SMPL-X in Unity

120220121/신종현

0. 서문

: 이번 과제에서는 먼저 SMPLX Script 를 Refactoring 하여 가독성 및 유지보수성을 올리고 전반적인 동작 원리를 살펴본다. 그 다음으로는 이러한 SMPL-X 가 적용된 Character 개수 및 Active 한 BlendShape (=BlendShapeWeight 가 0.0f 이 아닌) 개수에 따른 FPS (Frame Per Second)의 변화를 살펴보도록 한다

1) SMPLX Script Refactoring 및 구조 파악

: 여기서 Script 라고 표현하는 것은 모두 class SMPLX 를 의미한다

: 여기서 말하는 GUI 는, 실제 프로그램을 배포했을 때의 사용자가 사용 가능한 GUI 를 의미 하는게 아니고, 개발자가 Unity Editor 에서 사용 할 수 있는 Character 에 달려있는 Component 들을 의미한다

- GUI 구성
- 주요 구조
- Script 함수들
- Character GUI Editor (class SMPLX_CustomEditor) 함수들
- class JointManage 함수들
- class BodyPoseManager 함수들
- class HandPoseManager 함수들

2) SMPL-X FPS Test

: 여기서 Script 라고 표현하는 것은 모두 class SMPLXTestor 를 의미한다

- GUI 구성
- 주요 구조
- Script 함수들
- 실험 결과

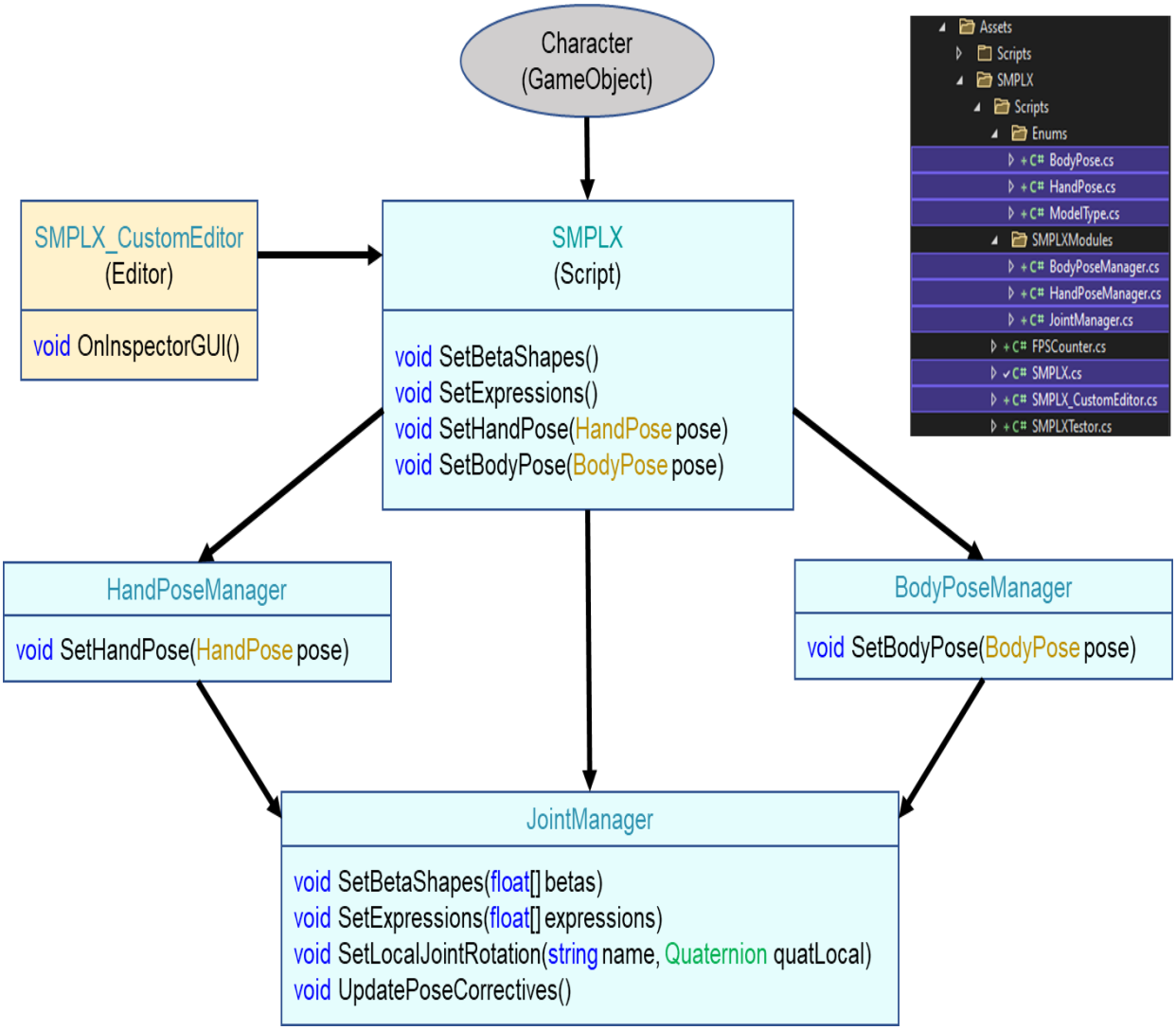
<SMPLX Script Refactoring 및 구조 파악>

1. GUI 구성

: Shape 에서 BodyShape 를 조절하는 10 가지 Beta 를 원하는 대로 Slider 를 이용해서 조절하여 Set Button 을 누르면 BodyShape 이 변하게 된다. Random Button 을 누르면 10 가지 Beta 가 Random 하게 -5 ~ +5 사이의 값을 가지게 된다. Reset Button 을 누르면 모든 Beta 값이 0 으로 초기화 된다. Expression 의 Set/Random/Reset Button 도 이와 비슷하게 동작 한다. T-Pose Button 혹은 A-Pose Button 를 누르면 그에 맞는 Body Pose 를 취하게 되고, Flat Button 혹은 Relaxed Button 를 누르면 그에 맞는 Hand Pose 를 취하게 된다

2. 주요 구조

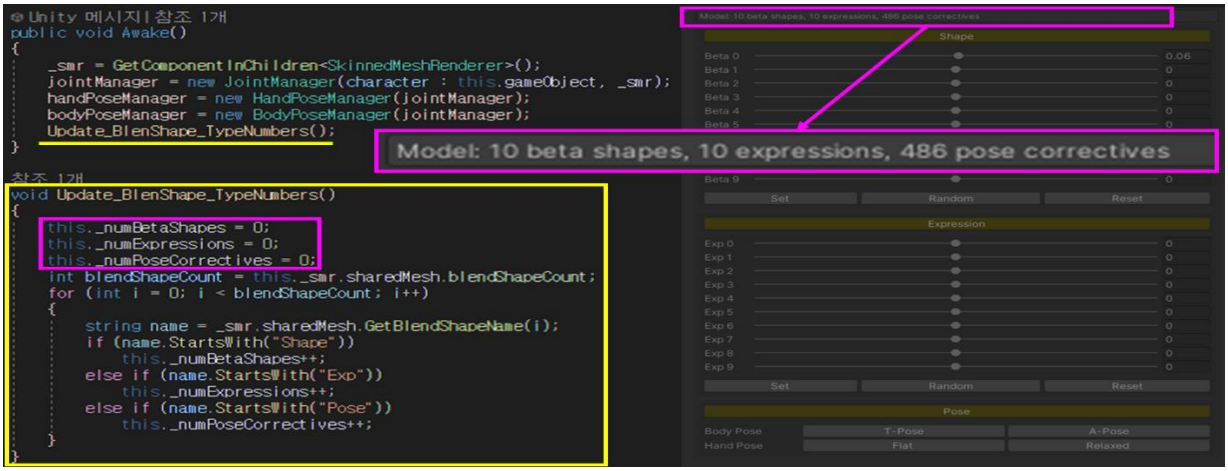
: Character(GameObject)가 참조하는 SMPLX Script 는 일종의 Façade Design Pattern 처럼 동작하여 Character 를 Control 하기 위한 함수들을 전달하고. SMPLX_CustomEditor 에서 이러한 SMPLX 를 참조하여 실질적으로 GUI 를 그려주고, 그려진 GUI Component 들을 통해 Character 를 Control 한다. SMPLX 의 GUI 상에서 Shape 와 Expression 쪽을 Control 하기 위해 쓰이는 함수 SetBetaShape() 및 SetExpression()의 실질적인 구현은 JointManager 에서 진행되었고, SMPLX 의 SetHandPose(..)의 실질적인 구현은 HandPoseManager 에서 진행되었으며, SMPLX 의 SetBodyPose(..)의 실질적인 구현은 BodyPoseManager 에서 진행되었다

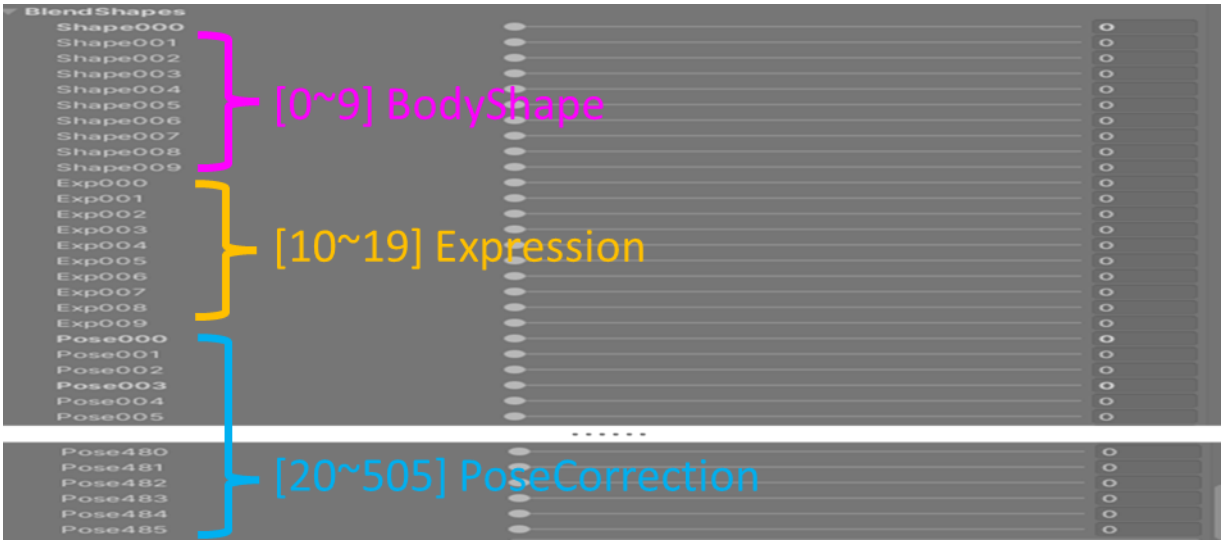


3. Script 함수들

- Awake()함수

: Awake() 에서는 Character 의 SkinnedMeshRender 를 가져오고, JointManager, HandPoseManager 및 BodyPoseManager 객체들을 생성하고 마지막으로 Update_BlendShape_TypeNumbers()함수를 호출하여 SkinnedMeshRender 의 BlendShape 중에서 BodyShape(=beta), Expression 및 Pose 가 몇 개 있는지 확인하는 public 멤버 변수들 (int _numXXX)을 업데이트 한다. 이 멤버 변수들은 추후 SMPLX_CustomEditor 에서 BlendShape 이 종류별로 몇 개씩 있는지 GUI 상에 보여주는데 사용된다. 실제로 Unity Editor 상에서 확인하면 Character 의 SkinnedMeshRender 의 BlendShape 는 index 기준 0~9 는 BodyShape, 10~19 는 Expression, 나머지 20~505 는 PoseCorrection 을 Control 하는 BlendShape 임을 알 수 있다





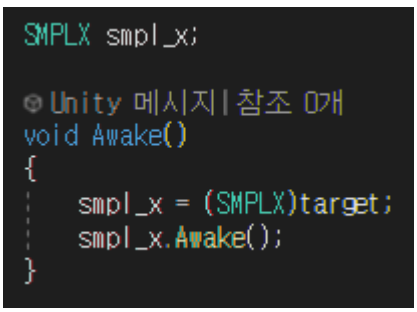
- Update() 및 나머지 함수들

: Update()에서는 JointManager 객체의 UpdatePoseCorrectives()를 호출하여 GUI 통해서 변경된 SMPLX 의 Parameter 들의 변경사항을 Unity 의 SkinnedMeshRenderer 의 동작에 호환되도록 변경하여, 매 Frame 마다 이러한 SMPLX 의 변경 사항들이 SkinnedMeshRenderer 에 반영되도록 한다. SMPLX_CustomEditor 에게 있어 SMPLX 는 일종의 Façade Pattern 처럼 동작기에 public 함수들의 실질적인 구현은 다른 Module 들이 진행하고, SMPLX 는 이러한 다른 Module 의 함수들을 delegate 하는 역할을 수행한다

4. Character GUI Editor (class SMPLX_CustomEditor) 함수들

- Awake()함수

: 여기서는 SMPLX 객체를 받아와서, 위에서 설명한 SMPLX 객체의 Awake()함수를 호출하여 Character 를 Control 하기 위해 필요한 객체들을 생성하고 blendshapeType 개수 관련 매개변수(int _numXXX)들을 업데이트 한다

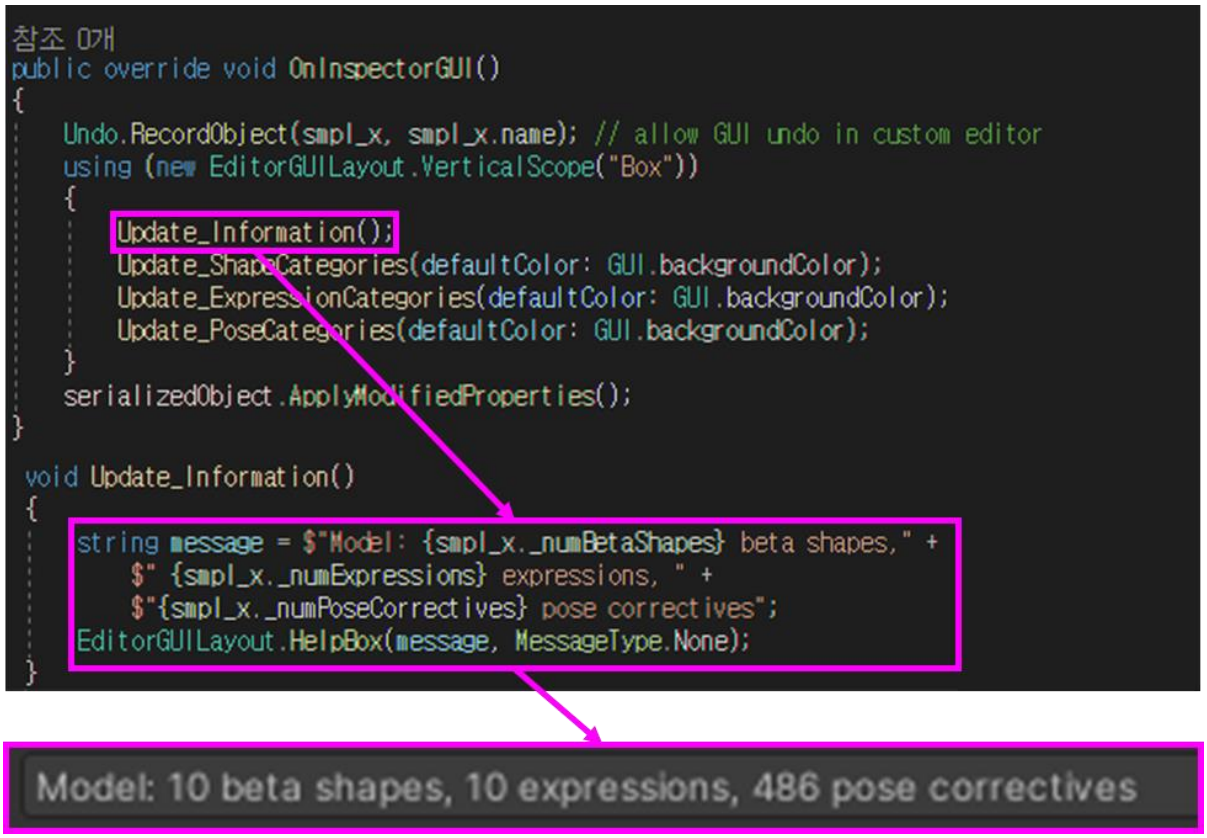


- **OnInspectorGUI()함수**

: 이 함수는 Character 에 달려있는 Script 에 GUI Component 들을 만들어 주고, 사용자가 생성된 GUI Component 들을 통해 Character 를 Control 하고자 할 때, Script(=SMPLX)의 함수들을 호출하여 Character 가 사용자의 의도대로 동작 하도록 한다

- **Update_Information()함수**

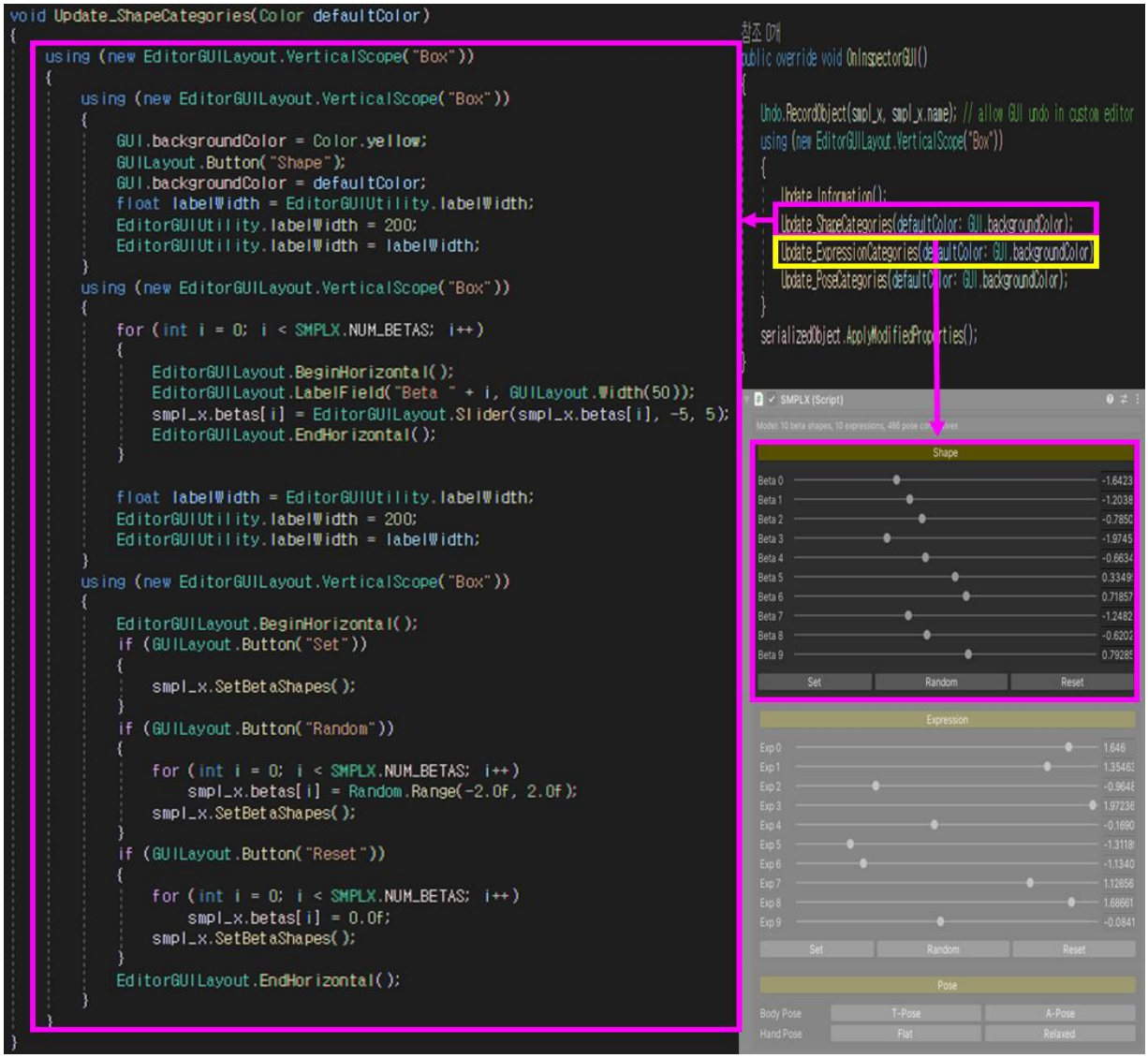
: 이 함수에서는 GUI 에 HelpBox 를 생성하여, SMPLX 의 blendshapeType 개수 관련 매개변수(int _numXXX)값을 보여 주도록 한다



- Update_ShapeCategories() 및 Update_ExpressionCategories()함수

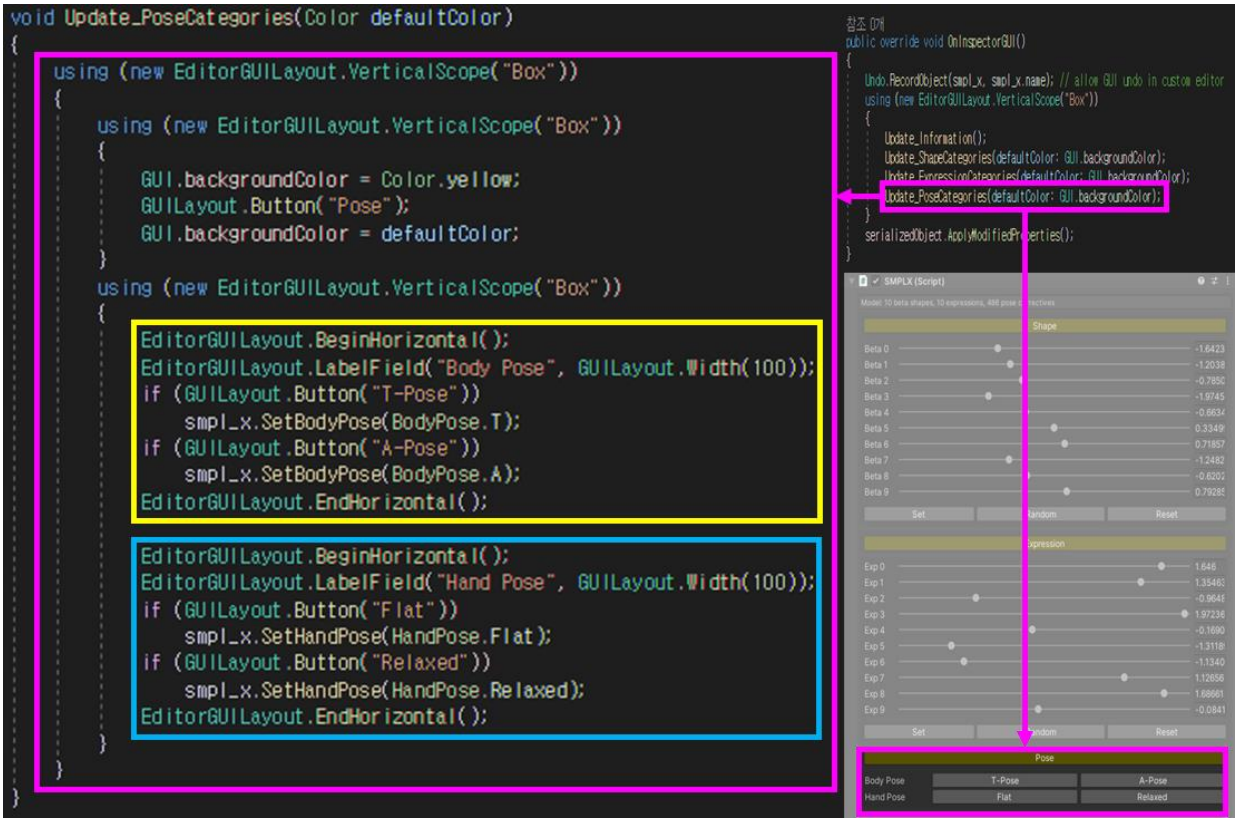
: Update_ShapeCategories()는 다음과 같이 GUI 의 Shape 부분을 구성하는 함수이다. 여기서 보면 Set/Random/Reset Button 이 Click 되면 SMPLX 의 betas 를 변경하거나 Slider 를 통해서 변경이 된 betas 를 Update 해주기 위한 SMPLX 의 SetBetaShapes()함수를 호출한다

: Update_ExpressionCategories()을 설명하는 Image 는 공간상 따로 보여주지는 않았지만, Update_ExpressionCategories()도 위에서 설명한 Update_ShapeCategories()과 비슷한 역할을 수행하며, 여기서는 BodyShape 대신 Expression 을 변경한다. 즉, Update_ExpressionCategories()는 SMPLX 의 betas 대신 expressions 을 변경하며, SMPLX 의 SetBetaShapes()대신 SetExpressions()함수를 호출한다



- Update_PoseCategories()함수

: T-Pose Button 이 Click 되면 SetBodyPose(BodyPose.T)를 호출하고 A-Pose Button 이 Click 되면 SetBodyPose(BodyPose.A)를 호출하도록 하여 Body Pose 를 Control 할 수 있다. 또한, Flat Button 이 Click 되면 SetHandPose(HandPose.Flat)을 호출하고 Relaxed Button 이 Click 되면 SetHandPose(HandPose.Relaxed)를 호출하여 Hand Pose 를 Control 할 수 있다



5. class JointManager 함수들

- SetBetaShapes(..), SetExpressions(..) 및 UpdatePoseCorrectives() 함수

: SetBetaShapes(..)는 GUI 를 통해 설정된 betas 를 Bodyshape 을 담당하는 Blendshape 의 Weight 에 적용한다 (index : 0 ~ 9). 또한 Bodyshape 이 변경되면 Character 가 마치 공중에 떠있는 것처럼 보일 수 있기 때문에, Character 의 y Position 을 변경하여 Bodyshape 가 변경되어도 Character 는 항상 plane 위에 서 있을 수 있도록 한다. (plane 의 y position 은 0 으로 해 두었음)

: SetExpressions(..)은 GUI 를 통해 설정된 expressions 를 Expression 을 담당하는 Blendshape 의 Weight 에 적용한다 (index : 10 ~ 19)

: UpdatePoseCorrectives()는 SMPLX 의 Joint 들의 Rotation 값이 Unity 의 SkinnedMeshRender 에 호환되도록 변경하여 Pose Correction 을 담당하는 Blendshape 의 Weight 에 적용되도록 한다 (index: 20 ~ 505)

```
참조 1개
public void SetBetaShapes(float[] betas)
{
    for (int i = 0; i < SMPLX.NUM_BETAS; i++)
        _smr.SetBlendShapeWeight(i, betas[i] * 100); // blend shape weights are specified in percentage

    Mesh _bakedMesh = new Mesh();
    _smr.BakeMesh(_bakedMesh);
    Vector3[] vertices = _bakedMesh.vertices;
    float yMin = float.MaxValue;
    for (int i = 0; i < vertices.Length; i++)
        if (vertices[i].y < yMin)
            yMin = vertices[i].y;
    Vector3 localPosition = character.transform.localPosition;
    character.transform.localPosition = new Vector3(localPosition.x, -yMin, localPosition.z);
}

참조 1개
public void SetExpressions(float[] expressions)
{
    for (int i = 0; i < SMPLX.NUM_EXPRESSIONS; i++)
        _smr.SetBlendShapeWeight(i + SMPLX.NUM_BETAS, expressions[i] * 100); // blend shape weights are specified in percentage
}

참조 2개
public void UpdatePoseCorrectives()
{
    for (int i = 1; i < _bodyJointNames.Length; i++)
    {
        string name = _bodyJointNames[i];
        Quaternion quat = _transformFromName[name].localRotation;
        Quaternion quatSMPLX = new Quaternion(-quat.x, quat.y, quat.z, -quat.w);
        Matrix4x4 m = Matrix4x4.Rotate(quatSMPLX);
        m[0, 0] = m[0, 0] - 1.0f;
        m[1, 1] = m[1, 1] - 1.0f;
        m[2, 2] = m[2, 2] - 1.0f;

        int poseStartIndex = SMPLX.NUM_BETAS + SMPLX.NUM_EXPRESSIONS + (i - 1) * 9;

        _smr.SetBlendShapeWeight(poseStartIndex + 0, 100.0f * m[0, 0]);
        _smr.SetBlendShapeWeight(poseStartIndex + 1, 100.0f * m[0, 1]);
        _smr.SetBlendShapeWeight(poseStartIndex + 2, 100.0f * m[0, 2]);

        _smr.SetBlendShapeWeight(poseStartIndex + 3, 100.0f * m[1, 0]);
        _smr.SetBlendShapeWeight(poseStartIndex + 4, 100.0f * m[1, 1]);
        _smr.SetBlendShapeWeight(poseStartIndex + 5, 100.0f * m[1, 2]);

        _smr.SetBlendShapeWeight(poseStartIndex + 6, 100.0f * m[2, 0]);
        _smr.SetBlendShapeWeight(poseStartIndex + 7, 100.0f * m[2, 1]);
        _smr.SetBlendShapeWeight(poseStartIndex + 8, 100.0f * m[2, 2]);
    }
}
```

character의 y position 조정
(항상 땅 위에 서 있도록)

6. class BodyPoseManager 함수들

- SetBodyPose(..) 함수

: 입력으로 전달된 BodyPose 가 BodyPose.T 일 경우에는 ResetBodyPose()를 진행하여 Body 의 모든 Joint 들을 초기화 시키고, 입력으로 전달된 BodyPose 가 BodyPose.A 일 경우에는 초기화된 Joint 에서 Collar 와 Shoulder Joint 들의 Rotation 을 변경하여 Character 의 팔이 좌우로 어느정도 내려와 있는 A 자 동작을 취하도록 한다

```
참조 1개
public void SetBodyPose(BodyPose pose)
{
    ResetBodyPose();
    if (pose == BodyPose.A)
    {
        jointManager.SetLocalJointRotation("left_collar", Quaternion.Euler(0.0f, 0.0f, 10.0f));
        jointManager.SetLocalJointRotation("left_shoulder", Quaternion.Euler(0.0f, 0.0f, 35.0f));
        jointManager.SetLocalJointRotation("right_collar", Quaternion.Euler(0.0f, 0.0f, -10.0f));
        jointManager.SetLocalJointRotation("right_shoulder", Quaternion.Euler(0.0f, 0.0f, -35.0f));
    }
    jointManager.UpdatePoseCorrectives();
}

참조 1개
void ResetBodyPose()
{
    foreach (string name in jointManager._bodyJointNames)
    {
        Transform joint = jointManager._transformFromName[name];
        joint.localRotation = Quaternion.identity;
    }
}
```

7. class HandPoseManager 함수들

- SetHandPose(..)함수

: 입력으로 전달된 HandPose 에따라, HandPose 관련 미리 설정된 Hand 관련 Joint 들에 대한 Left/Right Hand Joint Parameter 값들을 불러오고, JointManager 객체의 SetLocalJointRotaion(..)함수를 호출하여 Hand 관련 Joint 들을 이러한 값으로 변경하여 Character 가 설정된 HandPose 로 손 동작을 변경하도록 한다

```
참조 1개
public void SetHandPose(HandPose pose)
{
    float[] left = (pose == HandPose.Flat) ? _handFlatLeft : _handRelaxedLeft;
    float[] right = (pose == HandPose.Flat) ? _handFlatRight : _handRelaxedRight;
    for (int i = 0; i < HandJoingLength; i++)
    {
        Quaternion left_quat = QuatFromRodrigues(rodX: left[i * 3 + 0], rodY: left[i * 3 + 1], rodZ: left[i * 3 + 2]);
        JointManager.SetLocalJointRotation(name: _handLeftJointNames[i], left_quat);

        Quaternion right_quat = QuatFromRodrigues(rodX : right[i * 3 + 0], rodY: right[i * 3 + 1], rodZ: right[i * 3 + 2]);
        JointManager.SetLocalJointRotation(name: _handRightJointNames[i], right_quat);
    }
}
```

Left Hand

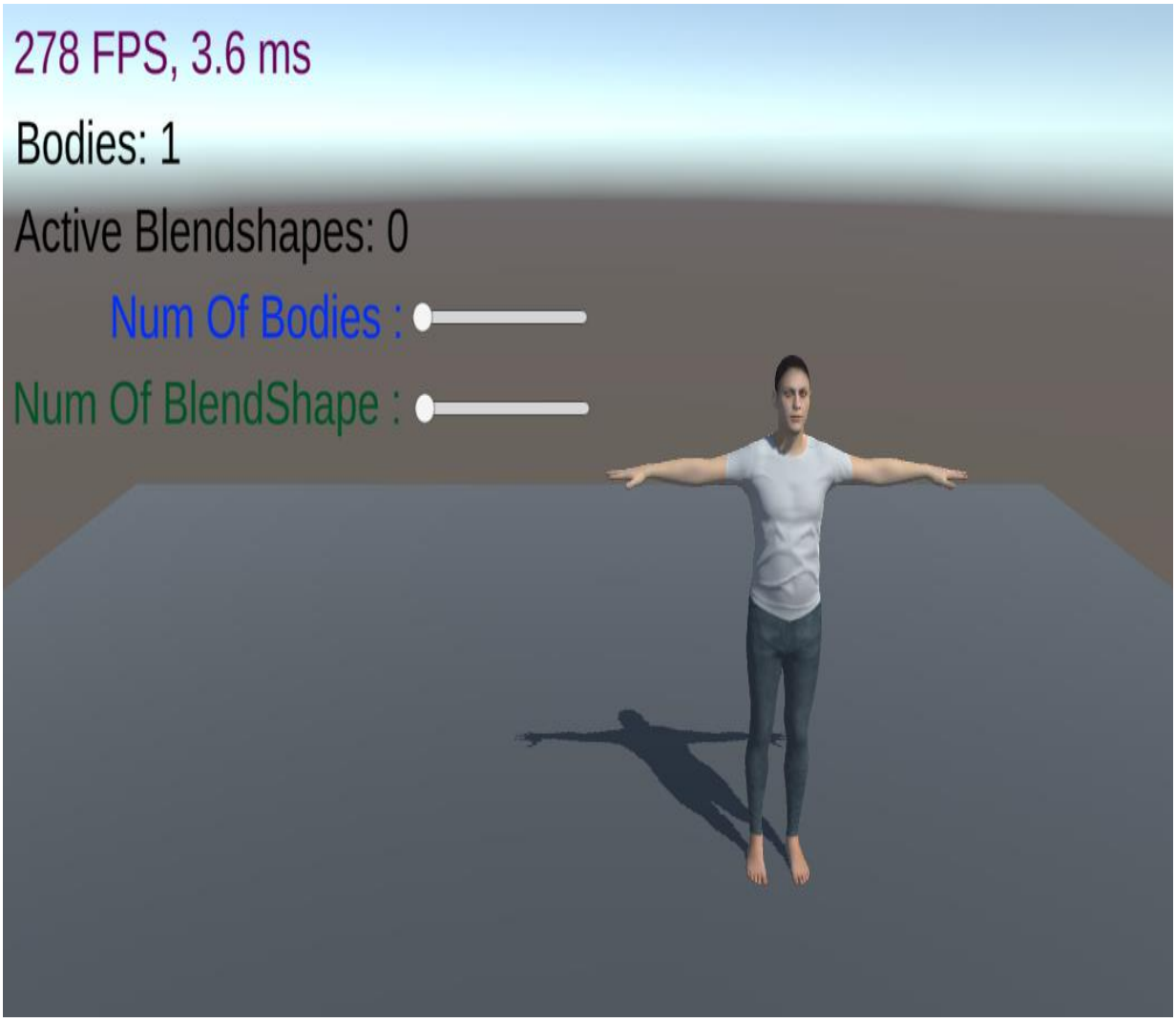
Right Hand

<SMPL-X FPS Test >

1. GUI 구성

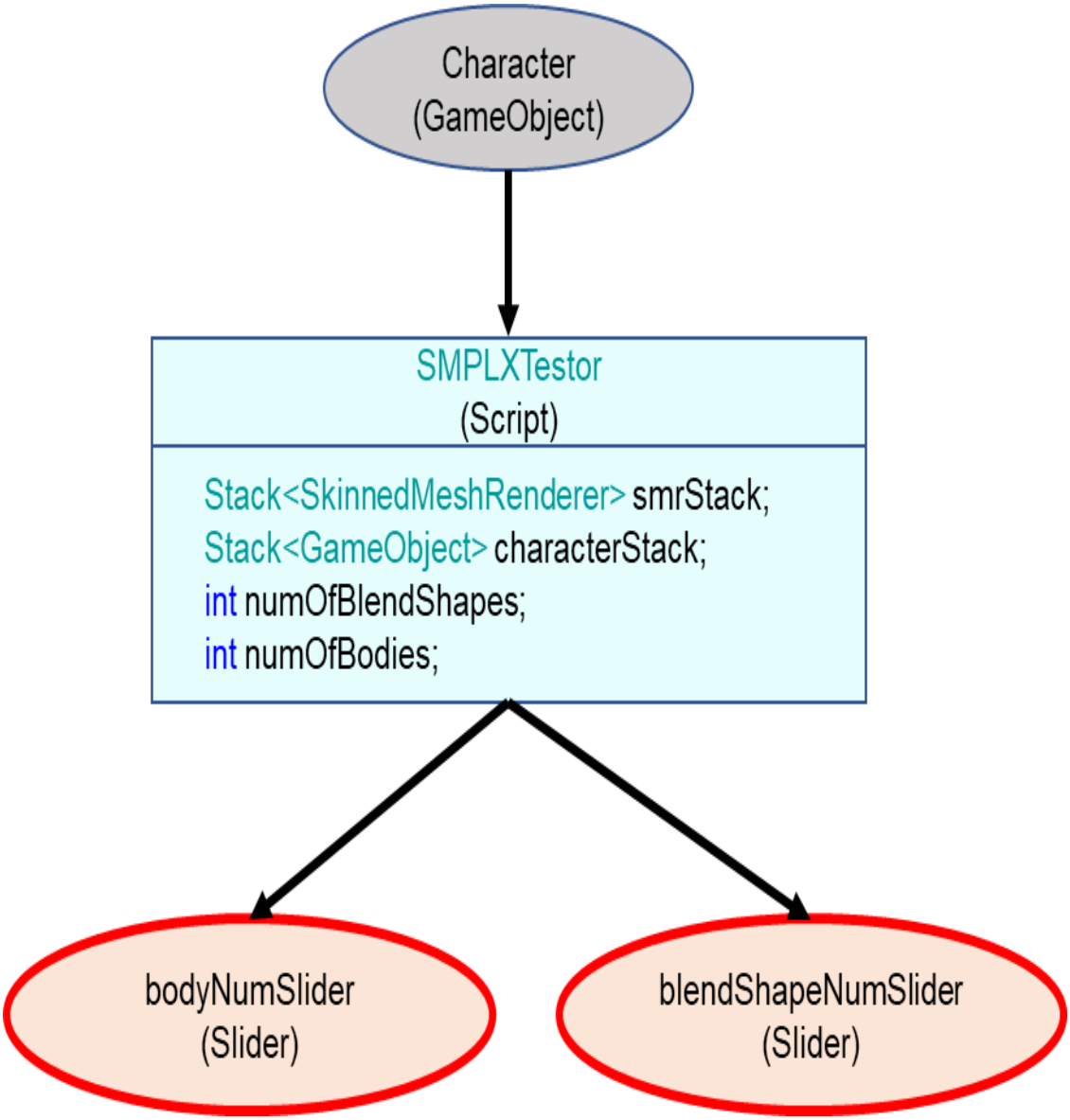
: 하기 이미지와 같이 2 가지 Slider 를 이용하여 GUI 를 통해 User 가 응용 프로그램을 조작할 수 있도록 하였다

- Num Of Bodies Slider: 1~10 까지 설정 가능한 정수로 Position 값을 제외한 나머지 상태가 모두 동일한 Character 를 생성할 수 있다
- Num Of BlendShape Slider: 0~500 까지 설정 가능한 정수로, 설정된 정수 개수만큼 BlendShape 의 Weight 가 0.0f 이 아닌 시간에 따른 Sin 의 값 50~150 사이의 Wave 형의 Value 를 가지도록 하였다 (제공되는 Character 의 Max BlendShape 가 506 인것을 고려하여 Max 값을 500 으로 설정)



2. 주요 구조

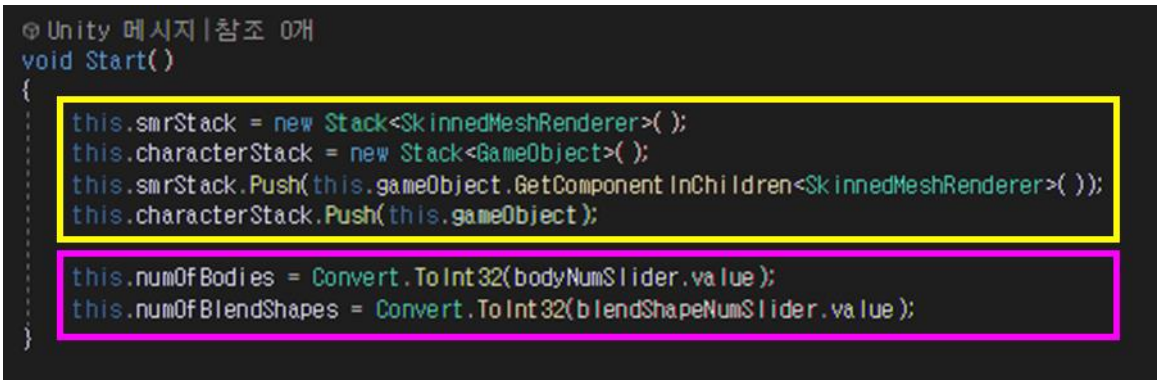
: 하기와 같은 구조로 SMPLXTestor 가 참조하는 Slider들이 int numOfBlendShapes와 int numOfBodies를 업데이트하여 이 변수들의 값이 변함에 따라 FPS가 어떻게 변하는지 확인할 수 있도록 구현하였다. 또한 GameObject와 SkinnedMeshRender를 저장할 수 있는 Stack들을 만들어 int numOfBodies 만큼 Character 와 해당 Character가 가지는 SkinnedMeshRender를 Stack에 저장 혹은 삭제되도록 하였다. FILO 특성을 가지는 Stack을 이용하는 이유는 int numOfBodies 의 값이 작아 짐에 따라 가장 마지막에 추가된 객체들이 먼저 삭제되도록 하기 위해서이다



3. Script 함수들

- Start()함수

: 먼저 Start() 함수를 살펴보면 다음과 같이 GameObject 를 저장하는 Stack(=characterStack)과 SkinnedMeshRenderer 를 저장하는 Stack(=smrStack) 객체들을 생성하고, SMPLXTestor 를 참조하는 현재 GameObject(=this.gameObject)를 characterStack 에 추가하고, 이러한 GameObject 의 SkinnedMeshRender 를 smrStack 에 추가한다



- Update()함수

: 매 Frame 마다 호출되는 Update()함수는 다음과 같이 4 가지 함수를 순차적으로 호출하여 GUI 상에서 변경되는 bodyNumSlider 와 blendShapeNumSlider 에 따라 Character 개수와 BlendShapeWeight 를 변경해주고 이를 GUI 상에서 보여준다

```
void Update()
{
    InitializeBlendShapeWeights();
    UpdateBodies();
    SetBlendShapeWeights();
    UpdateStatus();
}
```

- Update()에서 첫 번째로 호출하는 InitializeBlendShapeWeights()함수

: InitializeBlendShapeWeights()함수는 blendShapeNumSlider 의 값이 변경되었을 때, 기존의 506 개의 모든 BlendShapeWeight 를 0.0f 으로 초기화 시켜준다

```
void Update()
{
    InitializeBlendShapeWeights();
    UpdateBodies();
    SetBlendShapeWeights();
    UpdateStatus();
}
void InitializeBlendShapeWeights()
{
    if (this.numOfBlendShapes != Convert.ToInt32(blendShapeNumSlider.value))
        foreach (SkinnedMeshRenderer smr in this.smrStack)
            for (int i = 0; i < smr.sharedMesh.blendShapeCount; i++)
                smr.SetBlendShapeWeight(i, 0.0f);
}
```

- Update()에서 두 번째로 호출하는 UpdateBodies()함수

: UpdateBodies()는 bodyNumSlider 의 값이 변경되었을 때 characterStack 과 smrStack 을 이용하여 bodyNumSlider 값 만큼 GameObject(=character) 및 해당 GameObject 의 SkinnedMeshRenderer 를 추가 혹은 삭제를 한다. 여기서 Get_NewCharacter()함수는 새로운 GameObject(=character) 객체를 생성하는 함수이다

```
void Update()
{
    InitializeBlendShapeWeights();
    UpdateBodies();
    SetBlendShapeWeights();
    UpdateStatus();
}
```

```
void UpdateBodies()
{
    int curNumOfBodies = Convert.ToInt32(bodyNumSlider.value);
    while (numOfBodies < curNumOfBodies)
    {
        GameObject character = Get_NewChracter();
        characterStack.Push(character);
        smrStack.Push(character.GetComponentInChildren<SkinnedMeshRenderer>());
        numOfBodies++;
    }
    while (numOfBodies > curNumOfBodies)
    {
        smrStack.Pop();
        Destroy(characterStack.Pop());
        numOfBodies--;
    }
}
```

```
GameObject Get_NewChracter()
{
    float x = 2.5f * Mathf.Sin(2 * Mathf.PI * (this.numOfBodies) / 20.0f);
    float y = this.gameObject.transform.position.y;
    float z = this.numOfBodies * 0.25f;
    Vector3 pos = new Vector3(x, y, z);
    return Instantiate(this.gameObject, pos, Quaternion.Euler(0.0f, 180.0f, 0.0f));
}
```

- Update()에서 세 번째로 호출하는 SetBlendShapeWeights()함수

: SetBlendShapeWeight()는 smrStack 에 저장되어 있는 SkinnedMeshRenderer 의 numOfBlendShapes 만큼의 BlendShape 에 50~150 사이의 시간에 따른 Sin 파형의 값을 할당한다

```
void Update()
{
    InitializeBlendShapeWeights();
    UpdateBodies();
    SetBlendShapeWeights();
    UpdateStatus();
}
void SetBlendShapeWeights()
{
    float value = Mathf.Sin(2 * Mathf.PI * Time.time) * 50 + 100.0f;
    foreach (SkinnedMeshRenderer smr in smrStack)
        for (int i = 0; i < numOfBlendShapes; i++)
            smr.SetBlendShapeWeight(i, value);
}
```

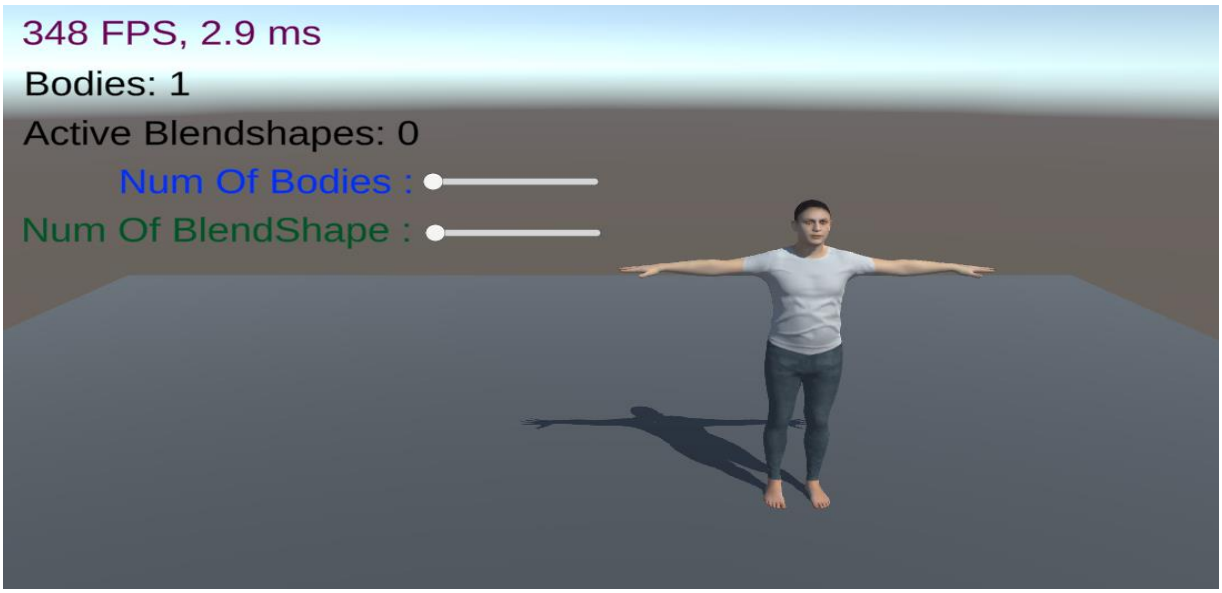
- Update()에서 네 번째로 호출하는 UpdateStatus()함수

: UpdateStatus()함수는 bodyNumSlider 와 blendShapeNumSlider 에 따라 int numOfBodies 와 numOfBlendShapes 를 업데이트 해주고 이러한 값들을 GUI 상에서 볼 수 있도록 업데이트 해준다

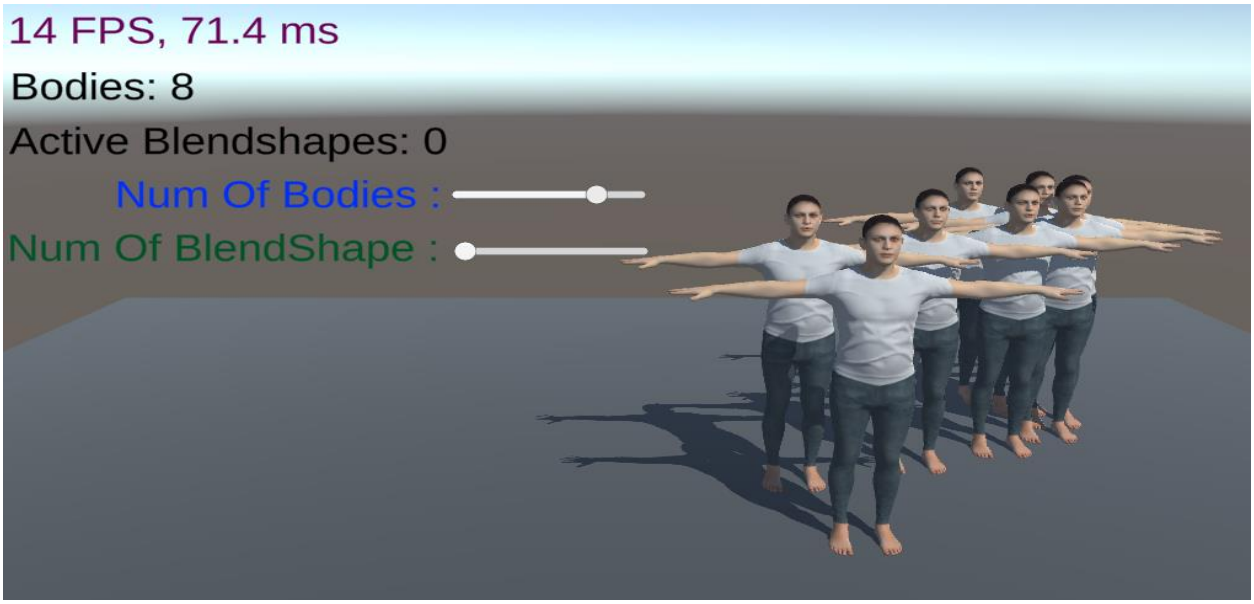
```
void Update()
{
    InitializeBlendShapeWeights();
    UpdateBodies();
    SetBlendShapeWeights();
    UpdateStatus();
}
void UpdateStatus()
{
    this.numOfBodies = Convert.ToInt32(bodyNumSlider.value);
    this.numOfBlendShapes = Convert.ToInt32(blendShapeNumSlider.value);
    this.statusText.text = $"Active Blendshapes: {GetActiveBlendshapes()}";
    this.bodiesText.text = $"Bodies: {this.numOfBodies}";
}
```

4. 실험 결과

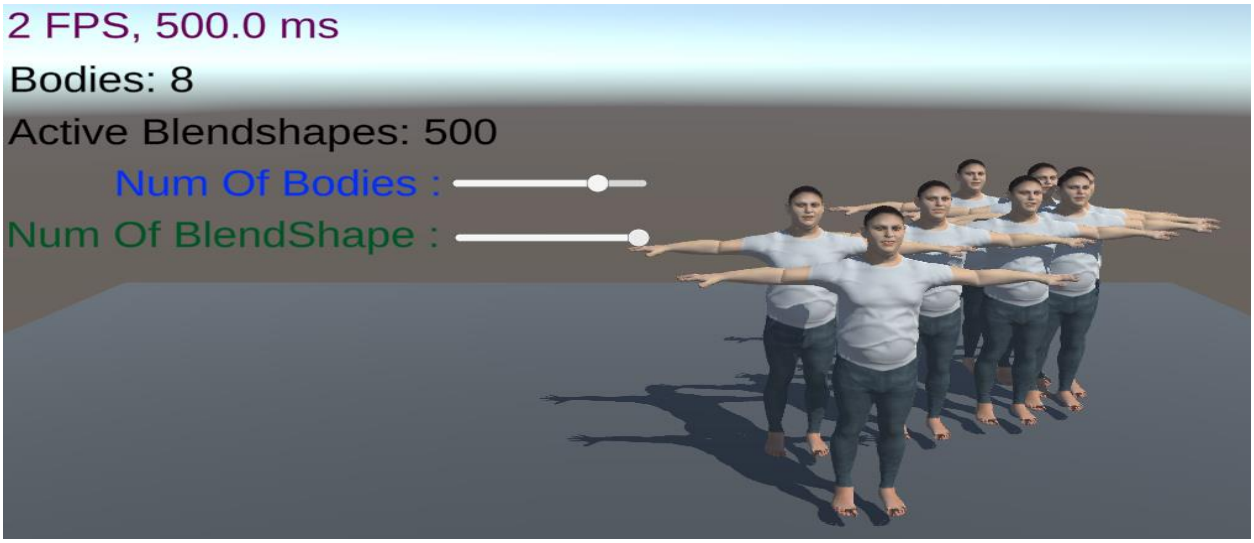
: 다음과 같이 Character 가 1 이고 Active BlendShape 이 0 일 때는 FPS 가 348 까지 올라가는 것을 확인할 수 있다 (=2.9ms)



다음과 같이 Character 가 8 개까지 늘어나고 Active Blendshape 를 여전히 0 으로 유지할 경우 FPS 가 14 까지 줄어드는 것을 확인할 수 있다(=71.4ms)



여기서 Num Of BlendShape 를 500 으로 설정하면 다음과 같이 FPS 가 2 로 더욱 줄어들게 되고 (=500.0ms) 육안으로 보았을 때 Character 의 변화가 끊기듯이 업데이트 되는 것을 확인할 수 있다. Update()에서 50~150 사이의 값을 가지는 Sin 의 파형을 Discrete 하게 Sampling 해서 BlendShapeWeight 를 설정해 주는데, 이러한 끊임이 있다는 것은 Samling 의 빈도가 낮아졌다는 것을 의미한다. 즉, Update()함수가 0.5 초에 한 번씩 호출되는 수준으로 FPS 가 떨어졌다는 것을 육안으로 느낄 수 있다



Num Of BlendShape 를 10 으로 설정하면 FPS 가 12 가 되어 (=83.3ms), 이전 대비 훨씬 Smooth 하게 Character 의 변화를 관찰할 수 있다. 즉, Update()에서 Sin 파형을 Sampling 하는 빈도가 짧아진 것을 육안으로 느낄 수 있다