehavior)
            //    {
            //        boxBehavior.DoExternalCollision(gameObject);
            //    }
            //}
            _velocity.y = -_velocity.y;
        }

        Vector2 v = _velocity;
        if (_isGrounded)
            v += _groundVelocity;
        transform.Translate(v * Time.deltaTime);
        if (isJumping != _isJumping)
        {
            _isJumping = isJumping;
            OnJumping(isJumping);
        }

        if (bDestroyCharacter)
        {
            _AddInternalEvent(new InternalEvent() {
eventType=InternalEventType.DestroyCharacter});
        }
        _ProcessInternalEvent();
    }
```

_UpdatePointCollInfo()

```
void _UpdatePointCollInfo()
   {
       _numBottomColl = 0;
       for (int i = 0; i < (int)CornerId.MAX; ++i)
       {
           Vector2 pos = transform.position;
           _cornerData[i].isCornerColl
               = LevelManager.IsOverlapWithWorld(pos + _cornerData[i].cornerOffset
                   , transform, ref _cornerData[i].cornerCollider2D);
           if (_cornerData[i].isCornerColl)
           {
               Collider2D coll2d = _cornerData[i].cornerCollider2D;
               if (_IsBottomCornerId(i))
                   _numBottomColl += 1;
               if (coll2d)
               {
```

Dongseo University 동서대학교

```
 void _UpdatePointCollInfo()
...
                    _numBottomColl += 1;
                if (coll2d)
                {
                    InternalEvent ie = new InternalEvent()
                    {
                        eventType = InternalEventType.CornerCollision,
                        go = coll2d.gameObject,
                        iParam = i
                    };
                    _AddInternalEvent(ie);
                }
            }
        }//for
        //if (_isJumping)
        //{
        //    if (_cornerData[0].isCornerColl || _cornerData[1].isCornerColl)
        //        _velocity.x = 0;
        //}
    }
    bool _IsBottomCornerId(int id)
    {
        return id >= 3 && id <= 5;
        //return id >= (int)CornerId.Bottom && id <= (int)CornerId.RightBottom;
    }

    bool _IsInternalEventExist(InternalEvent ievent)
    {
        foreach (InternalEvent e in _internalEvents)
        {
            if (e.eventType == ievent.eventType && e.go == ievent.go && e.iParam ==
ievent.iParam)
            {
                return true;
            }
        }
        return false;
    }
```

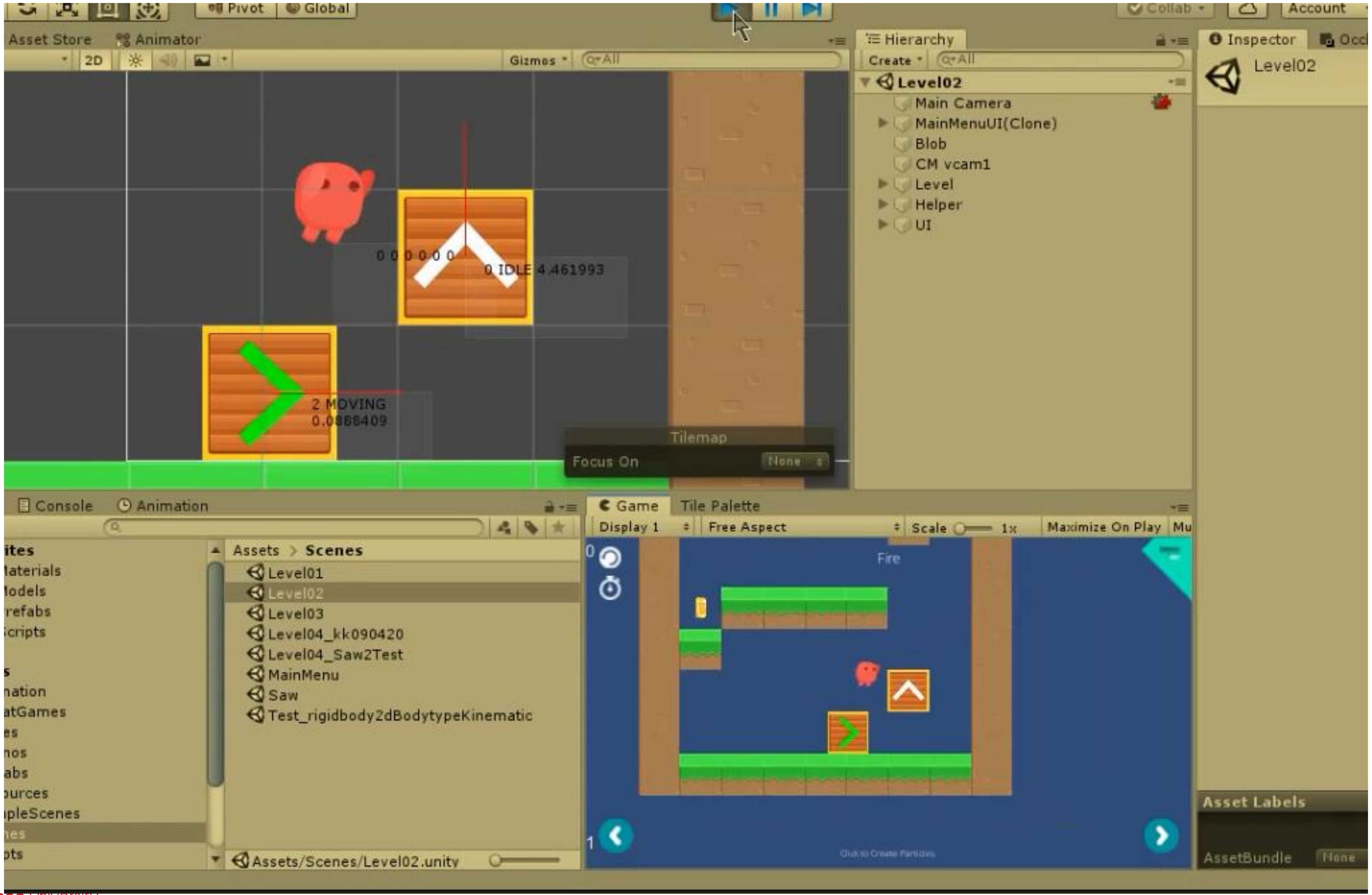
```
bool _AddInternalEvent(InternalEvent ievent, bool bAllowDuplicate=false)
   {
       if (bAllowDuplicate == false)
       {
           if (_IsInternalEventExist(ievent))
               return false;
       }

       _internalEvents.Enqueue(ievent);
       return true;
   }

   void _ProcessInternalEvent()
   {
       _groundVelocity = Vector2.zero;
       foreach (InternalEvent e in _internalEvents)
       {
           if (e.eventType == InternalEventType.DestroyCharacter)
           {
               _DestroyCharacter();
           }
           else if (e.eventType == InternalEventType.CornerCollision)
           {
               if (_IsBottomCornerId(e.iParam))
               {
```

```
            else if (e.eventType == InternalEventType.CornerCollision)
            {
                if (_IsBottomCornerId(e.iParam))
                {
                    _groundVelocity = e.go.GetVelocity();
                }

                if (e.go.CompareTag("Box"))
                {
                    BoxBehaviour boxBehavior = e.go.GetComponent<BoxBehaviour>();
                    if (boxBehavior)
                    {
                        boxBehavior.DoExternalCollision(gameObject);
                    }
                }
                else if (e.go.CompareTag("Saw"))
                {
                    _DestroyCharacter();
                }
            }//if.. else if..
        }
        _internalEvents.Clear();
    }
```



Assets > Scenes
- Level01
- Level02
- Level03
- Level04_kk090420
- Level04_Saw2Test
- MainMenu
- Saw
- Test_rigidbody2dBodytypeKinematic

Assets/Scenes/Level02.unity

Regression Test

- ✓ **Regression testing** (rarely *non-regression testing*[1]) is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.[2]
- ✓ If not, that would be called a *regression*.
- ✓ Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components.
- ✓ As regression test suites tend to grow with each found defect, test automation is frequently involved.

MY **BRIGHT** FUTURE

DSU Dongseo University 동서대학교