Technical Document for Thai Dance Animation using C#

Contents

[Methodology. 3](#_Toc511209815)

[Layout. 3](#_Toc511209816)

[Instruction Using Xml and Unity 3](#_Toc511209817)

[Class Script. 3](#_Toc511209818)

[Body class 3](#_Toc511209819)

[Measure class 4](#_Toc511209820)

[XmlContainerScript 5](#_Toc511209821)

[parseFile() 5](#_Toc511209822)

[BodyControllerScript 5](#_Toc511209823)

[ProcessAndCreateJoints() 5](#_Toc511209824)

[GetandProcessXMLTask() 5](#_Toc511209825)

[Duration of a Beat 5](#_Toc511209826)

[Support 6](#_Toc511209827)

[AssignTaskToJoints() 6](#_Toc511209828)

[JointControllerScript 6](#_Toc511209829)

[Lerp 6](#_Toc511209830)

[Number of Frame 7](#_Toc511209831)

[Update 7](#_Toc511209832)

# Methodology.

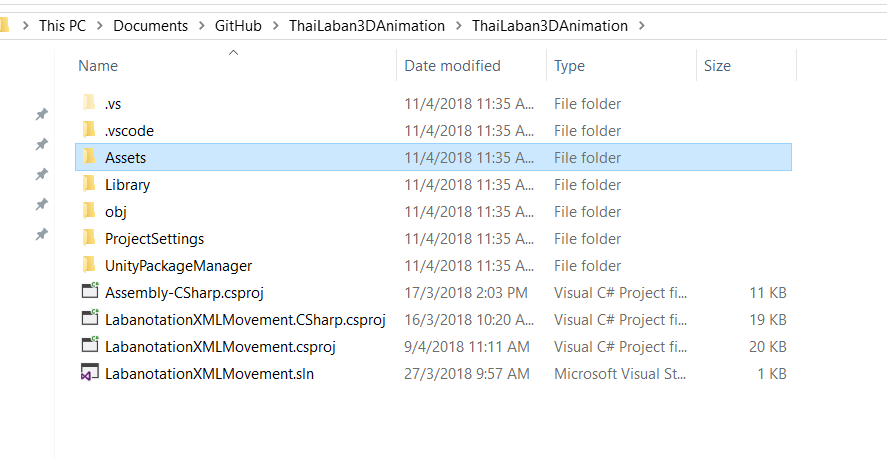
The Laban Dance Notation will be recorded into an Xml file, that will be translated into C# Syntax. A script called XmlContainer, will be used to translate the Xml file and stored in a List, in C#. The second script called the BodyController, will then retrieve the List from XmlContainer and sort the timing of all the dance move from beats to seconds. After sorting, the BodyController will parse the dance move to the script of each specific body part. Every single body part, from the head to the toes, arm to the fingers, contain a script called the JointController. This script is used to animate the model.

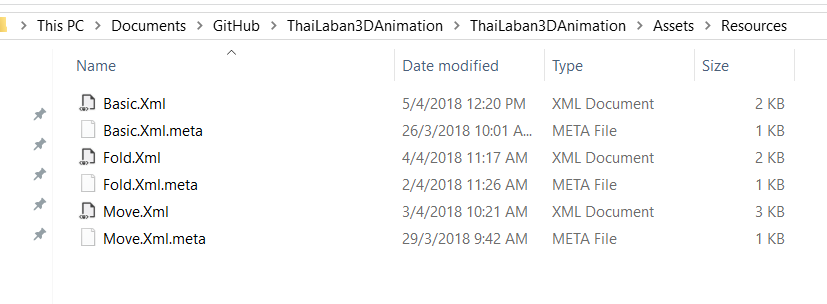
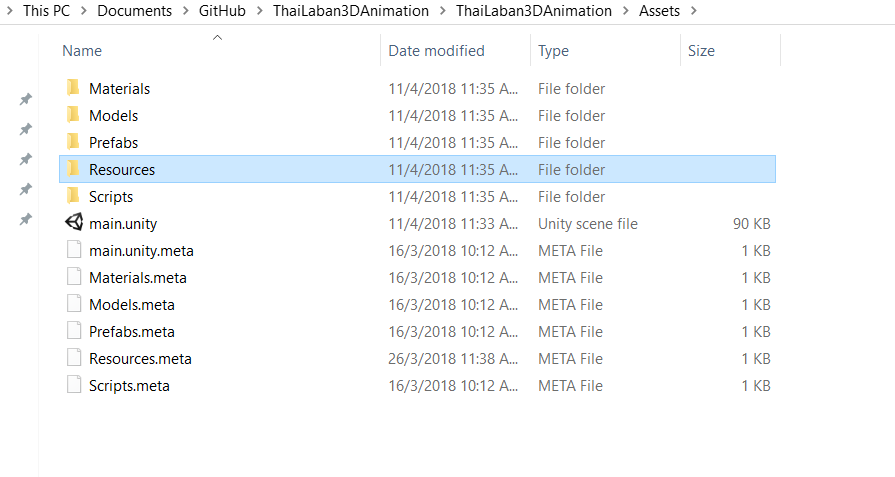
## Layout.

In this project, there are 5 folders and 1 scene. The scene is called main where the models are placed and programmed. The “Script” folders contain both scripts and class scripts. The “Resources” folder is where all the Xml files are stored.

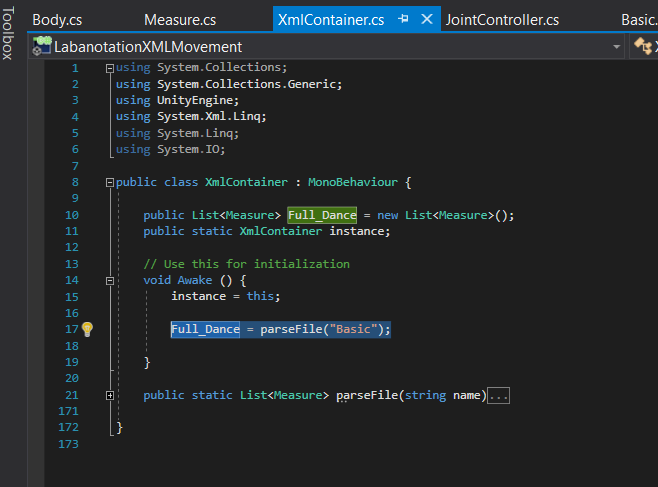
# Instruction Using Xml and Unity

Drag all Thai Laban Xml files into the Unity Project Resources Folder.





Change the name of the string in the XmlContainerScript to the name of the Xml file. \*Case Sensitive



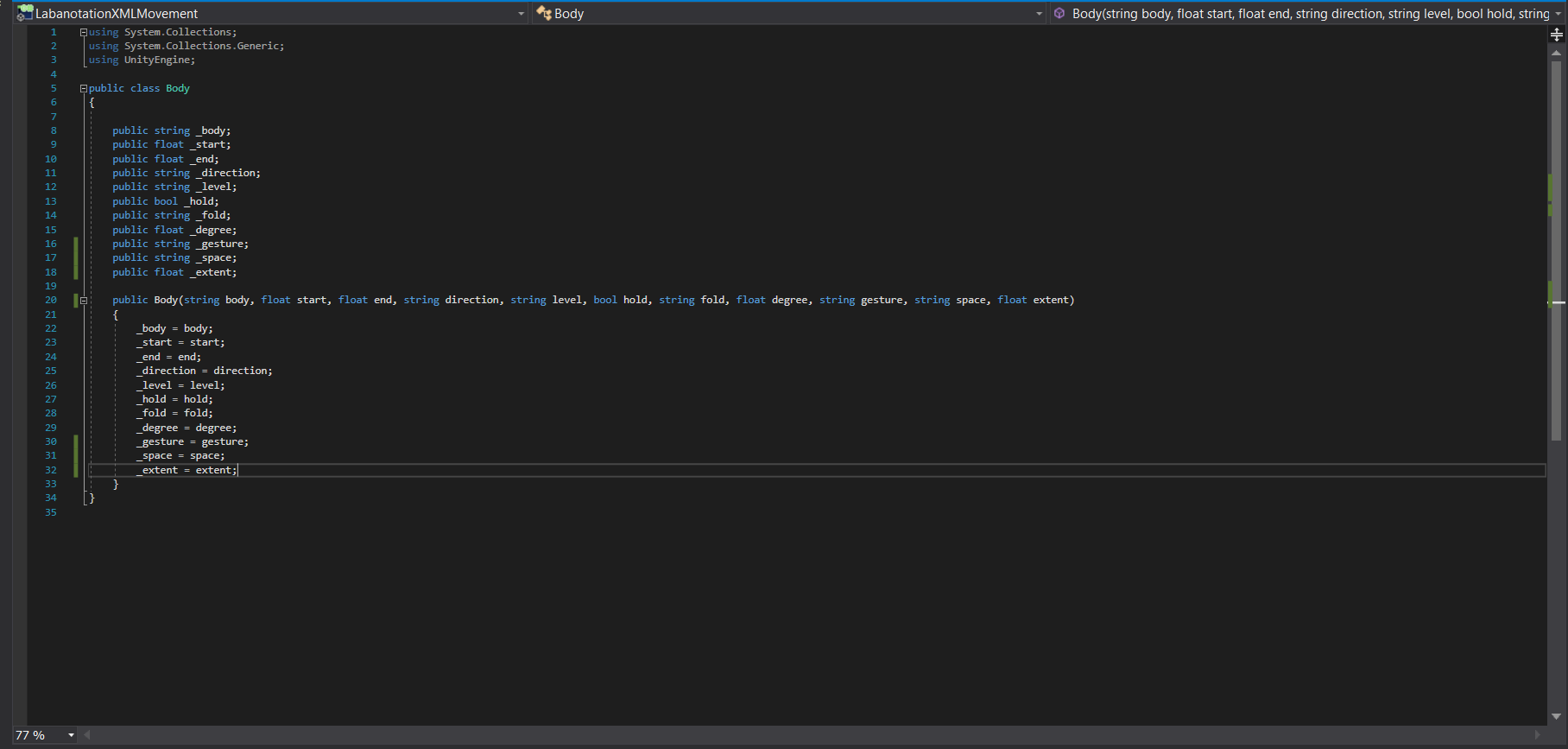
# Class Script.

There are two class script. One is called the Body class, the other is called the Measure class.

## Body class

This class takes in 11 arguments.

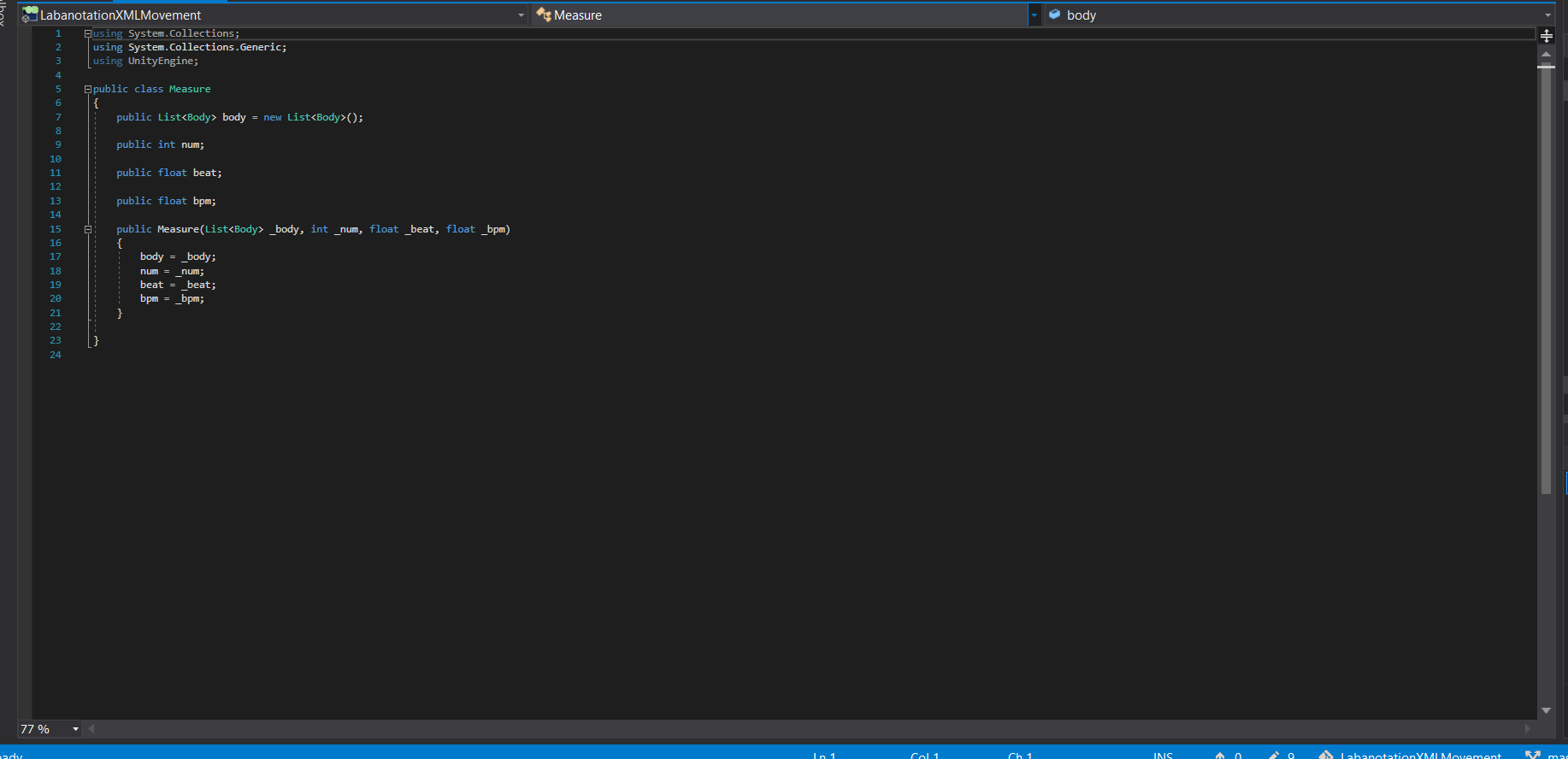
1. string body name
2. float beat start
3. float beat end
4. string direction of move
5. string level of move
6. bool hold
7. string fold
8. float degree of fold
9. string gesture
10. string space
11. float extent



## Measure class

This class takes in 4 arguments including a List of <BodyClass>.

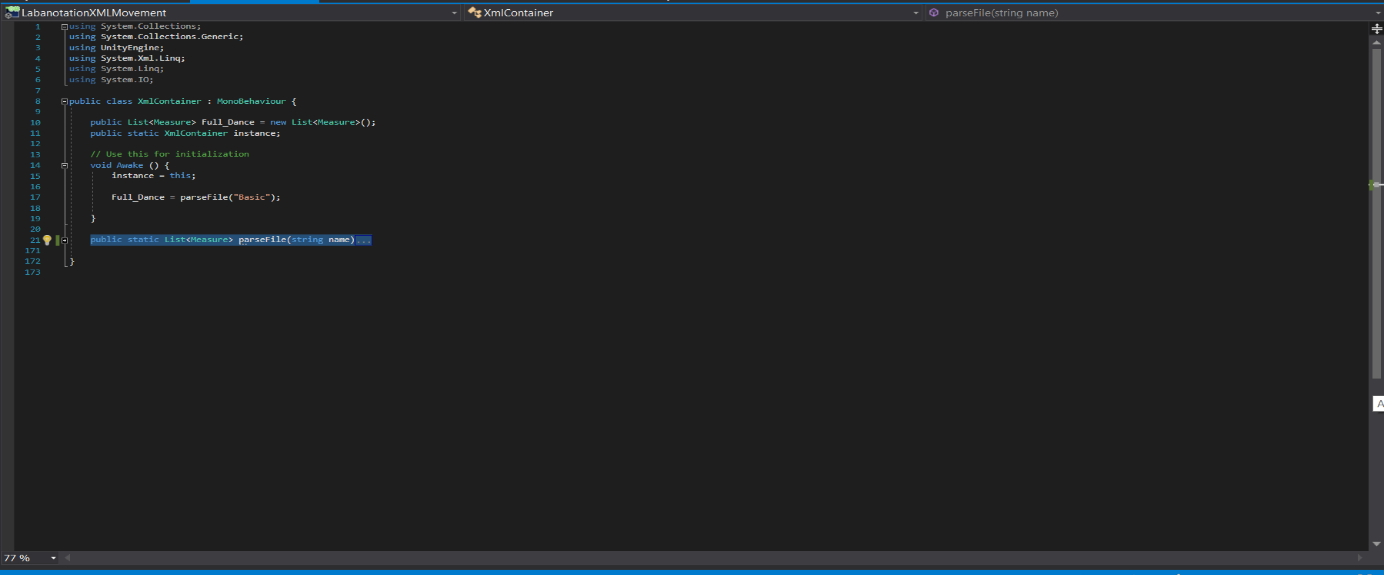
1. List<Body> a list of all the body part
2. int measure number
3. float number of beat in a measure
4. float beat per minute



# XmlContainerScript

## parseFile()

This script contains a method called parseFile. This method takes in a string argument. The argument is the name of the Xml file and it is case sensitive. The method will return a List of measures (List<Measure>).



# BodyControllerScript

This script contains 3 methods, 4 classes and 8 getMethods. There are float variables for Level and Fold. These values can be changed to normalise or exaggerate the models’ movement. There are 4 List of TimeSortedMoves. 3 of the List takes in Quaternion while the timeSortedHipPos takes in Vector3. The supports’ move has to be stored separately from other body part moves as it acts differently from the rest of the move.

### ProcessAndCreateJoints()

The script will get the bone transform of the model and for each component of the bone transform, a JointController script is added.

### GetandProcessXMLTask()

In Labanotation, the start and end of a move is defined by the beat. However, Unity recognises timing using seconds. This method will convert all the start and end of the dance moves, from beat to second. This method also gets the values for the direction and level, space and the extent of movement, or the fold and the degree of fold of the dance move. All the values will be stored in each specific list. The List for move, List for gesture, List for support, List for Hip position.

## Duration of a Beat

The formula for calculating duration of beat is: 60/bpm

TimePassed += (60/bpm) X total number of beat in current measure

Formula for Start and End of move:

(60/bpm) X start + TimePassed

(60/bpm) X end + TimePassed

\*TimePassed is the accumulation of seconds from the previous Measure.

## Support

The support move is different from other body movement as it affects the hip and the other leg. When the left support gets its direction Vector3, the right support will get the -X and -Z of the Vector3. This is used to animate the model, making it look like it took a step towards the specified direction.



### AssignTaskToJoints()

This method will assign all the joints with its specific task, from each TimeSorted variable. There are a total of 4 type of task.

Task calls for the rotate coroutine from the JointController.

Task2 calls for the translation of the hip position from the JointController.

Task3 calls for the rotate coroutine of the Support, from the JointController.

Task4 calls for the gesture rotate coroutine from the JointController.

# JointControllerScript

This script is attached to every single bone transform of the model. This script consists of 4 Task methods parsed from the BodyController script. The type of Task method parsed by the BodyController will call one of the 4 method Rotate(), GestureRotate(), SupportRotate() or Translate(). One of the 4 method will call a DelayCoroutine which will call the (rotate or translate Coroutine) and the StopCoroutine simultaneously. The rotate or translate Coroutine will stop once the delay for the StopCoroutine has exceeded its given timing. This script contains 7 Enumerator, 8 Methods.

## Lerp

The Enumerators uses the Lerp function. Lerp takes in 2 Vector3 and 1 float. The 1st Vector represents its current position or rotation. The 2nd Vector stores the position or rotation it needs to be. The float represents the position or rotation between the 1st and 2nd Vector, with 0.0f being the 1st Vector and 1.0f being the 2nd Vector. The float is continuously incrementing in a while loop from 0.0f to 1.0f. The Lerp function does nothing once the float exceeds 1.0f. Example (Lerp (Vector3 Start, Vector3 End, float t)). \*float t values between 0.0f – 1.0f.

## Number of Frame

For the Lerp to animate smoothly, the number of frame is set to 100 frames. This do not represent frame per second, it represents 100 frames within the duration of the dance move, 100frame/(End seconds – Start seconds). Since the 100 frame must be played within the Lerps’ float value of 1.0f, to calculate the increment of float t, take 1.0f / 100 frame. When a frame is finishes, t will increment by (1.0f/100).

## Update

In the Update(), if Hold is not true, the update will lerp back to its original position. If Hold is true, it will not run the Lerp in Update().

