เปียโนบนอากาศโดย LEAP MOTION
HAND FREE PIANO USING LEAP MOTION

โดย

นางสาววริษฐา ผึ้งฉ่ำ

นางสาววิริยา วานานวงศ์

นางสาวศศิอาภา จายะวานิช

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2559

เปียโนบนอากาศโดย LEAP MOTION

HAND FREE PIANO USING LEAP MOTION

โดย

นางสาววริษฐา	ผึ้งฉ่ำ	56011091
นางสาววิริยา	วานานวงศ์	56011141
นางสาวศศิอาภา	จายะวานิช	56011198

อาจารย์ที่ปรึกษา

ผศ.ดร. ธเนศ พัฒนธาดาพงษ์

ผศ.ดร. นภัทร สระเอี่ยม

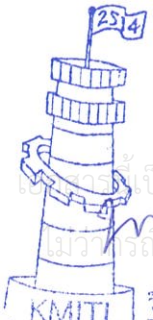
ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

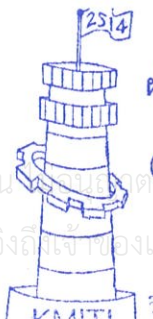
ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2559





ปริญญานิพนธ์ปีการศึกษา 2559

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เปียโนบนอากาศโดย LEAP MOTION

HAND FREE PIANO USING LEAP MOTION

ผู้จัดทำ

1. นางสาววริษฐา ผึ้งฉ่ำ 56011091
2. นางสาววิริยา วานานวงศ์ 56011141
3. นางสาวศศิอาภา จายะวานิช 56011198

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. ธเนศ พัฒนธาดาพงษ์)

..... อาจารย์ที่ปรึกษาร่วม

(ผศ.ดร. นภัทร สระเอี่ยม)

กิตติกรรมประกาศ

การจัดทำโครงงาน เรื่อง เปียโนบนอากาศโดย Leap Motion นี้สำเร็จลุล่วงไปได้ด้วยดี ด้วยความกรุณาจาก ผศ.ดร. ธเนศ พัฒนธาดาพงษ์ อาจารย์ที่ปรึกษาโครงงาน และ ผศ.ดร. นภัทร สระเอี่ยม อาจารย์ที่ปรึกษาร่วม สำหรับคำปรึกษา คำแนะนำและให้ความช่วยเหลือเมื่อเกิดปัญหาในการจัดทำโครงงาน รวมทั้งสนับสนุนสถานที่ เครื่องมือ และอุปกรณ์ต่าง ๆ ที่จำเป็นต้องใช้ในระหว่างจัดทำโครงงาน รวมถึงพี่ ๆ และเพื่อน ๆ ทุกคนที่คอยให้คำแนะนำและให้กำลังใจในการทำโครงงานตลอดมา

ผู้จัดทำขอขอบพระคุณทุกๆท่านเป็นอย่างสูง ณ ที่นี้ ที่ได้ช่วยให้การจัดทำโครงงานครั้งนี้สำเร็จลุล่วงไปได้ด้วยดี

นางสาววริษฐา ผึ้งฉ่ำ
นางสาววิริยา วานานวงศ์
นางสาวศศิอาภา จายะวานิช
ผู้จัดทำ

เปียโนบนอากาศโดย LEAP MOTION

HANDS FREE PIANO USING LEAP MOTION

โดย นางสาววริษฐา ผึ้งฉ่ำ 56011091
นางสาววิริยา วานานวงศ์ 56011141
นางสาวศศิอาภา จายะวานิช 56011198

อาจารย์ที่ปรึกษา ผศ.ดร. ธเนศ พัฒนธาดาพงษ์
อาจารย์ที่ปรึกษาร่วม ผศ.ดร. นภัทร สระเอี่ยม

บทคัดย่อ

โครงงานนี้มีวัตถุประสงค์เพื่อสร้างเกมเปียโนจำลองพื้นฐานเพื่อความบันเทิงและความแปลกใหม่ของรูปแบบในการควบคุมการเล่นเปียโน โดยใช้อุปกรณ์รับข้อมูลแบบ 3 มิติที่เรียกว่า ลีปโมชัน ผู้ใช้งานสามารถควบคุมการเล่นด้วยมือได้บนอากาศ โดยที่มือไม่ต้องสัมผัสคีย์บอร์ดหรือเม้าส์ รวมทั้งมีโหมดสอนเล่น จึงเหมาะสำหรับผู้เริ่มต้นเล่นเปียโนเป็นอย่างยิ่ง การจัดทำโครงงานแบ่งออกเป็นสองส่วน ส่วนที่หนึ่งเป็นส่วนที่เกี่ยวกับการสร้างเกมเปียโนจำลองโดยโปรแกรมยูนิตี้ ซึ่งเป็นซอฟแวร์ที่ใช้พัฒนาเกม 3 มิติบนระบบปฏิบัติการวินโดวส์ ทำการออกแบบเกมเปียโนจำลอง ทั้งลักษณะของเครื่องดนตรี เสียง และตำแหน่งของตัวโน้ต ส่วนที่สองเป็นการนำอุปกรณ์ลีปโมชันมาประยุกต์ใช้กับเกมเปียโนจำลอง เพื่อให้สามารถควบคุมการเล่นเปียโนผ่านการใช้ท่าทางและตำแหน่งของนิ้วมือ โดยใช้การป้อนข้อมูลของมนุษย์ในการประมวลผล จากการทดลองพบว่าผู้ใช้งานสามารถควบคุมการเล่นเปียโนบนอากาศด้วยมือ และเปียโนจำลองสามารถเล่นตามตำแหน่งตัวโน้ตได้อย่างถูกต้อง

ABSTRACT

The objective of this project is to create the basic hands-free virtual piano game for entertainment and innovative playing with 3D input device called Leap Motion. Users can control playing hands-freely without touching keyboard or mouse. And, there is tutorial mode so that the game is also suitable for beginners. The work is divided into two parts. The first part is about the virtual piano game using Unity program which is the software used to develop the 3D game on Windows OS. The designed piano game includes the characteristics of the instrument, sounds of notes and positions of notes. The second part is about applying a Leap Motion device to control the piano through gestures and positions of fingers by human input processing. After testing the game, it is found that users can control the piano with their hands on the air, and the virtual piano can give the correct sounds.

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

หน้า

รูปที่

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เปียโนเป็นเครื่องดนตรีที่นิยมมากในปัจจุบัน แต่เปียโนเป็นเครื่องดนตรีที่มีราคาสูงและมีขนาดใหญ่ ซึ่งเป็นข้อจำกัดสำหรับหลายๆ คนในการตัดสินใจลงทุนซื้อหรือหาพื้นที่สำหรับจัดวาง ดังนั้นคณะผู้จัดทำจึงทำการออกแบบโปรแกรมประยุกต์เปียโนจำลองถือว่าเป็นทางเลือกหนึ่งสำหรับผู้ที่มีความสนใจในการเล่นเปียโน โปรแกรมประยุกต์เปียโนจำลองใช้โปรแกรมยูนิตี้ (Unity) ซึ่งเป็นซอฟต์แวร์ที่ใช้พัฒนาเกม 3 มิติบนระบบปฏิบัติการวินโดวส์ที่มีความสมจริง มีความสวยงามของภาพ อย่างไรก็ตามการควบคุมการเล่นเปียโนผ่านคีย์บอร์ดหรือเมาส์แทนนิ้วมือทั้ง 10 นิ้วนั้นทำได้ไม่สะดวกและรวดเร็วพอ คณะผู้จัดทำจึงนำอุปกรณ์ลีปโมชัน (Leap Motion) มาประยุกต์ใช้ร่วมกับโปรแกรมประยุกต์เปียโนจำลอง เนื่องจากคุณสมบัติของอุปกรณ์ลีปโมชันสามารถตรวจจับตำแหน่งนิ้วมือผ่านท่าทางของผู้เล่น ทำให้สามารถควบคุมการทำงานต่างๆ ได้โดยการใช้ “มือ” หรือสิ่งของอื่นๆ ซึ่งการควบคุมการเล่นเปียโนสามารถทำได้บนอากาศโดยไม่ต้องสัมผัสกับหน้าจอ และตรวจจับท่าทางได้หลายแบบหลายทิศทาง ทำให้มีสะดวก สนุกสนานและมีความแปลกใหม่ในการเล่นเปียโนมากยิ่งขึ้น

1.2 วัตถุประสงค์

1) จำลองเครื่องดนตรีเปียโน เพื่อความบันเทิงและมีความแปลกใหม่
2) นำโปรแกรมประยุกต์เปียโนจำลองมาประยุกต์ใช้กับอุปกรณ์ลีปโมชัน ให้สามารถเล่นเปียโนได้บนอากาศ

1.3 ขอบเขตของปริญญานิพนธ์

นำอุปกรณ์ลีปโมชันมาใช้ร่วมกับโปรแกรมประยุกต์เปียโนจำลองที่ได้ออกแบบไว้ เพื่อให้สามารถควบคุมการเล่นเปียโนได้บนอากาศ และแสดงผลออกมาในรูปแบบของโปรแกรมประยุกต์เปียโนจำลองผ่านจอแสดงผล และเสียงผ่านทางลำโพง บล็อกไดอะแกรมภาพรวมของปริญญานิพนธ์แสดงดังรูปที่ 1.1

โปรแกรมยูนิตี้โดย
ใช้ภาษา C# ในการทำงาน

โปรแกรมประยุกต์เปียโนจำลอง
พร้อมมาร์กเกอร์แสดงตำแหน่งนิ้วมือ

แสดงตำแหน่งนิ้วมือบน
โปรแกรมประยุกต์เปียโน
จำลอง

เปล่งเสียงผ่านลำโพง

เคลื่อนไหวนิ้วมือบนอากาศ
เหนืออุปกรณ์ลีปโมชัน

อุปกรณ์ลีปโมชันตรวจจับนิ้วมือ

รูปที่ 1.1 บล็อกไดอะแกรมภาพรวมของปริญญานิพนธ์

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 เปียโน

เปียโนเป็นเครื่องดนตรีขนาดใหญ่ที่สร้างเสียงเมื่อคีย์ถูกกดและกลไกภายในเครื่องตีสาย คำว่า เปียโน เป็นตัวย่อของคำว่า ปีอาโนฟอเต (pianoforte) ซึ่งเป็นคำภาษาอิตาเลียนที่แปลว่า “เบาดัง” มาจากความสามารถของเปียโนที่จะปรับความดังเบาตามแรงที่กดคีย์

2.1.1 โครงสร้างคีย์บอร์ด

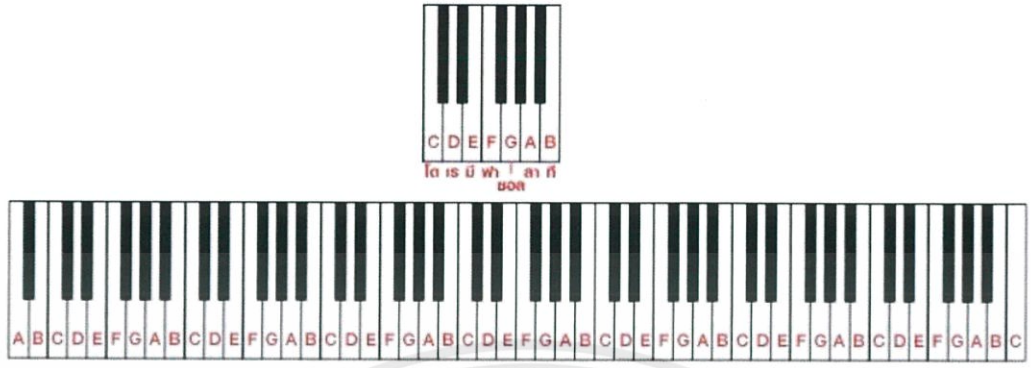
คีย์บอร์ดของเปียโนมาตรฐานมีทั้งหมด 88 คีย์ ประกอบด้วยคีย์สีขาว 52 คีย์และคีย์สีดำ 36 คีย์ ดังรูปที่ 2.1 แต่ละคีย์จะให้ระดับเสียงที่ต่างกัน ดังนั้นเปียโนจึงมีช่วงเสียงที่กว้างถึง 88 เสียง โดยยิ่งไปทางด้านซ้าย เสียงจะยิ่งต่ำและยิ่งไปทางด้านขวา เสียงจะยิ่งสูง

รูปที่ 2.1 คีย์บอร์ดของเปียโนมาตรฐาน

2.1.2 ชื่อคีย์บอร์ด

การเรียกชื่อคีย์สีขาวทั้ง 7 ตัวจะใช้ทั้งหมด 7 ชื่อ ดังรูปที่ 2.2 โดยการเรียกชื่อจะมี 2 ระบบ คือ

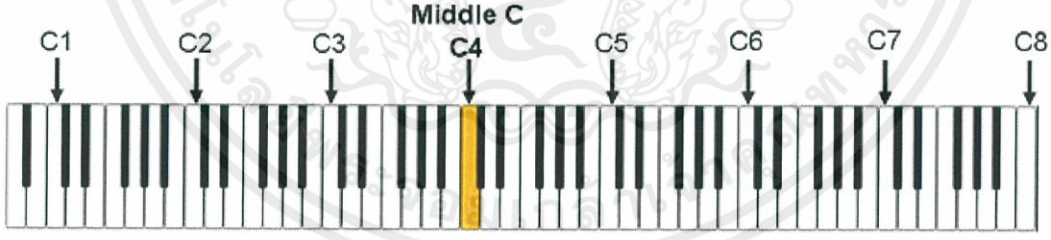
1. ระบบตัวอักษร ใช้ตัวอักษรภาษาอังกฤษ 7 ตัว ได้แก่ A B C D E F และ G
2. ระบบซอล-ฟา มี 7 ตัวเหมือนกัน คือ โด เร มี ฟา ซอล ลา และที โดยชื่อแต่ละตัวของทั้ง 2 ระบบมีความสัมพันธ์กัน คือ A-ลา, B-ที, C-โด, D-เร, E-มี, F-ฟา และ G-ซอล ส่วนคีย์สีดำทุกตัวจะมีชื่อเรียกอย่างใดอย่างหนึ่งว่า “ชาร์ป” (#) หรือ “แฟลต” (♭) ในการเรียกโดยอ้างอิงจากคีย์สีขาว



รูปที่ 2.2 ชื่อคีย์บอร์ด

2.1.3 การแบ่งคีย์บอร์ด

คีย์พื้นฐานจะแบ่งออกเป็นกลุ่ม ๆ แต่ละกลุ่มมี 12 คีย์ ประกอบด้วยคีย์สีขาว 7 ตัวและคีย์สีดำ 5 ตัว และแต่ละกลุ่มก็จะมีลักษณะซ้ำ ๆ กัน โดยคีย์ขาวทุกตัวอยู่ในบันไดเสียง C เมเจอร์ จากรูปที่ 2.3 สามารถหารูปแบบโดยเริ่มจากตัว C ที่อยู่ทางซ้ายของรูป และหาตัว C ที่อยู่ถัดมาทางขวาของแป้นคีย์ ซึ่งจะเห็นรูปแบบเดียวกับคีย์ก่อนหน้านี้ โดยจะซ้ำกันในทุก ๆ ช่วงคู่แปดของแป้นคีย์หรือเรียกว่า อ็อกเทฟ (octave) นั่นหมายถึง ช่วงเสียงของโน้ตตัวเดียวกันที่ห่างกันเป็นระยะ 8 ตัวโน้ต โดย Middle C เป็นตัวที่อยู่กึ่งกลางเปียโนและใช้เป็นตัวอ้างอิงในหลาย ๆ กรณี ซึ่งในปริญญานิพนธ์จะสร้างเปียโนจำลองจำนวน 4 อ็อกเทฟ

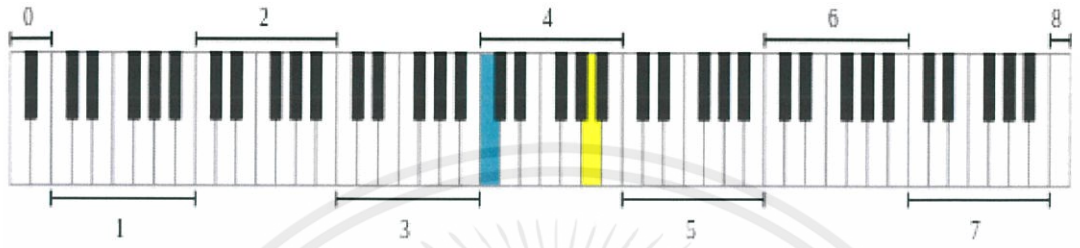


รูปที่ 2.3 การแบ่งคีย์บอร์ดเปียโน

2.1.4 ความถี่เสียงของเปียโน

จากรูปที่ 2.4 แสดงแผนผังเปียโน 88 คีย์ โดยมีเลขอ็อกเทฟกำกับ จำแนกสีสำหรับโน้ต C กลาง เป็นสีฟ้า และโน้ต A440 เป็นสีเหลือง ความถี่ของเสียงเปียโนโดยทั่วไปมีเสียงทั้งหมด 88 คีย์ โดยเทียบเสียงตามคีย์ A440 หรือ A4 ซึ่งจะแสดงความถี่ เป็นหน่วยรอบต่อวินาที หรือเฮิรตซ์ (Hz) ของโน้ตดนตรีแต่ละตัว

การกระจายความถี่นี้ เรียกกันว่า equal temperament ซึ่งคือ ระดับเสียงถัดไปแต่ละเสียงได้จากการคูณเสียงก่อนหน้านี้ด้วยรากที่ 12 ของ $2(\sqrt[12]{2})$ หรือประมาณ 1.05946309436 โดยความถี่ของเสียงเปียโนในแต่ละโน้ตแสดงดังตารางที่ 2.1



รูปที่ 2.4 แผนผังเปียโน 88 คีย์ แบ่งออกเป็น 8 อ็อกเทฟ

ตารางที่ 2.1 ความถี่ของเสียงเปียโนในแต่ละโน้ต

หมายเลขคีย์	ชื่อโน้ต	ความถี่ (Hz)
88	C8	4186.01
87	B7	3951.07
86	A#7/Bb7	3729.31
85	A7	3520.00
84	G#7/Ab7	3322.44
83	G7	3135.96
82	F#7/Gb7	2959.96
81	F7	2793.83
80	E7	2637.02
79	D#7/Eb7	2489.02
78	D7	2349.32
77	C#7/Db7	2217.46
76	C7	2093.00
75	B6	1975.53

ตารางที่ 2.1 (ต่อ) ความถี่ของเสียงเปียโนในแต่ละโน้ต

หมายเลขคีย์	ชื่อโน้ต	ความถี่ (Hz)
74	A#6/Bb6	1864.66
73	A6	1760.00
72	G#6/Ab6	1661.22
71	G6	1567.98
70	F#6/Gb6	1479.98
69	F6	1396.91
68	E6	1318.51
67	D#6/Eb6	1244.51
66	D6	1174.66
65	C#6/Db6	1108.73
64	C6	1046.50
63	B5	987.77
62	A#5/Bb5	932.33
61	A5	880.00
60	G#5/Ab5	830.61
59	G5	783.99
58	F#5/Gb5	739.99
57	F5	698.46
56	E5	659.26
55	D#5/Eb5	622.26
54	D5	587.33
53	C#5/Db5	554.37
52	C5	523.25
51	B4	493.88
50	A#4/Bd4	466.16

ตารางที่ 2.1 (ต่อ) ความถี่ของเสียงเปียโนในแต่ละโน้ต

หมายเลขคีย์	ชื่อโน้ต	ความถี่ (Hz)
49	A4 (A440)	440
48	G#4/Ab4	415.305
47	G4	391.995
46	F#4/Gb4	369.994
45	F4	349.228
44	E4	329.628
43	D#4/Eb4	311.127
42	D4	293.665
41	C#4/Db4	277.183
40	C4 (C กลาง)	261.626
39	B3	246.942
38	A#3/Bb3	233.082
37	A3	220
36	G#3/Ab3	207.652
35	G3	195.998
34	F#3/Gb3	184.997
33	F3	174.614
32	E3	164.814
31	D#3/Eb3	155.563
30	D3	146.832
29	C#3/Db3	138.591
28	C3	130.813
27	B2	123.471
26	A#2/Bb2	116.541
25	A2	110

ตารางที่ 2.1 (ต่อ) ความถี่ของเสียงเปียโนในแต่ละโน้ต

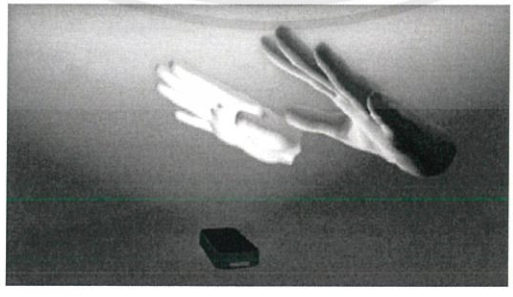
หมายเลขคีย์	ชื่อโน้ต	ความถี่ (Hz)
24	G#2/Ab2	103.826
23	G2	97.9989
22	F#2/Gb2	92.4986
21	F2	87.3071
20	E2	82.4069
19	D#2/Eb2	77.7817
18	D2	73.4162
17	C#2/Db2	69.2957
16	C2	65.4064
15	B1	61.7354
14	A#1/Bb1	58.2705
13	A1	55
12	G#1/Ab1	51.9130
11	G1	48.9995
10	F#1/Gb1	46.2493
9	F1	43.6536
8	E1	41.2035
7	D#1/Eb1	38.8909
6	D1	36.7081
5	C#1/Db1	34.6479
4	C1	32.7032
3	B0	30.8677
2	A#0/Bb0	29.1353
1	A0	27.5

2.2 ลีปโมชัน[4]

ลีปโมชัน (Leap Motion) คือ อุปกรณ์สำหรับเชื่อมต่อกับคอมพิวเตอร์ ตรวจจับการเคลื่อนไหวของมือ นิ้วมือและอุปกรณ์ที่มีรูปร่างทรงกระบอก เช่น ปากกา ดินสอ เป็นต้น ดังรูปที่ 2.5 อุปกรณ์ลีปโมชันตรวจจับด้วยอัตราการส่งข้อมูลที่ 120 เฟรมต่อวินาที เชื่อมต่อโดยสาย USB

รูปที่ 2.5 อุปกรณ์ลีปโมชัน

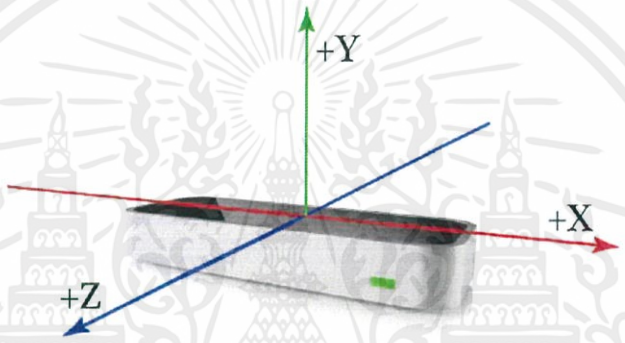
อุปกรณ์ลีปโมชันตรวจจับการเคลื่อนไหวด้วยความแม่นยำสูง และรายงานตำแหน่ง ท่าทางและการเคลื่อนไหวอย่างต่อเนื่อง โดยใช้เซ็นเซอร์แสงและแสงอินฟราเรด เซ็นเซอร์มีทิศทางในแนวแกน Y มุมมองประมาณ 150 องศา และมีประสิทธิภาพในระยะประมาณ 25 ถึง 600 มิลลิเมตร เหนืออุปกรณ์ ดังรูปที่ 2.6 การตรวจจับและการติดตามจะทำงานได้ดีเมื่อโครงร่างเงาของวัตถุชัดเจนและมีความคมชัดสูง ซอฟต์แวร์ของลีปโมชันจะรวมข้อมูลเซ็นเซอร์ที่มีโมเดลภายในของมือมนุษย์ เพื่อช่วยจัดการการติดตามการเคลื่อนไหวในรูปแบบต่างๆ นอกจากนี้ ลีปโมชันสามารถพัฒนา แอปพลิเคชันได้ในภาษาที่หลากหลาย เช่น ภาษา Python, C++, C# เป็นต้น การตรวจจับการเคลื่อนไหวโดยใช้ตัวควบคุมลีปโมชันในปริญญานิพนธ์นี้ใช้ Leap Motion C++ API ในการให้ข้อมูลตำแหน่งของปลายนิ้วมือ



รูปที่ 2.6 มุมมองการควบคุมลีปโมชันด้วยมือ

2.2.1 ระบบพิกัด

ระบบของลีปโมชันใช้ระบบพิกัดคาร์ทีเซียนมือขวา โดยตำแหน่งจุดกำเนิดอยู่ที่จุดศูนย์กลางบนตัวอุปกรณ์ ค่า X เป็นค่าแสดงระยะทางที่ห่างจากจุดกำเนิดตามแกนแนวยาวของอุปกรณ์ จะมีค่าเป็นบวกเมื่อตำแหน่งที่ถูกตรวจจับอยู่ทางด้านขวาของอุปกรณ์ ค่า Y เป็นค่าแสดงระยะทางที่ห่างจากจุดกำเนิดตามแนวตั้งของอุปกรณ์ โดยจะมีค่าเป็นบวกเสมอ และ ค่า Z จะเป็นค่าแสดงระยะทางที่ห่างจากจุดกำเนิดตามแกนแนวขวางของอุปกรณ์ จะมีค่าเป็นบวกเมื่อตำแหน่งที่ถูกตรวจจับอยู่ทางด้านหน้าของอุปกรณ์ ดังรูปที่ 2.7



รูปที่ 2.7 ระบบพิกัดมือขวาของลีปโมชัน

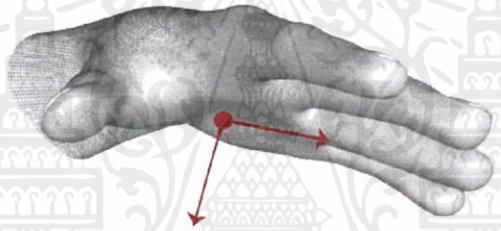
API ของลีปโมชันวัดปริมาณทางกายภาพด้วยหน่วยดังต่อไปนี้

ระยะทาง :	มิลลิเมตร
เวลา :	ไมโครวินาที
ความเร็ว :	มิลลิเมตร/วินาที
มุม :	เรเดียน

2.2.2 ข้อมูลการติดตามการเคลื่อนไหว

ลีปโมชันติดตามการเคลื่อนไหวของมือ นิ้วมือ และอุปกรณ์ในมุมมองของตัวเอง สามารถอัพเดตเป็นชุดเฟรม หรือข้อมูลก็ได้ แต่ละวัตถุเฟรม (Frame Object) จะแทนเฟรมที่ประกอบด้วยสิ่งที่ถูกติดตามการเคลื่อนไหว เช่น มือ นิ้วมือและอุปกรณ์ รวมถึงท่าทางและองค์ประกอบที่อธิบายการเคลื่อนไหวโดยรวมที่เกิดขึ้นบนจอ วัตถุเฟรมจึงเป็นพื้นฐานที่จำเป็นของโมเดลข้อมูลลีปโมชัน ซึ่งข้อมูลการเคลื่อนไหวมีรายละเอียดดังต่อไปนี้

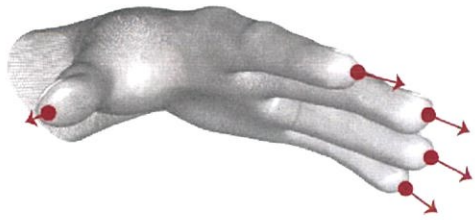
1) มือ: โมเดลมือให้ข้อมูลเกี่ยวกับการระบุตัวตน ตำแหน่ง และลักษณะอื่นๆ ของมือที่ตรวจจับได้ รวมถึงแขนที่ติดมากับมือและส่วนของนิ้วมือที่สัมพันธ์กับมือ มือจะถูกแทนด้วย Hand Class[2] ดังรูปที่ 2.8 ซอฟต์แวร์ของลีปโมชันใช้โมเดลภายในของมือมนุษย์เพื่อทำนายการติดตามเมื่อชิ้นส่วนของมือนั้นมองไม่เห็น โมเดลมือให้ตำแหน่งทั้งห้านิ้วมือเสมอ อย่างไรก็ตามการติดตามที่ดีที่สุดคือโครงร่างเงาของมือและทั้งห้านิ้วมือมองเห็นได้อย่างชัดเจน ซอฟต์แวร์ใช้ส่วนที่มองเห็นได้ของโมเดลภายในมือและข้อสังเกตที่ผ่านมาในการคำนวณตำแหน่งที่ใกล้เคียงมากที่สุดของส่วนที่ไม่สามารถมองเห็นได้ในขณะนั้น หากการเคลื่อนไหวของนิ้วมือขัดกับมือหรือส่วนป้องกันจากเซ็นเซอร์ของลีปโมชัน มักไม่พบการตรวจจับ หากมีมือของผู้อื่นอยู่ในมุมมองในเฟรมจะปรากฏมือมากกว่าสองมือ อย่างไรก็ตามควรรักษาทั้งสองมือให้อยู่ในขอบเขตมุมมองของตัวควบคุมลีปโมชันให้มากที่สุดเพื่อการติดตามการเคลื่อนไหวที่มีประสิทธิภาพ



รูปที่ 2.8 เวกเตอร์ palm_normal และเวกเตอร์ทิศทางกำหนดแนวการวางของมือ

2) แขน: แขนเป็นวัตถุคล้ายกระดูกที่มีการวางแนว ความยาว ความกว้าง และจุดสิ้นสุดของแขน โดยข้อศอกไม่ได้อยู่ในมุมมอง ตัวควบคุมลีปโมชันจะประมาณตำแหน่งตามการสังเกตที่ผ่านมาเช่นเดียวกับสัดส่วนทั่วไปของมนุษย์

3) นิ้วมือ: ตัวควบคุมลีปโมชันให้ข้อมูลเกี่ยวกับนิ้วแต่ละนิ้ว ถ้าทั้งหมดหรือบางส่วนของนิ้วมือนั้นมองไม่เห็น จะมีการประมาณลักษณะนิ้วมือตามการสังเกตที่ผ่านมาและโมเดลทางกายวิภาคของมือ นิ้วมือถูกระบุโดยชื่อประเภท นั่นคือ นิ้วหัวแม่มือ (thumb), นิ้วชี้ (index), นิ้วกลาง (middle), นิ้วนาง (ring) และนิ้วก้อย (pinky) นิ้วมือจะถูกแสดงโดย Finger Class ดังรูปที่ 2.9



รูปที่ 2.9 เวกเตอร์ตำแหน่งปลายนิ้วและเวกเตอร์ทิศทางที่แต่ละนิ้วชี้ออกมา

Finger Class เป็นการแสดงนิ้วมือที่ถูกตรวจจับโดยอุปกรณ์ลีปโมชันและสามารถเรียกรายละเอียดต่าง ๆ ของนิ้วมือได้ เช่น ตำแหน่ง ความเร็ว และ ขนาด เป็นต้น ซึ่งรายละเอียดที่สามารถเรียกใช้งานได้นั้น แสดงดังตารางที่ 2.2 โดยการใช้คำสั่ง Finger() เนื่องจากนิ้วมือเป็นอ็อบเจกต์ที่ซอฟต์แวร์ของลีปโมชันได้จำแนกประเภทของนิ้วมือของเป็น 5 ประเภท ดังที่กล่าวไว้ข้างต้น ดังนั้นจึงสามารถเรียกอ็อบเจกต์นิ้วมือจากเฟรม Frame.Hands.Finger() [11] หรืออ็อบเจกต์มือ Hands.Fingers() [12] การเรียกใช้งานนิ้วมือแต่ละชนิดสามารถเรียกใช้งานได้จากคำสั่ง Finger.type() [13] โดยการระบุชนิดของนิ้วมือลงไปภายในวงเล็บ ซึ่งใช้คำสั่งดังต่อไปนี้ TYPE_THUMB คือนิ้วโป้ง TYPE_INDEX คือนิ้วชี้ TYPE_MIDDLE คือนิ้วกลาง TYPE_RING คือนิ้วนาง และ TYPE_PINKY คือนิ้วก้อย

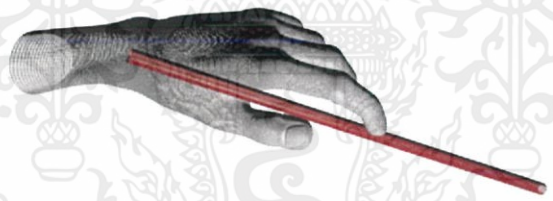
ตารางที่ 2.2 รายละเอียดนิ้วมือที่สามารถเรียกใช้งานได้

ชนิดของตัวแปร	ตัวแปรรายละเอียดนิ้วมือ
long	frameId
int	handId
int	fingerId
float	timeVisible
Vector	tipPosition
Vector	tipVelocity
Vector	direction
Vector	stabilizedTipPosition
float	width
float	length
Finger	FingerType type

โดยที่

frameId	คือ ID ของเฟรมนิ้วมือที่ปรากฏ
handId	คือ ID ของมือข้างที่นิ้วมือที่ปรากฏ
fingerId	คือ ID ของนิ้วมือที่ปรากฏ
timeVisible	คือ ระยะที่มองเห็นนิ้วมือ ระยะที่อุปกรณ์สามารถตรวจจับได้
tipPosition	คือ ตำแหน่งของปลายนิ้วมือ
tipVelocity	คือ ความเร็วของปลายนิ้วมือ
Direction	คือ ทิศทางของนิ้วมือ
stabilizedTipPosition	คือ ตำแหน่งปลายนิ้วมือที่เสถียร
width	คือ ความกว้างเฉลี่ยของนิ้วมือ
length	คือ ความยาวของนิ้วมือ
type	คือ ชนิดของนิ้วมือ

4) อุปกรณ์: อุปกรณ์ที่ลีปโมชันตรวจจับได้ต้องมีรูปร่างทรงกระบอก ซึ่งอุปกรณ์จะมีความยาว บางและตรงกว่านิ้วมือ ดังรูปที่ 2.10 อุปกรณ์จะถูกแสดงโดย Tool Class



รูปที่ 2.10 โมเดลอุปกรณ์

5) ท่าทาง: ซอฟต์แวร์ของลีปโมชันรับรู้รูปแบบการเคลื่อนไหว เช่น ท่าทาง ซึ่งอาจแสดงถึงความตั้งใจของผู้ใช้หรือคำสั่ง และรายงานท่าทางที่สังเกตได้ เช่นเดียวกับข้อมูลการติดตามการเคลื่อนไหวอื่นๆ เช่น นิ้วมือและมือ ท่าทางจะถูกแสดงโดย Gesture Class และคลาสย่อย ได้แก่ วงกลม (CircleGesture), การแตะ (KeyTapGesture, ScreenTapGesture) และการปัด (SwipeGesture) รูปแบบการติดตามการเคลื่อนไหวโดยซอฟต์แวร์ของลีปโมชัน ดังรูปที่ 2.11 โดยก่อนใช้ท่าทางในการประยุกต์ใช้งาน ต้องเปิดใช้งานการรับรู้สำหรับแต่ละท่าทาง คลาสที่ใช้ควบคุมคือ enableGesture() สามารถใช้เพื่อเปิดใช้งานการรับรู้ประเภทของท่าทางที่ใช้

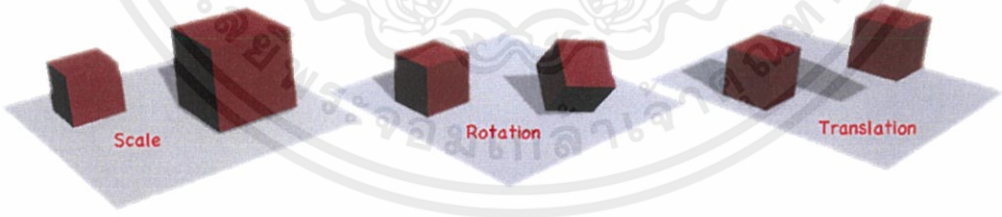
(ก) วงกลม-นิ้ววาดเป็นวงกลม (ข) การปัด-การเคลื่อนไหวเชิงเส้นของมือและนิ้วมือ

(ค) การแตะ-การเคลื่อนไหวโดยการแตะด้วยนิ้วถ้ากดแป้นพิมพ์ (ง) การแตะที่หน้าจอ-การเคลื่อนไหวโดยการแตะหน้าจอคอมพิวเตอร์ด้วยนิ้วในแนวตั้ง

รูปที่ 2.11 รูปแบบการเคลื่อนไหวต่าง ๆ

6) การเคลื่อนไหว

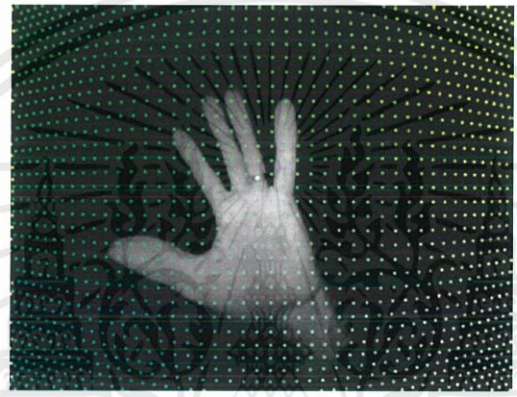
การประมาณการตามประเภทของการเคลื่อนไหวพื้นฐานในการเปลี่ยนแปลงของมือของผู้ใช้ในช่วงระยะเวลาหนึ่ง การเคลื่อนไหวจะรวมการสเกล การหมุนและการแปลง (การเปลี่ยนแปลงในตำแหน่ง) ดังรูปที่ 2.12



รูปที่ 2.12 การเคลื่อนไหวพื้นฐานในการเปลี่ยนแปลงของมือของผู้ใช้

2.2.3 เซ็นเซอร์ภาพ

ควบคู่กับการคำนวณข้อมูลการติดตาม จะได้รับภาพเซ็นเซอร์ (raw sensor images) จากกล้องของลีปโมชัน ดังรูปที่ 2.13 ข้อมูลภาพจะรวมการวัดค่าความสว่างของรังสีอินฟราเรด และการเปรียบเทียบข้อมูลที่ต้องการความถูกต้องสำหรับความผิดเพี้ยนของเลนส์ สามารถใช้ภาพเซ็นเซอร์เพื่อการใช้งานแบบสมจริง โดยเฉพาะอย่างยิ่งเมื่อฮาร์ดแวร์ของลีปโมชันติดตั้งกับชุดหูฟังเสมือนจริง (VR Headset)

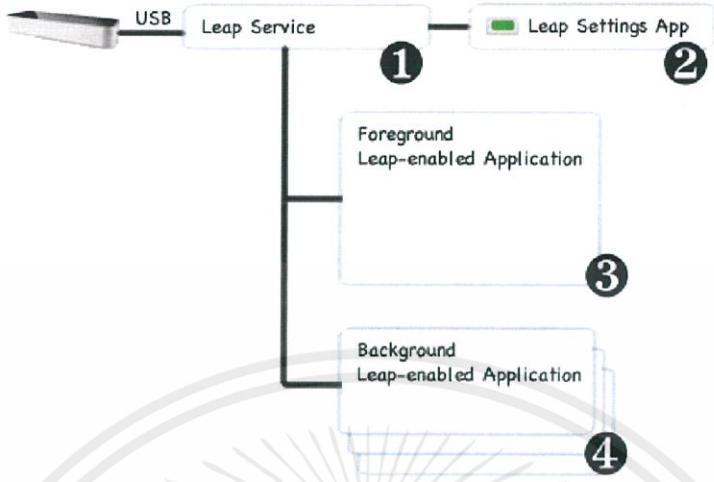


รูปที่ 2.13 ภาพเซ็นเซอร์ที่ถูกเทียบกับจุดทดสอบ

2.2.4 โครงสร้างพื้นฐาน

ซอฟต์แวร์ของลีปโมชันทำงานเป็น service รันบน Windows หรือทำงานเป็น daemon รันบน Mac และ Linux โดยซอฟต์แวร์เชื่อมต่อกับอุปกรณ์ลีปโมชันผ่านทาง USB แอปพลิเคชันของลีปโมชันที่เปิดใช้งานเข้าถึงบริการของลีปโมชันเพื่อรับข้อมูลการติดตามการเคลื่อนไหว การรับข้อมูลของลีปโมชันจาก service ของลีปโมชันนั้น SDK ของลีปโมชันจะให้ช่องทางการเชื่อมต่อแบบเนทีฟอินเตอร์เฟซ ซึ่งช่องทางการเชื่อมต่อจะช่วยสร้างแอปพลิเคชันของลีปโมชันได้ในหลายภาษาโปรแกรม

เนทีฟอินเตอร์เฟซ เป็นอินเตอร์เฟซผ่านไลบรารี dynamic loading ไลบรารีนี้เชื่อมต่อกับ service ของลีปโมชัน และให้ข้อมูลการติดตามการเคลื่อนไหวกับแอปพลิเคชัน โดยสามารถเชื่อมต่อกับไลบรารีโดยตรง ซึ่งภาษาที่รองรับทั้งหมด 5 ภาษา ประกอบไปด้วยภาษา C++ Objective-C Java C# และ Python ซึ่งสามารถเลือกใช้งานภาษาใดก็ได้จาก 5 ภาษานี้



รูปที่ 2.14 การส่งข้อมูลให้แอปพลิเคชันของลีปโมชันที่เปิดใช้งาน

จากรูปที่ 2.14 แสดงการส่งข้อมูลให้แอปพลิเคชันของลีปโมชันที่เปิดใช้งาน ดังนี้

1. service ของลีปโมชันได้รับเฟรมข้อมูลจากอุปกรณ์ลีปโมชันผ่านทาง USB จากนั้นประมวลผลข้อมูลและส่งไปยังแอปพลิเคชันของลีปโมชันที่เปิดใช้งาน โดยค่าเริ่มต้น service จะส่งข้อมูลการติดตามไปยังแอปพลิเคชันที่รันอยู่ (Foreground) เท่านั้น อย่างไรก็ตาม service สามารถส่งคำร้องขอในการรับข้อมูลในแอปพลิเคชันที่ไม่ได้รันอยู่ (Background) ได้

2. แอปพลิเคชันของลีปโมชันทำงานแยกจาก service และช่วยผู้ใช้งานคอมพิวเตอร์ในการตั้งค่าลีปโมชัน แอปพลิเคชันของลีปโมชันเป็น Control Panel applet บน Windows และเป็นแถบเมนูบน Mac OS X

3. แอปพลิเคชัน Foreground รับข้อมูลการติดตามการเคลื่อนไหวจาก service แอปพลิเคชันของลีปโมชันสามารถเชื่อมต่อไปยัง service ของลีปโมชันโดยใช้ไลบรารีเนทีฟลีปโมชัน แอปพลิเคชันสามารถเชื่อมโยงกับไลบรารีเนทีฟลีปโมชันโดยตรง (C++ และ Objective-C) หรือผ่านการโปรแกรมด้วยภาษา Java, C# และ Python อย่างใดอย่างหนึ่ง

4. เมื่อแอปพลิเคชันของลีปโมชันที่เปิดใช้งานมีการสูญเสียการเชื่อมต่อ service ของลีปโมชันจะหยุดส่งข้อมูล แอปพลิเคชันจะทำงานใน Background และส่งคำร้องขอรับข้อมูลเมื่ออยู่ใน Background แล้ว การตั้งค่าจะถูกกำหนดโดยแอปพลิเคชัน Foreground

2.3 ซอฟต์แวร์ของอุปกรณ์ลีปโมชัน[5]

ตัวควบคุมอุปกรณ์ลีปโมชันประกอบไปด้วยส่วนประกอบฮาร์ดแวร์และซอฟต์แวร์ ฮาร์ดแวร์ของอุปกรณ์ลีปโมชัน ประกอบไปกล้องอินฟราเรดสเตอริโอ 2 ตัวและไฟ LED อินฟาเรด 3 ตัว เซ็นเซอร์กล้องส่องขึ้นไปด้านบน (เมื่ออุปกรณ์อยู่ในแนวราบ) ดังรูปที่ 2.6 ซอฟต์แวร์ของอุปกรณ์ลีปโมชัน จะได้รับข้อมูลเซ็นเซอร์และวิเคราะห์ข้อมูลนี้ โดยเฉพาะมือ นิ้วมือและแขน ซอฟต์แวร์จะเก็บโมเดลภายในของมนุษย์ไว้และเปรียบเทียบแบบจำลองกับข้อมูลเซ็นเซอร์ เพื่อหาข้อมูลที่เหมาะสมที่สุด มีการวิเคราะห์ข้อมูลเซนเซอร์แบบทีละเฟรมและ service จะส่งข้อมูลแต่ละเฟรมไปยังแอปพลิเคชันที่เปิดใช้อุปกรณ์ลีปโมชัน วัตถุเฟรมที่แอปพลิเคชันได้รับประกอบด้วยตำแหน่งความเร็วและข้อมูลที่ติดตาม เช่น มือและนิ้วมือ ซอฟต์แวร์ของอุปกรณ์ลีปโมชันทำงานเป็นบริการ (Windows) หรือ daemon (Mac และ Linux) บนคอมพิวเตอร์ แอปพลิเคชันที่เปิดใช้อุปกรณ์ลีปโมชันสามารถเชื่อมต่อกับบริการนี้โดยใช้ไลบรารีแบบไดนามิกของอุปกรณ์ลีปโมชัน โดยจัดเตรียม API ให้เป็นส่วนหนึ่งของอุปกรณ์ลีปโมชัน SDK แอปพลิเคชันเว็บสามารถเชื่อมต่อกับเซิร์ฟเวอร์ WebSocket ที่โฮสต์โดยบริการนี้ WebSocket ให้ข้อมูลการติดตามเป็นข้อความที่จัดรูปแบบ JSON นั้นคือหนึ่งข้อความต่อเฟรมของข้อมูล

2.3.1) การตั้งต่าไฟล์และทดสอบโปรแกรมเริ่มต้น

1) ดาวน์โหลด Leap Motion SDK ล่าสุดจากเว็บไซต์นักพัฒนาซอฟต์แวร์[6] และเปิดเครื่องอุปกรณ์ลีปโมชัน

2) เปิดหน้าต่างคอนโซลและไปที่โฟลเดอร์ตัวอย่างใน SDK

3) เลือกสคริปต์ตัวอย่าง Sample.cs ซึ่งเป็นคำสั่งสำเร็จรูปเพื่อใช้ในการเริ่มต้น สามารถเปลี่ยนชื่อไฟล์ที่มีอยู่แล้วและสร้างไฟล์ Sample.cs ใหม่ที่ว่างเปล่าไว้ในโฟลเดอร์นี้ นั่นคือ การเก็บไฟล์ที่มีอยู่เพื่อใช้อ้างอิง

4) ในไฟล์ Sample.cs ใหม่ ให้เพิ่ม LeapNameSpace ดังรูปที่ 2.15 เพื่ออ้างอิงคลาสอุปกรณ์ลีปโมชัน

```
using leap;
```

รูปที่ 2.15 LeapNameSpace สำหรับอ้างอิงคลาสอุปกรณ์ลีปโมชัน

5) เพิ่มโครงสร้างเพื่อกำหนดโปรแกรมคำสั่งดังรูปที่ 2.16 ซึ่งคำสั่งนี้จะแสดงข้อความเพียงอย่างเดียวและรอให้ป้อนข้อมูลจากแป้นพิมพ์ก่อน

```
class Sample
{
    public static void Main ()
    {
        // Keep this process running until Enter is pressed
        Console.WriteLine ("Press Enter to quit...");
        Console.ReadLine ();
    }
```

รูปที่ 2.16 โครงสร้างเพื่อกำหนดโปรแกรมคำสั่งแสดงทิศทางพร้อมใช้งาน

2.3.2) การเชื่อมต่อ

ขั้นตอนต่อไปคือการเพิ่มอ็อบเจ็กต์การควบคุมลงในโปรแกรมดังรูปที่ 2.17 ซึ่งทำหน้าที่เป็นการเชื่อมต่อกับอุปกรณ์ลีปโมชัน

```
class Sample
{
    public static void Main ()
    {
        Controller controller = new Controller ();

        // Keep this process running until Enter is pressed
        Console.WriteLine ("Press Enter to quit...");
        Console.ReadLine ();
    }
}
```

รูปที่ 2.17 เพิ่มอ็อบเจ็กต์การควบคุมลงในโปรแกรม

เมื่อสร้างอ็อบเจ็กต์การควบคุมจะเชื่อมต่อกับบริการอุปกรณ์ลีปโมชันโดยอัตโนมัติและเมื่อตั้งค่าการเชื่อมต่อแล้วจะสามารถรับข้อมูลการติดตามได้โดยใช้ Controller.Frame() อ็อบเจ็กต์การควบคุมจะส่งข้อมูลจำนวนสถานะที่สามารถใช้เพื่อรับข้อมูลการติดตาม สถานะเหล่านี้ ได้แก่ Connect Device และ Frame Ready สถานะการเชื่อมต่อจะถูกส่งเมื่อแอปพลิเคชันเชื่อมต่อกับบริการอุปกรณ์ลีปโมชันเรียบร้อยแล้ว สถานะของการเชื่อมต่ออุปกรณ์จะถูกส่งไปเมื่ออุปกรณ์ลีปโมชันพร้อมใช้งานซึ่งอาจเป็นได้ทันทีหลังจากเชื่อมต่อหรือเมื่อเสียบปลั๊ก ส่วนสถานะของ Frame Ready จะถูกส่งก็ต่อเมื่อเฟรมมีการติดตามหรือตรวจจับใหม่

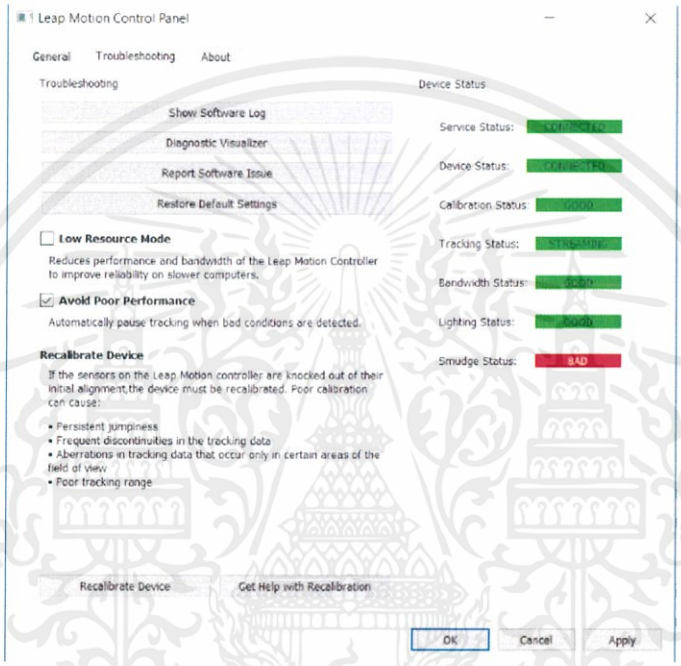
สำหรับคำแนะนำการใช้งานในหัวข้อถัดไป จึงทำการเพิ่มคลาส SampleListener ลงในโปรแกรม เพื่อกำหนดฟังก์ชันตัวจัดการสถานะดังรูปที่ 2.18 และทำการลองรันคำสั่งตัวอย่าง

```
class SampleListener
{
    public void OnServiceConnect(object sender,
ConnectionEventArgs args)
    {
      Console.WriteLine("Service Connected");
    }

    public void OnConnect(object sender, DeviceEventArgs args)
    {
      Console.WriteLine("Connected");
    }

    public void OnFrame(object sender, FrameEventArgs args)
    {
      Console.WriteLine("Frame Available.");
    }
}

class Sample
{
    public static void Main ()
    {
        Controller controller = new Controller ();
        SampleListener listener = new SampleListener();
        controller.Connect += listener.OnServiceConnect;
        controller.Device += listener.OnConnect;
        controller.FrameReady += listener.OnFrame;

  //Keep this process running until Enter is pressed
        Console.WriteLine ("Press Enter to quit...");
        Console.ReadLine ();

        controller.RemoveListener (listener);
        controller.Dispose ();
    }
}
```

รูปที่ 2.18 คลาส SampleListener เพื่อกำหนดฟังก์ชันตัวจัดการสถานะ

หลังจากทดสอบโปรแกรมตัวอย่างเรียบร้อยหากทุกอย่างถูกต้อง (และเชื่อมต่อฮาร์ดแวร์ของอุปกรณ์ลีปโมชัน) จะเห็นข้อความ "Service Connected" และ "Connected" แสดงบนหน้าต่างเทอร์มินัลตามด้วยชุด "Frame Available" หากมีสิ่งผิดพลาดเกิดขึ้นและไม่สามารถเข้าใจได้ว่าเกิดจากสาเหตุอะไรสามารถขอความช่วยเหลือได้ที่ฟอรัมนักพัฒนาซอฟต์แวร์ที่[7]

และเมื่อใดก็ตามที่พบปัญหาในการพัฒนาแอปพลิเคชันของอุปกรณ์ลีปโมชันให้ลองเปิดหน้าต่าง Leap Motion Control Panel ดังรูปที่ 2.19 หรือ visualizer ในการตรวจสอบ โปรแกรมนี้จะแสดงภาพข้อมูลการติดตามการเคลื่อนไหวของอุปกรณ์ลีปโมชันและสามารถเปรียบเทียบสิ่งที่เห็นในโปรแกรมกับสิ่งที่เห็นในเครื่องมือสร้างภาพ (ซึ่งใช้ API เดียวกัน) เพื่อแยกและระบุปัญหาจำนวนมาก



รูปที่ 2.19 หน้าต่าง Leap Motion Control Panel

1) สถานะ Connect Device เมื่อออบเจ็กต์การควบคุมเชื่อมต่อกับบริการอุปกรณ์ลีปโมชันได้สำเร็จจึงจะส่งสถานะ Connect Device แต่อุปกรณ์จะไม่ส่งข้อมูลแบบสตรีมมิ่ง

2) สถานะ Frame Ready ข้อมูลการติดตามในระบบของอุปกรณ์ลีปโมชันในรูปแบบของเฟรมวัตถุ สามารถเรียกใช้วัตถุเฟรมจากคอนโทรลเลอร์ (หลังจากที่ได้เชื่อมต่อและอุปกรณ์กำลังสตรีมมิ่ง) โดยเรียกใช้ Controller.Frame() ตัวจัดการสถานะใดๆ ที่รับสถานะ Frame Ready หากไม่ได้รับสามารถเปรียบเทียบค่า ID กับเฟรมสุดท้ายที่ประมวลผลเพื่อดูว่ามีเฟรมใหม่หรือไม่

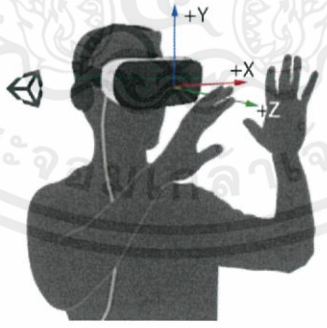
2.4 ลีปโมชันใช้งานร่วมกับโปรแกรมยูนิตี้[8]

ตัวควบคุมลีปโมชันติดตามมือและนิ้วมือและรายงานตำแหน่ง ความเร็วและทิศทาง โดยใช้เวลาน้อยและมีความแม่นยำสูง ตัวควบคุมสามารถติดตั้งบนชุดหูฟังเสมือนจริง (VR Headset) หรือใช้งานบนโต๊ะ

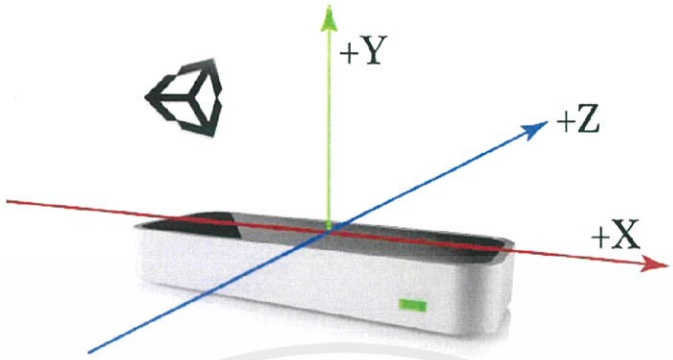
ระบบของตัวควบคุมลีปโมชันประกอบด้วยอุปกรณ์ฮาร์ดแวร์และองค์ประกอบของซอฟต์แวร์ที่ทำงานเป็นเซอร์วิสหรือดีมอนบนโฮสต์คอมพิวเตอร์ องค์ประกอบของซอฟต์แวร์วิเคราะห์ภาพที่ถูกผลิตโดยฮาร์ดแวร์และส่งข้อมูลการติดตามไปยังแอปพลิเคชัน ปลั๊กอินของลีปโมชันที่ใช้งานร่วมกับโปรแกรมยูนิตี้จะเชื่อมต่อกับบริการนี้เพื่อให้ได้ข้อมูล สคริปต์จะมีปลั๊กอินที่แปลพิกัดของลีปโมชันไปยังระบบพิกัดของโปรแกรมยูนิตี้ สคริปต์และส่วนที่ใส่รายละเอียดของภาพ (Graphic Assets) ที่เพิ่มเติมขึ้นมานี้ ทำให้ง่ายต่อการเพิ่ม 3 มิติ และการควบคุมการเคลื่อนไหวด้วยมือไปยังฉากในโปรแกรมยูนิตี้

2.4.1 ระบบพิกัด

โปรแกรมยูนิตี้ใช้ระบบพิกัดมือซ้าย แกน Z จะมีทิศทางตรงกันข้ามกับ API ของลีปโมชัน ซึ่งใช้ระบบพิกัดมือขวา นอกจากนั้นค่าเริ่มต้นในโปรแกรมยูนิตี้ใช้หน่วยเมตร ในขณะที่ API ของลีปโมชันใช้หน่วยมิลลิเมตร ปลั๊กอินภายในสคริปต์จะแปลงข้อมูลการติดตามให้ใช้ระบบพิกัดมือซ้าย ดังรูปที่ 2.20 และ 2.21 ตามลำดับ และสเกลระยะทางโดยใช้หน่วยเมตร



รูปที่ 2.20 ระบบพิกัดของโปรแกรมยูนิตี้ซ้อนทับบนอุปกรณ์ลีปโมชัน
ในจอภาพแบบสวมศีรษะ

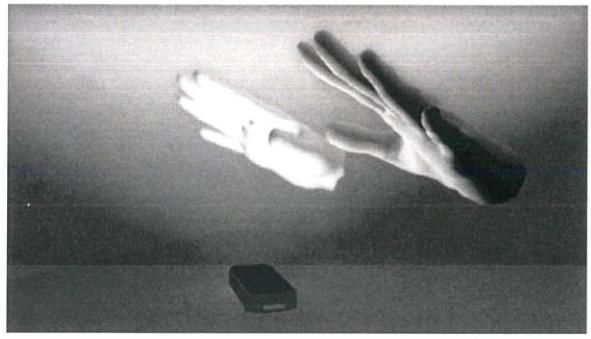


รูปที่ 2.21 ระบบพิกัดมือซ้ายของโปรแกรมยูนิตี้ซ้อนทับบนอุปกรณ์ลีปโมชันบนพื้นโต๊ะ

2.4.2 การติดตามมือ

ตัวควบคุมลีปโมชันใช้เซนเซอร์แสงและแสงอินฟราเรด โดยเซ็นเซอร์มีมุมมองประมาณ 150 องศาและมีประสิทธิภาพในระยะประมาณ 0.03 ถึง 0.6 เมตร เหนืออุปกรณ์ ดังรูปที่ 2.22 และ 2.23 ตามลำดับ

รูปที่ 2.22 วิธีการติดตามจะเพิ่มประสิทธิภาพการรับรู้และติดตามมือจากมุมมองของจอภาพแบบสวมศีรษะ



รูปที่ 2.23 มุมมองการควบคุมลีปโมชันด้วยมือในโหมดบนพื้นโต๊ะ

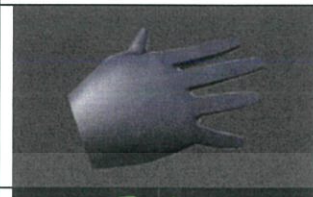
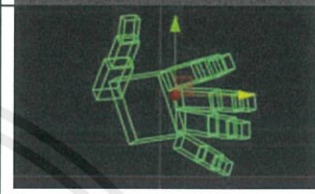
2.4.3 Asset มือ

Asset ของลีปโมชันสำหรับโปรแกรมยูนิตี้จะประกอบด้วย prefab มือที่สามารถใช้งานได้เลย เป็นตัวอย่างในการสร้าง prefab มือขึ้นเอง วิธีในการสร้างมือคือสร้างคอมโพเนนต์ (Component) สำหรับชิ้นส่วนของมือและเคลื่อนไหวชิ้นส่วนเหล่านั้นได้ สร้าง mesh ให้กับชิ้นส่วนกระดูกมือและเปลี่ยนรูป mesh โดยการหมุนกระดูก และเขียนคำสั่งมือเพื่อสร้างกราฟิก ซึ่ง LeapHandController Class จะจัดการสิ่งที่ได้และแอปพลิเคชันการติดตามข้อมูลไปยังมือและนิ้วมือ โดย HandModel Class และ FingerModel Class เป็นคลาสพื้นฐานสำหรับอเนมิชันมือและนิ้วมือ โดย Asset มือแบบต่างๆ แสดงดังตารางที่ 2.3

ตารางที่ 2.3 Asset มือแบบต่างๆ

ชนิด	Prefabs	ที่ตั้ง	ตัวอย่าง
Capsule	CapsuleHand	Core Assets	
Procedural Hands	PolyHand1/2/3	Hand Module	
Rigged Hands	PepperBaseCut, PepperBaseFull	Hands Module	

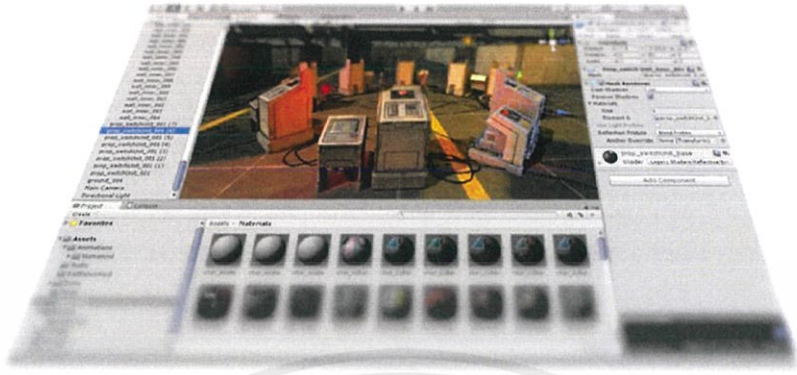
ตารางที่ 2.3 (ต่อ) Asset มือแบบต่างๆ

ชนิด	Prefabs	ที่ตั้ง	ตัวอย่าง
Low Polygon Hands	LoPoly_Rigged_Hand	Hands Module	
Physics Hands	RigidHand, RigidFullHand, RigidRoundHand, ThickRigidHand	Core Assets	
Interaction Hands	BrushHand	Interaction Engine Module	(invisible)
Attachment Hands	HandAttachments	Attachments Module	

2.5 โปรแกรมยูนิตี้[9]

โปรแกรมยูนิตี้ (Unity) เป็นโปรแกรมสร้างเกม สามารถพอร์ทลงบนแพลตฟอร์มได้ในหลาย ๆ ที่ โดยแพลตฟอร์มที่โปรแกรมยูนิตี้รองรับอย่างเป็นทางการ ได้แก่ Web, PC, Mac, iOS, Android, Windows Phone, Blackberry, Xbox และ PlayStation ภาพรวมของโปรแกรมยูนิตี้ดังรูปที่ 2.24 วิธีการทำงานของโปรแกรมยูนิตี้จะแปลงตัวโปรเจค ซึ่งใช้ภาษา C# หรือ JavaScript ให้กลายเป็นเนทีฟของแพลตฟอร์มนั้นๆ โดยใช้คำสั่งผ่าน API ของ Unity ทำให้ผู้ใช้งานสามารถพัฒนางานที่มีความหลากหลาย ลดต้นทุนในการทำงาน เรียนรู้ได้ง่ายและใช้งานไม่ยาก

โปรแกรมยูนิตี้มีร้านค้า Asset สำหรับการซื้อขายโหลด ปลั๊กอิน และ Asset ต่างๆ นอกจากนี้ยังมีเครือข่ายกลุ่มผู้ใช้ที่มีจำนวนมาก ทำให้ลดระยะเวลาแก้ปัญหาต่างๆในชิ้นงานและพัฒนาชิ้นงานที่ต้องการได้ง่ายและสะดวกมากขึ้น

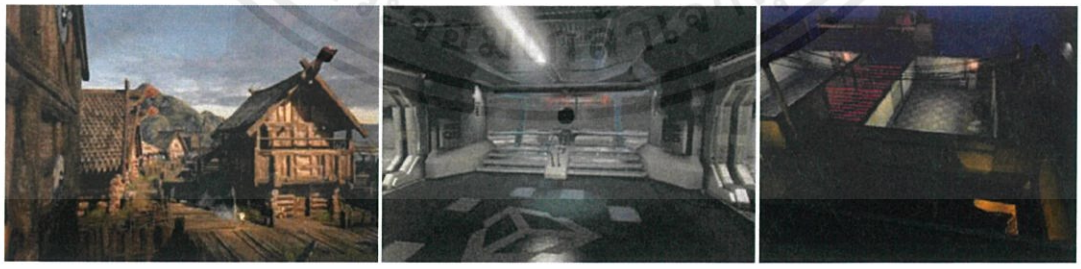


รูปที่ 2.24 ภาพรวมของโปรแกรมยูนิตี้

2.5.1 มิติเกม

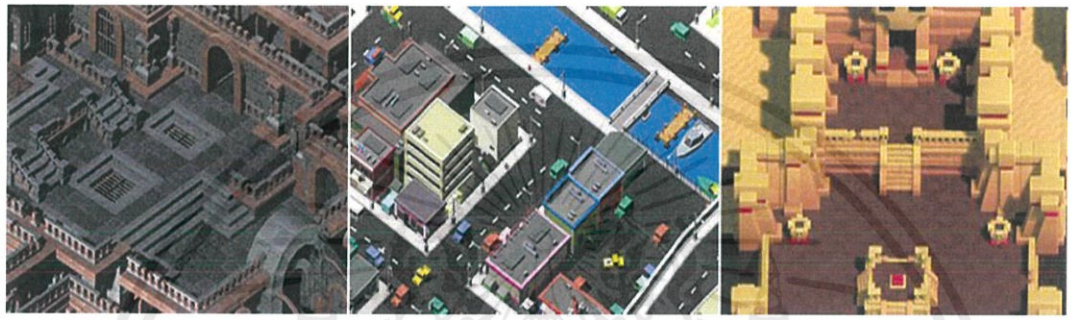
โปรแกรมยูนิตี้เหมาะสำหรับการสร้างเกมทั้ง 2 มิติและ 3 มิติ เมื่อสร้างโปรเจคใหม่ในโปรแกรมยูนิตี้ การเลือกโหมดระหว่าง 2D หรือ 3D จะกำหนดการตั้งค่าบางอย่างสำหรับ Unity Editor เช่น ไม่ว่าภาพจะถูกนำเข้ามาเป็นพื้นผิว (texture) หรือสไปรท์ (sprite) สามารถสลับโหมดระหว่าง 2D หรือ 3D ได้ตลอดเวลาโดยไม่คำนึงถึงโหมดที่ตั้งค่าไว้เมื่อตอนเริ่มสร้างโปรเจค เป็นต้น เกมประเภทต่าง ๆ มีดังนี้

- **เกม 3 มิติ**: มักจะใช้รูปทรงเรขาคณิต 3 มิติด้วยวัสดุและพื้นผิวแสดงบนพื้นผิวของเกมอ็อบเจกต์ (GameObjects) เพื่อให้ปรากฏเป็นสภาพแวดล้อม ตัวละคร และวัตถุที่สร้างโลกของเกม ดังรูปที่ 2.25 กล้องสามารถย้ายได้ทั้งภายในและรอบ ๆ ฉากได้อย่างอิสระด้วยแสงและเงาที่เสมือนจริง เกม 3 มิติ มักแสดงฉากโดยใช้ Perspective ดังนั้นเมื่ออยู่ใกล้กล้อง วัตถุบนฉากจะใหญ่ขึ้น สำหรับเกมทั้งหมดที่ตรงกับคำอธิบายนี้จะเริ่มต้นในโหมด 3D



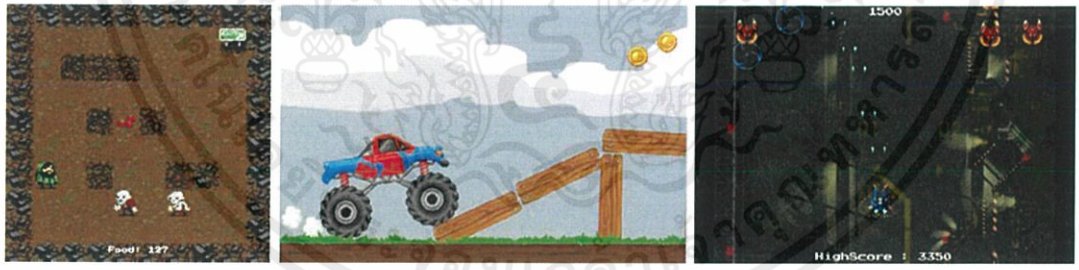
รูปที่ 2.25 ฉาก 3 มิติ

- **Orthographic 3 มิติ**: เกมบางเกมใช้รูปทรงเรขาคณิต 3 มิติ แต่จะใช้กล้อง Orthographic แทน Perspective ซึ่งเป็นเทคนิคทั่วไปที่ใช้ในเกม ให้มุมมองจากมุมสูง และบางครั้งเรียกว่า 2.5 มิติ ดังรูปที่ 2.26 ถ้าสร้างเกมในรูปแบบนี้ควรใช้ Editor ในโหมด 3D ถึงแม้ว่าจะไม่เป็น Perspective แต่จะยังคงสามารถใช้งานด้วย Asset และโมเดล 3 มิติ ฉะนั้นจะต้องเปลี่ยนมุมมองของกล้องและฉากเป็นแบบ Orthographic



รูปที่ 2.26 เกม 3 มิติที่ใช้มุมมอง Orthographic

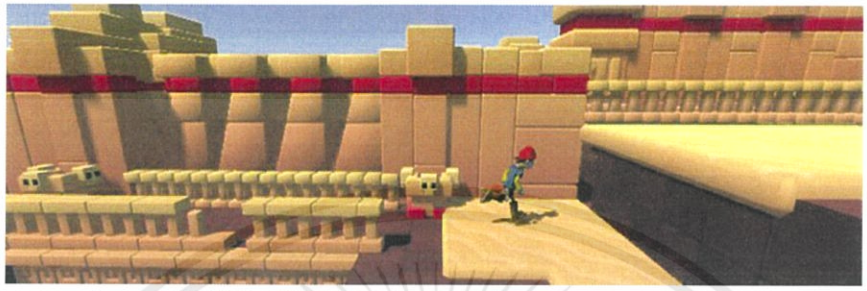
- **เกม 2 มิติ**: หลายเกมใช้กราฟิกแบน (flat graphics) บางครั้งเรียกว่า สไปรท์ ซึ่งไม่มีรูปทรงเรขาคณิต 3 มิติ สร้างเป็นภาพแบน และกล้องของเกมไม่เป็น Perspective ดังรูปที่ 2.27 สำหรับเกมประเภทนี้ควรเริ่มต้น Editor ในโหมด 2D



รูปที่ 2.27 เกม 2 มิติ

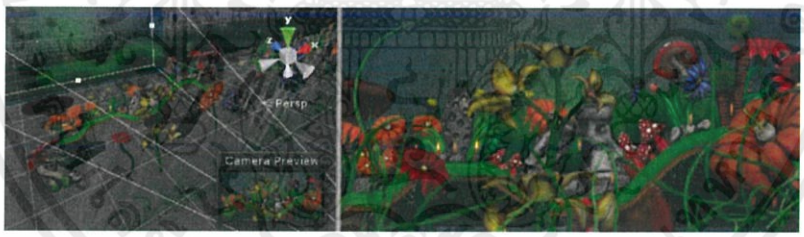
- **เกม 2 มิติ โดยใช้กราฟิก 3 มิติ**: เกม 2 มิติ บางเกมใช้รูปทรงเรขาคณิต 3 มิติสำหรับสร้างสภาพแวดล้อมและตัวละคร แต่จำกัดการเล่นเกมเป็น 2 มิติ ยกตัวอย่างเช่น กล้องอาจแสดงมุมมองที่เลื่อนไปด้านข้าง (side-scrolling) และผู้เล่นสามารถเคลื่อนที่ใน 2 มิติ แต่ตัวเกมยังคงใช้โมเดล 3 มิติสำหรับอุปสรรคในเกมและใช้กล้อง Perspective 3 มิติ ดังรูปที่ 2.28

เกมประเภทนี้บางครั้งเรียกว่า 2.5 มิติ ถึงแม้ว่าการเล่นเกมจะเป็น 2 มิติ แต่ส่วนใหญ่จะสร้างเกมด้วยโมเดล 3 มิติ ดังนั้นควรเริ่มต้น Editor ในโหมด 3D



รูปที่ 2.28 มุมมองของเกมแบบ side-scrolling

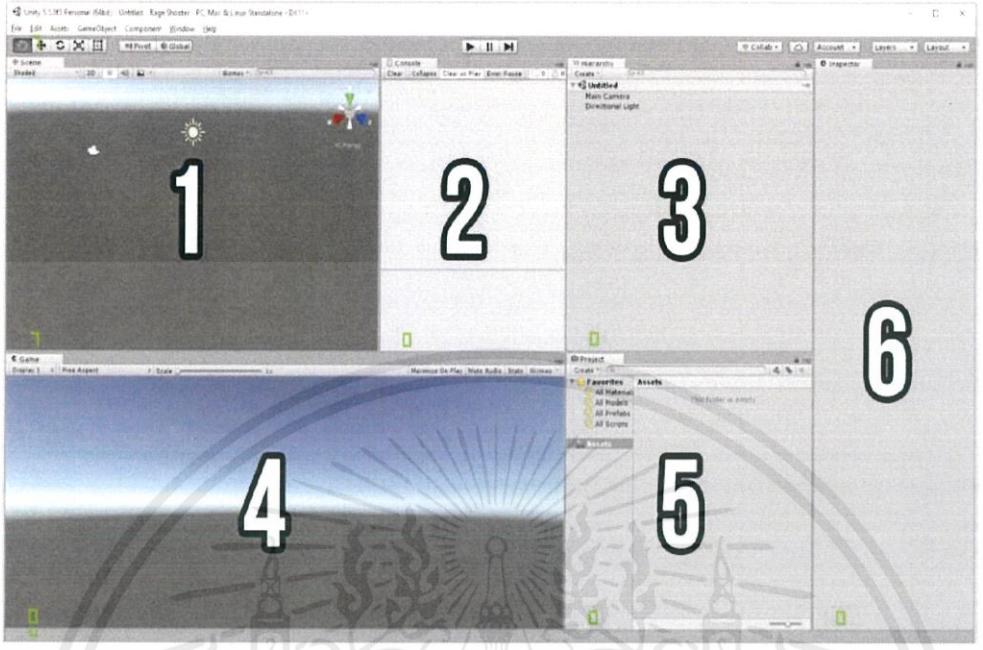
- เกมและกราฟิก 2 มิติ โดยใช้กล้อง Perspective: รูปแบบนี้เป็นอีกรูปแบบหนึ่งของเกม 2 มิติที่นิยมโดยใช้กราฟิก 2 มิติ แต่ใช้มุมมองกล้อง Perspective เพื่อให้เคลื่อนที่แบบ parallax scrolling ซึ่งเป็นฉากแบบ cardboard theater ที่กราฟิกทั้งหมดจะเป็นแบบแบน แต่จัดการที่ระยะทางต่างๆจากกล้อง ดังรูปที่ 2.29 อย่างไรก็ตามควรเปลี่ยนโหมดการฉายกล้องไปที่ Perspective และโหมดมุมมองฉากที่ 3D



รูปที่ 2.29 เกมรูปแบบ cardboard theater 2 มิติ

2.5.2 หน้าต่างการทำงาน

หน้าต่างการทำงานของโปรแกรมยูนิตี้แบ่งได้เป็นหลายส่วนย่อย ดังรูปที่ 2.30 ซึ่งหน้าต่างหลักมีดังนี้



รูปที่ 2.30 หน้าต่างการทำงานของโปรแกรมยูนิตี้

1. หน้าต่าง Scene เป็นหน้าต่างที่บ่งบอกว่าในฉากที่กำลังทำงานมีวัตถุอะไรบ้าง และสามารถจัดการวัตถุต่างๆได้ เช่น กล้อง แสง เอฟเฟกต์ วัตถุในเกมหรือโมเดล 3 มิติ สามารถเห็นภาพโดยรวมของเกมและการขยับกล้องในโลกนี้ ซึ่งจะมีแต่ผู้พัฒนาเท่านั้นที่เห็น ผู้เล่นของเกมจะไม่สามารถมองเห็นได้

2. หน้าต่าง Console ส่วนนี้เปรียบเสมือนหน้าต่างบอกข้อมูลแก่ผู้พัฒนา หากตัวเกมมีความผิดพลาดอะไรเกิดขึ้น จะแสดงผลขึ้นมาในหน้าต่างนี้ รวมถึงการใช้คำสั่ง print(); หรือ Debug.Log(); ก็จะโชว์ในหน้าต่างนี้เช่นกัน

3. หน้าต่าง Hierarchy หน้าต่างนี้จะแสดงรายการเกมอ็อบเจกต์ของหน้าต่าง Scene ที่กำลังเปิดอยู่ มีหน้าที่บอกว่าในฉากนั้นมีเกมอ็อบเจกต์อะไรอยู่บ้าง

4. หน้าต่าง Game เป็นส่วนที่แสดงการทำงานของเกมในฉาก ทำให้มองเห็นภาพสถานะและการทำงานของวัตถุต่างๆภายในฉากที่สร้างขึ้น

5. หน้าต่าง Project เป็นส่วนที่ใช้ในการเก็บ Asset ต่างๆก่อนนำไปสร้างเกม เช่น สคริปต์ต่างๆที่ใช้กำหนดควบคุมตัวเกม โมเดล 3D ที่ใช้เป็นตัวละครหรือวัตถุต่างๆในเกม พื้นผิว ไฟล์เสียงหรือวีดีโอ วัตถุต้นแบบ (Prefab) และอื่นๆ

6. หน้าต่าง Inspector หน้าต่างนี้จะแสดงผลคุณสมบัติต่างๆของเกมอ็อบเจกต์ที่กำลังจะเลือกอยู่ ไม่ว่าจะเป็นตำแหน่ง ขนาด ทิศทาง รวมถึงข้อมูลและฟังค์ชั่นต่างๆที่เขียนโค๊ดเพิ่มเข้าไปด้วย

2.5.3 การเขียนสคริปต์

พฤติกรรมของเกมอ็อบเจกต์ถูกควบคุมโดยคอมโพเนนต์ (Component) ที่แนบไปกับวัตถุ และสามารถสร้างคอมโพเนนต์ได้เองโดยใช้สคริปต์ (scripts) ซึ่งช่วยให้สามารถทริกเกอร์ (trigger) สถานะในเกม ปรับเปลี่ยนคุณสมบัติของคอมโพเนนต์เมื่อเวลาผ่านไป และตอบสนองต่ออินพุตของผู้ใช้งานในแบบต้องการ โปรแกรมยูนิตี้รองรับ 2 ภาษาโปรแกรม นั่นคือ ภาษา C# และ UnityScript ซึ่งเป็นภาษาที่ออกแบบมาเพื่อใช้กับโปรแกรมยูนิตี้โดยเฉพาะ และพัฒนามาจาก JavaScript นอกจากนี้ภาษาอื่นๆอย่างภาษา .NET นั้นสามารถนำมาใช้กับโปรแกรมยูนิตี้ได้ถ้าสามารถคอมไพล์ DLL ที่เข้ากันได้ สคริปต์เริ่มต้นจะมีลักษณะดังรูปที่ 2.31 สคริปต์จะเชื่อมต่อกับการทำงานภายในโปรแกรมโดยคลาสภายในที่เรียกว่า MonoBehaviour ซึ่งเป็นชนิดของพิมพ์เขียวสำหรับการสร้างประเภทของคอมโพเนนต์ที่สามารถแนบไปกับเกมอ็อบเจกต์ ทุกครั้งที่แนบสคริปต์ของคอมโพเนนต์ไปกับเกมอ็อบเจกต์ จะสร้างชื่อที่ประกาศในการเรียกใช้วัตถุใหม่ซึ่งกำหนดโดยแม่แบบ ชื่อคลาสนำมาจากชื่อไฟล์ที่สร้าง ชื่อคลาสและชื่อไฟล์จะต้องเหมือนกันเพื่อเปิดใช้งานสคริปต์ของคอมโพเนนต์ที่จะแนบไปกับเกมอ็อบเจกต์

```
using UnityEngine;
                    using System.Collections;

public Class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {
    }

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

รูปที่ 2.31 สคริปต์เริ่มต้น

มีสองฟังก์ชั่นที่กำหนดไว้ภายในคลาส คือฟังก์ชัน Update เป็นที่ใส่โค้ดที่จะปรับปรุงและแก้ไขเฟรมให้กับเกมอ็อบเจกต์ ซึ่งอาจรวมถึงการเคลื่อนไหว การทริกเกอร์และการตอบสนองต่ออินพุตของผู้ใช้หรืออื่นๆ เปิดใช้งานฟังก์ชัน Update เพื่อตั้งค่าตัวแปร อ่านการตั้งค่าและสร้างการเชื่อมต่อกับเกมอ็อบเจกต์อื่น ๆ ก่อนที่จะใส่การกระทำในเกม และฟังก์ชั่น Start จะถูกเรียกโดยโปรแกรมก่อนที่เกมจะเล่น นั่นคือก่อนที่ฟังก์ชัน Update จะถูกเรียกเป็นครั้งแรก และเป็นที่ที่เหมาะสำหรับการเริ่มต้นใด ๆ สคริปต์ UnityScript ทำงานแตกต่างกับสคริปต์ C# เล็กน้อย ฟังก์ชั่น Start และ Update มีความหมายเดียวกัน แต่ไม่ได้ประกาศคลาสอย่างชัดเจน สคริปต์จะถือว่าเป็นการกำหนดคลาสจาก MonoBehaviour และใช้ชื่อจากชื่อไฟล์ของสคริปต์ใน Asset

2.6 MIDI[14]

MIDI หรือ มาตรฐานการประสานเครื่องดนตรีแบบดิจิทัล (Music Instrument Digital Interface) เป็นโพรโทคอลมาตรฐาน โดยเป็นระบบการติดต่อสื่อสารทางดนตรีของอุปกรณ์อิเล็กทรอนิกส์ทางดนตรี เช่น คอมพิวเตอร์ ซินธิไซเซอร์ ซีเควนเซอร์ ซาวด์โมดูล แซมเพลอร์ เป็นต้น ซึ่งใช้สัญญาณไฟฟ้าแบบดิจิทัลในการส่งข้อมูลระหว่างอุปกรณ์ที่เชื่อมต่อ ข้อมูลทั้งหมดจะอยู่ในรูปของคำสั่งที่จะไปสั่งเครื่องดนตรีว่าให้เปล่งเสียงโน้ตตัวใด (Note ON), ระดับความดัง (Velocity) และคำสั่งอื่นๆ ตามคุณสมบัติเฉพาะของเครื่องดนตรีแต่ละชนิด ด้วยเหตุที่เป็นไฟล์คำสั่งแบบดิจิทัล ทำให้ไฟล์ MIDI มีขนาดที่เล็กมาก และนักคอมพิวเตอร์สามารถนำข้อมูลดิจิทัลนี้มาพัฒนาทำให้ทั้งคอมพิวเตอร์และเครื่องดนตรีสามารถสื่อสารกันได้อย่างสมบูรณ์โดยผ่านระบบมิดิ

มาตรฐานเกี่ยวกับการใช้ชุดคำสั่ง MIDI เรียกว่า General MIDI (GM) ประกอบด้วยสาระสำคัญเรื่องการกำหนดมาตรฐานขั้นต่ำของการรองรับคำสั่ง MIDI เช่น ต้องรองรับการปรับความดังตัวโน้ต และต้องเล่นได้อย่างน้อย 24 โน้ตพร้อมกัน เป็นต้น รวมทั้งกำหนดเสียงเครื่องดนตรีที่ใช้ใน MIDI ทั้งหมด 128 ชนิด ซึ่งจะรวมเสียงเอฟเฟกต์ต่างๆด้วย โดยจะมีการแบ่งกลุ่มออกเป็น 16 กลุ่ม เรียกว่า แพ็ตช์ (Patch) ดังตารางที่ 2.4 โดยในปริญญานิพนธ์ใช้หมายเลข 1 ซึ่งคือแพ็ตช์ของเปียโนและค่าพารามิเตอร์ MIDI ของเสียงเปียโนที่ใช้ในการกำหนดให้เครื่องสังเคราะห์เสียงเปล่งเสียงเป็นเสียงใด แสดงดังตารางที่ 2.5 ซึ่งการใช้งานจะกล่าวถึงในบทถัดไป

ตารางที่ 2.4 แพ็ตช์ของเครื่องดนตรีใน MIDI

หมายเลข	ชื่อแพ็ตช์	หมายเลข	ชื่อแพ็ตช์
1	เปียโน (Piano)	9	เครื่องลมไม้ (Reed)
2	เครื่องตีกระทบโครมาติก (Chromatic Percussion)	10	เครื่องเป่าลมไม้ (Pipe)
3	ออร์แกน (Organ)	11	ดนตรีนำสังเคราะห์ (Synth Lead)
4	กีตาร์ (Guitar)	12	แป้นสังเคราะห์ (Synth Pad)
5	เบส (Bass)	13	เอฟเฟกต์สังเคราะห์ (Synth Effects)
6	เครื่องสาย (Strings)	14	เครื่องดนตรีพื้นเมือง (Ethnic)
7	วงดนตรี (Ensemble)	15	เครื่องตีกระทบ (Percussive)
8	เครื่องเป่าทองเหลือง (Brass)	16	เสียงเอฟเฟกต์ (Sound Effects)

ตารางที่ 2.5 ค่าพารามิเตอร์ในตาราง MIDI ของเปียโน
เป็นตัวกำหนดให้เครื่องสังเคราะห์เสียงเปล่งเสียงได้ตามที่ต้องการ

Octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107

2.6.1 เสียงแบบดิจิทัล (Digital Audio)

สัญญาณเสียงที่ส่งมาจากไมโครโฟน เครื่องสังเคราะห์เสียง เครื่องเล่นเทป หรือจากแหล่งกำเนิดเสียงต่างๆ ทั้งจากธรรมชาติ และที่สร้างขึ้น แล้วนำข้อมูลที่ได้แปลงเป็นสัญญาณดิจิทัล ซึ่งข้อมูลจะถูกสุ่มให้อยู่ในรูปแบบของบิต และไบต์ โดยเรียกอัตราการสุ่มข้อมูลที่ได้มา เรียกว่า " Sampling Rate " และจำนวนของข้อมูลที่ได้เรียกว่า "Sampling Size" ซึ่งจะเป็นตัวกำหนดคุณภาพของเสียงที่ได้จากการเล่นเสียงแบบดิจิทัล

การเก็บข้อมูลเสียงแบบดิจิทัลจะมีขนาดข้อมูลใหญ่ ทำให้ต้องใช้หน่วยความจำและทรัพยากรบนหน่วยประมวลผลกลางมากกว่า MIDI แต่จะแสดงผลได้หลากหลายและเป็นธรรมชาติกว่า MIDI มาก เสียงแบบดิจิทัลที่พบบ่อย จะอยู่ช่วงความถี่ 44.1 kHz, 22.05 kHz และ 11.023 kHz ซึ่งมี Sampling Size เป็น 8 บิต และ 16 บิต โดยที่ Sampling Rate และ Sampling Size ที่สูงกว่าจะให้คุณภาพของเสียงที่ดีกว่า

ตารางที่ 2.6 แสดงอัตราการสุ่มข้อมูล Sampling Rate

Sampling Rate (kHz)	Sampling Size (bit)	Stereo หรือ Mono	จำนวน Byte ที่ใช้ 1 วินาที
44.10	16	Stereo	8.5 MB
44.10	16	Mono	5.25 MB
44.10	8	Stereo	5.25 MB
44.10	8	Mono	2.6 MB
22.05	16	Stereo	5.25 MB
22.05	16	Mono	2.5 MB
22.05	8	Stereo	2.6 MB
22.05	8	Mono	1.3 MB
11.03	8	Stereo	1.3 MB
11.03	8	Mono	650 MB

2.7 ซินธิไซเซอร์

เครื่องสังเคราะห์เสียง หรือ ซินธิไซเซอร์ (synthesizer) คือ เครื่องดนตรีอิเล็กทรอนิกส์ที่มีประสิทธิภาพในการผลิตเสียง โดยทำให้เกิดสัญญาณไฟฟ้าที่แตกต่างกัน ออกแบบมาเพื่อสร้างเสียงจำลองโดยใช้เทคนิคต่างๆ เช่น การเพิ่มเสียง, การลดเสียง, การสังเคราะห์เสียงกายภาพ หรือการทำให้คลื่นเสียงผิดเพี้ยนรูปร่างไป เป็นต้น ซินธิไซเซอร์สามารถผลิตช่วงเสียงได้ค่อนข้างกว้างและยังสามารถลอกเลียนเสียงเครื่องดนตรีอื่น ๆ หรือสามารถทำให้เกิดลักษณะของเสียงแบบใหม่ได้ ซินธิไซเซอร์มักจะถูกควบคุมในรูปแบบของคีย์บอร์ด นอกจากนั้นยังมีตัวควบคุม (Controller) อีกหลายรูปแบบที่ถูกคิดขึ้นมาใหม่ และซินธิไซเซอร์ที่ไม่ต้องมีตัวควบคุมมักจะถูกเรียกว่าโมดูล (modules) ซึ่งถูกควบคุมโดยวิธีใช้ MIDI

ในการควบคุม MIDI ซินธิไซเซอร์กลายมาเป็นสิ่งที่ง่ายในการรวมเสียงและเข้าจังหวะเสียงกับเครื่องดนตรีอิเล็กทรอนิกส์และอุปกรณ์ควบคุมชนิดอื่นๆ การต่อ MIDI แพร่หลายเกือบจะในทุกอุปกรณ์ดนตรี และยังใช้เป็นส่วนประกอบพื้นฐานของคอมพิวเตอร์ส่วนบุคคล (PCs) ด้วย โดยมาตรฐานซอฟต์แวร์ General MIDI ถูกประดิษฐ์ขึ้น เพื่อรองรับวิธีที่ตรงกันในการอธิบายเสียงสูงต่ำกว่า 200 ชุด รวมไปถึงเสียงเคาะด้วย สามารถที่จะใช้กับคอมพิวเตอร์ส่วนบุคคล สำหรับเสียงโน้ตดนตรีรูปแบบไฟล์ MIDI เป็นที่แพร่หลายและกลายเป็นมาตรฐานที่เป็นที่นิยมใช้สำหรับการแปลงโน้ตเสียงของคอมพิวเตอร์

2.8 การสังเคราะห์เสียงโดยใช้คอมพิวเตอร์

การสังเคราะห์เสียง (sound synthesis) นั้นถูกจำแนกไว้หลากหลายวิธีตามวิธีการสังเคราะห์ไม่ว่าจะเป็นการสร้างเสียงตัวโน้ตจากการรวมคลื่นไซน์ (sine wave) หลากหลายความถี่ [21] หรือการสร้างเสียงตัวโน้ตจากเทคนิคการกรองด้วยตัวกรองสัญญาณ [22] ซึ่งมีรายละเอียดและเทคนิคที่แตกต่างกันไป [20] การสังเคราะห์เสียงที่นิยมใช้สำหรับการสังเคราะห์เสียงเปียโนที่สมจริงนั้นเป็นการสังเคราะห์รูปคลื่นเสียงที่มาจากโมเดลทางฟิสิกส์ซึ่งพิจารณาองค์ประกอบทางกายภาพของวัตถุเช่นสายและฆ้อนของเปียโนรวมถึงสภาพแวดล้อมจนสามารถโมเดลคลื่นเสียงด้วยการแก้สมการคลื่นที่ยุ่งยากซับซ้อน แต่มีข้อดีคือสามารถสร้างเสียงเปียโนที่เหมือนจริงมาก ทั้งนี้ก็ขึ้นอยู่กับโมเดลทางฟิสิกส์ที่เลือกใช้ ว่าจะพิจารณาองค์ประกอบของการเกิดเสียงเปียโนได้ครบถ้วนหรือไม่ ตัวอย่างของการสังเคราะห์เสียงด้วยการวิเคราะห์โมเดลทางฟิสิกส์ [22] การสังเคราะห์เสียงเปียโนที่นิยมทำกันอีกแบบหนึ่ง โดยเฉพาะที่ใช้กับ MIDI นั้นคือการสังเคราะห์เสียงโดยใช้เสียงตัวอย่าง (sample-based synthesis)

หรือ sampler [19,23] การสังเคราะห์เสียงในลักษณะนี้เป็นที่นิยมค่อนข้างมากเนื่องจากง่ายกว่าวิธีการก่อนหน้า และให้เสียงที่ไพเราะสมจริง

การสังเคราะห์เสียงโดยใช้เสียงตัวอย่างนั้น เป็นการนำเสียงของโน้ตเครื่องดนตรีต้นฉบับที่บันทึกเอาไว้มาทำการแก้ไขดัดแปลงด้วยวิธีการทางการประมวณสัญญาณ แทนที่จะใช้วิธีการกำเนิดเสียงจากรูปคลื่นไซน์ หรือพิจารณาโมเดลทางกายภาพ จึงทำให้ง่ายและประหยัดเวลาในการประมวลผลจึงเป็นที่นิยมโดยเฉพาะการสังเคราะห์เสียงจากไฟล์ MIDI นั้นนิยมใช้วิธีนี้ในการสังเคราะห์เสียง เสียงตัวอย่างในกระบวนการสังเคราะห์เสียงโดยใช้เสียงตัวอย่างอาจถูกเรียกว่า sound font [24] ซึ่งเป็นชื่อทางการค้า และอาจคุ้นหูผู้คนทั่วไปมากกว่า หากทำการเปลี่ยนฟอนต์เสียง หรือ sound font จะทำให้รายละเอียดของเสียงตัวโน้ตเปลี่ยนไปด้วยคล้ายกับการเปลี่ยนฟอนต์ของตัวอักษรในโปรแกรมพิมพ์เอกสาร การรวมฟอนต์เสียงเข้าด้วยกันหลาย ๆ ตัวโน้ต จะเรียกว่าธนาคารเสียง หรือ soundbank ซึ่งธนาคารเสียงที่มีคุณภาพสูงจะทำให้เสียงดนตรีที่บรรเลงมีความไพเราะมากกว่าและนั่นรวมถึงราคาของมันด้วย ท่านสามารถฟังเปรียบเทียบระหว่างเสียงเมื่อใช้ธนาคารเสียงคุณภาพปกติ กับคุณภาพสูงได้ใน [25]

ในปริญญานิพนธ์นี้ได้ใช้ตัวสังเคราะห์เสียงชื่อว่า C# Synth [26] ในการเล่นเสียงตัวโน้ตของเครื่องดนตรีเปียโน ซึ่ง C# Synth เป็นโปรแกรมสังเคราะห์เสียงที่อยู่ในรูปของ C# ไลบรารี่ ซึ่งใช้วิธีการสังเคราะห์เสียงแบบใช้ตัวอย่างเสียงดังที่ได้กล่าวไปแล้วใน [25] เองก็เป็นการนำ C# Synth มาใช้เพื่อสร้างเสียง MIDI ซึ่งคุณภาพของเสียงนั้นขึ้นอยู่กับธนาคารเสียงที่เลือกใช้ การใช้งาน C# Synth เป็นไลบรารี่ จะกล่าวถึงในบทถัดไป

บทที่ 3

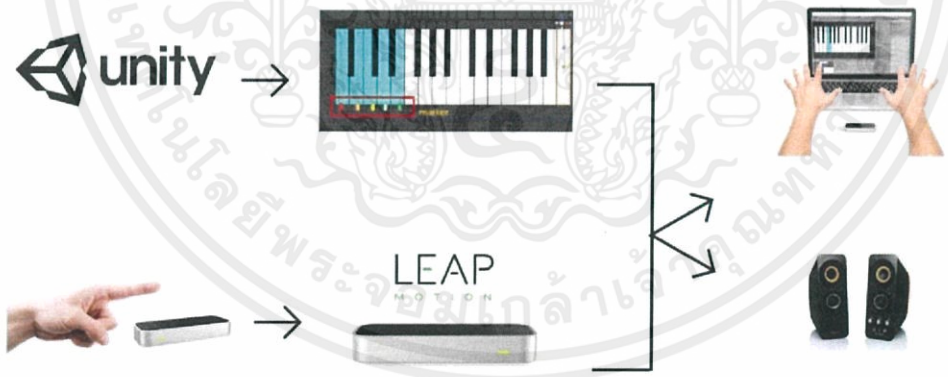
การออกแบบและการจัดทำปริญญานิพนธ์

ปริญญานิพนธ์นี้แบ่งการทำงานออกเป็นสองส่วน ซึ่งส่วนแรกเป็นส่วนที่เกี่ยวกับการสร้างโมเดลเปียโน ทำการออกแบบโปรแกรมจำลองนี้โดยใช้โปรแกรมยูนิตี้ ซึ่งเป็นโปรแกรมที่ใช้ในการสร้างและพัฒนาเกม และส่วนที่สองเป็นการนำอุปกรณ์ลีปโมชันมาประยุกต์ใช้กับโปรแกรมจำลองเปียโน โดยอุปกรณ์นี้มีคุณสมบัติในการตรวจจับมือและวัตถุที่มีลักษณะคล้ายนิ้วมือ

3.1 การออกแบบ

3.1.1 ภาพรวมของปริญญานิพนธ์

ปริญญานิพนธ์นี้ศึกษาและออกแบบเปียโนบนอากาศโดยอุปกรณ์ลีปโมชัน ซึ่งเป็นอุปกรณ์ที่มีความสามารถในการตรวจจับมือและนิ้วมือ อุปกรณ์ลีปโมชันสามารถทำงานได้โดยการเชื่อมต่ออุปกรณ์เข้ากับเครื่องคอมพิวเตอร์ [4] เราเลือกใช้โปรแกรมยูนิตี้ [9] ในการออกแบบและพัฒนาเปียโนบนอากาศ ร่วมกับอุปกรณ์ลีปโมชัน โดยใช้ภาษา C#



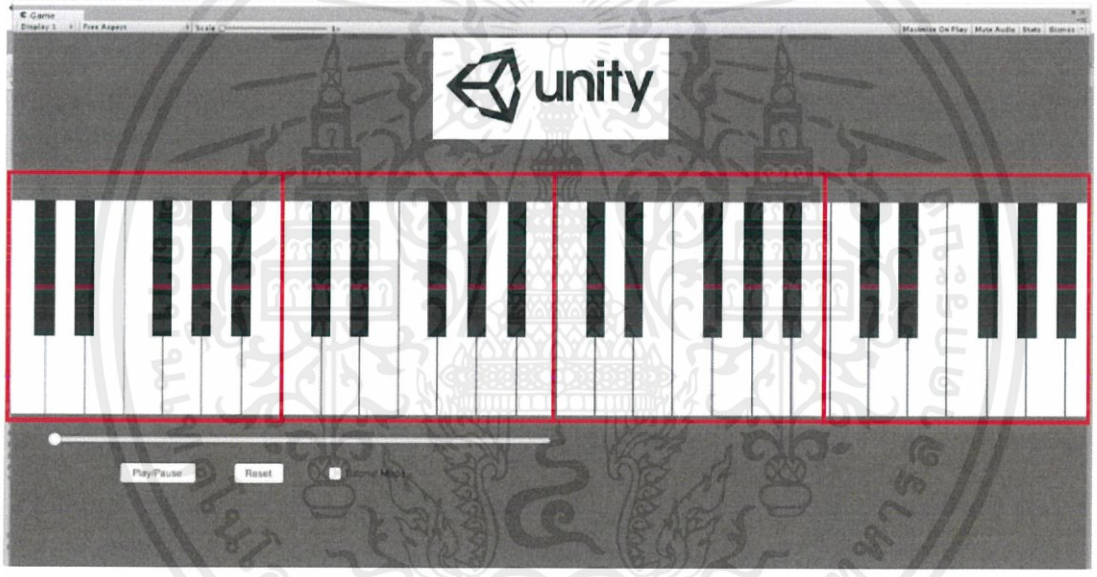
รูปที่ 3.1 ภาพรวมของปริญญานิพนธ์

จากรูปที่ 3.1 ปริญญานิพนธ์นี้ทำการออกแบบโมเดลเปียโนจำลองโดยใช้โปรแกรมยูนิตี้โดยมีอินพุตคือนิ้วมือของผู้ใช้งาน ลีปโมชันมีกระบวนการประมวลผลภาพจากกล้องอินฟาเรดภายในอุปกรณ์ สามารถตรวจจับการเคลื่อนไหวและแสดงตำแหน่งของนิ้วผ่านทางจอแสดงผล

เมื่อมีการขยับมือ หรือ นิ้วมือ ตามเงื่อนไขที่กำหนดไว้ภายในโปรแกรม จะมีการเปล่งเสียงเปียโนผ่านทางลำโพงเป็นเอาต์พุต

3.1.2 การออกแบบและสร้างโมเดลเปียโน

การออกแบบโมเดลเปียโนไม่ได้มีความซับซ้อนมากสามารถทำได้ง่ายๆ บนโปรแกรมยูนิตี้ รูปที่ 3.2 เป็นตัวอย่างการออกแบบโมเดลเปียโนจำนวน 4 อ็อกเทฟ (octave) การออกแบบทำได้โดยการสร้างวัตถุเกมหรือเกมออปเจกต์ (Game object) 3 มิติ เช่น คีย์เปียโน 1 คีย์ก็คือเกมออปเจกต์ 1 ออปเจกต์ ทำการสร้างคีย์เปียโนตามจำนวนและขนาดที่ต้องการ จากนั้นจัดเรียงตำแหน่งให้เหมาะสม และเพิ่มสีให้กับออปเจกต์นั้นๆ



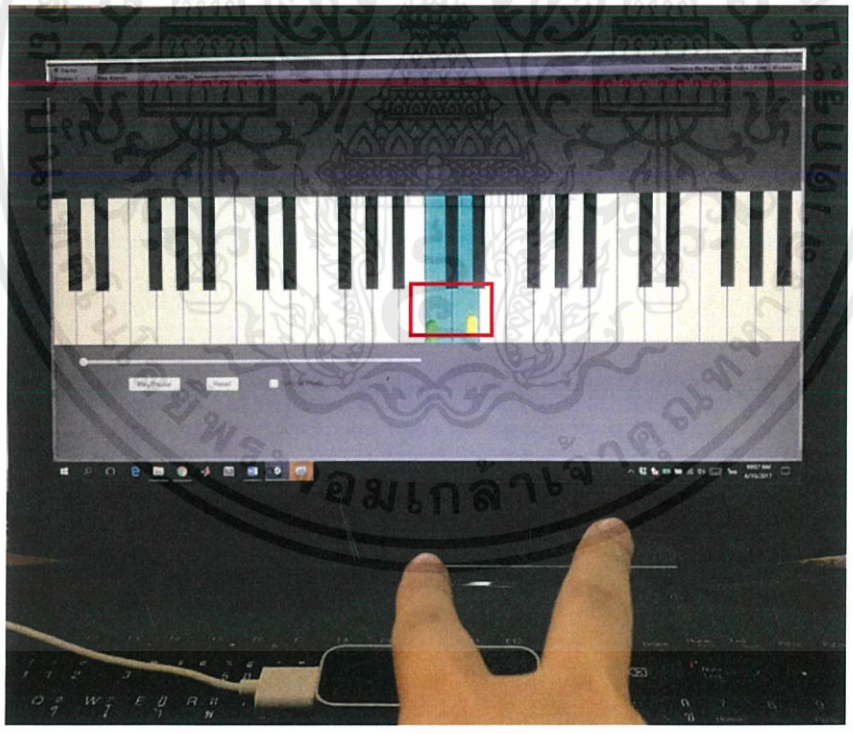
รูปที่ 3.2 โมเดลเปียโนจำนวน 4 อ็อกเทฟที่ถูกออกแบบบนโปรแกรมยูนิตี้

3.1.3 มาร์กเกอร์สำหรับแสดงตำแหน่งปลายนิ้วมือ

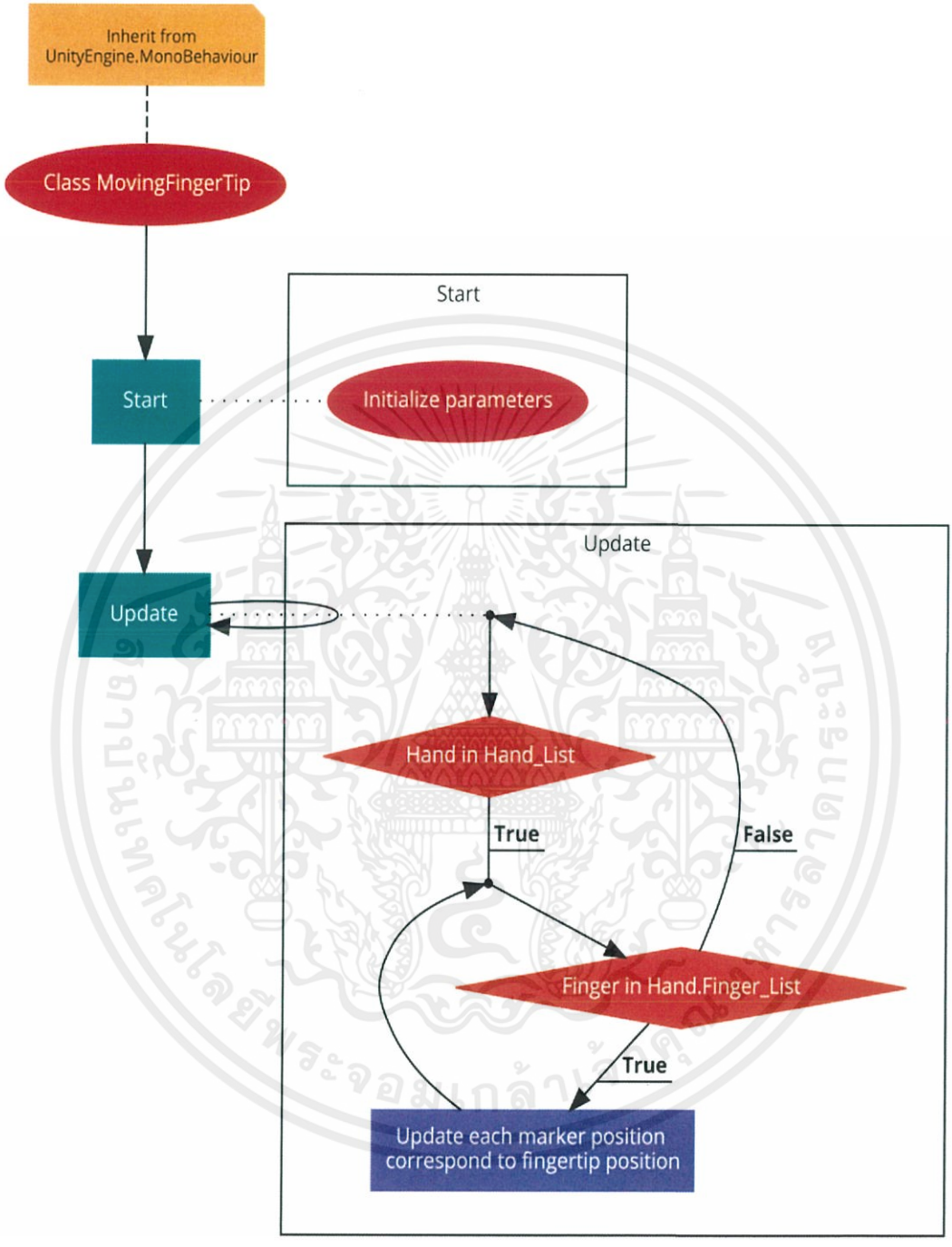
มาร์กเกอร์ (Marker) หรือตัวแสดงตำแหน่งปลายนิ้วมือนั้นสำคัญกับผู้ใช้งาน เนื่องจากเป็นตัวบอกว่าขณะนั้นผู้เล่นวางตำแหน่งนิ้วมือแต่ละนิ้วของผู้เล่นไว้บริเวณคีย์ใด นิ้วใดและยังสามารถแสดงทิศทางที่นิ้วนั้น ๆ กำลังเคลื่อนไหวได้ ทำให้ผู้เล่นสามารถประมาณการวางนิ้วมือบนคีย์ได้ง่ายขึ้น

มาร์กเกอร์สามารถสร้างได้จากเกมออปเจกต์ ในที่นี้จะใช้ออปเจกต์ 3D รูปร่างแคปซูล (capsule) ดังรูปที่ 3.3 ซึ่งทำการออกแบบทั้งหมด 10 แคปซูล เพื่อแทนนิ้วมือทั้ง 10 นิ้ว

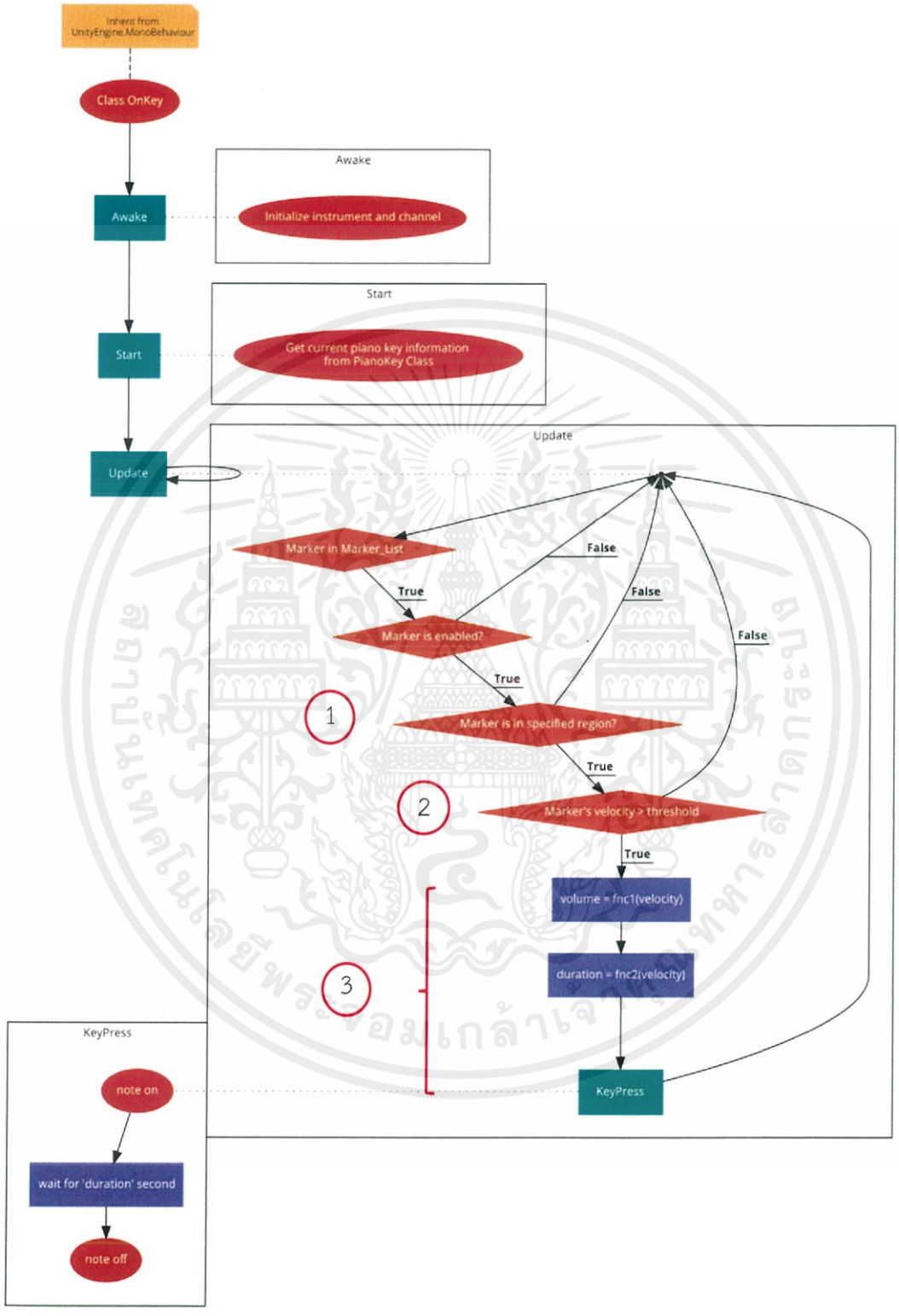
อัลกอริธึมการทำงานของมาร์เกอร์แสดงในรูปที่ 3.4 ในฟังก์ชัน update() ซึ่งเป็นลูปของการแสดงผลในโปรแกรมยูนิตี แสดงการเข้าถึงตำแหน่งปลายนิ้วมือของผู้ใช้โดยอุปกรณ์ลีปโมชัน โดยใช้เงื่อนไขในการตรวจสอบหากมีมือเข้ามาอยู่ในเฟรมของอุปกรณ์ (Hand in Hand_List ในรูป 3.4) ให้ทำคำสั่งต่อไป นั่นคือเข้าถึงออปเจกต์มือทีละข้าง ในออปเจกต์มือแต่ละข้างประกอบไปด้วยลิสต์ (list) ของออปเจกต์นิ้วมือทั้งห้านิ้ว การเข้าถึงออปเจกต์นิ้วมือแต่ละนิ้วอาศัยการวนลูปโดยใช้เงื่อนไขหากตรวจสอบเจอนิ้วมือในลิสต์ (Finger in Hand.Finger_List ในรูป 3.4) โปรแกรมจะทำการวนลูปและแสดงออปเจกต์ของนิ้วมือแต่ละนิ้ว หากตรวจเจอนิ้วมือก็ให้ทำการอัปเดตตำแหน่งของมาร์กเกอร์ให้สอดคล้องกับตำแหน่งของปลายนิ้วมือ ผลลัพธ์คือมาร์กเกอร์มีการขยับตามตำแหน่งของปลายนิ้วมือ เพื่อให้ผู้เล่นทราบถึงตำแหน่งนิ้วมือบนคีย์เปียโน



รูปที่ 3.3 ตัวอย่างมาร์กเกอร์เมื่อจำนวนนิ้วมือเท่ากับ 2 นิ้ว



รูปที่ 3.4 อัลกอลิทึมการทำงานของมาร์กเกอร์



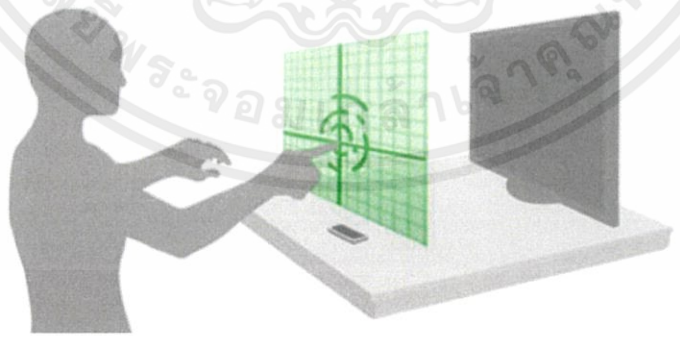
รูปที่ 3.5 อัลกอลิทึมการตรวจจับมาร์กเกอร์

ในรูปที่ 3.5 คลาส Onkey แสดงอัลกอริธึมสำหรับตรวจจับมาร์กเกอร์ มาร์กเกอร์นั้นเป็นวัตถุในเกมหรือเกมออปเจกต์ ซึ่งมีลักษณะคล้ายวัตถุบนโลกแห่งความจริง คือ สามารถกำหนดคุณสมบัติ ขนาดวัตถุ ตำแหน่ง น้ำหนักได้ ตำแหน่งของมาร์กเกอร์นั้นถูกกำหนดตามตำแหน่งของปลายนิ้วมือแต่ละนิ้ว และตำแหน่งของมาร์กเกอร์จะแต่ละมาร์กเกอร์จะถูกตรวจสอบว่าตรงตามเงื่อนไขที่ได้กำหนดเอาไว้ เพื่อที่จะทำคำสั่งต่อไป

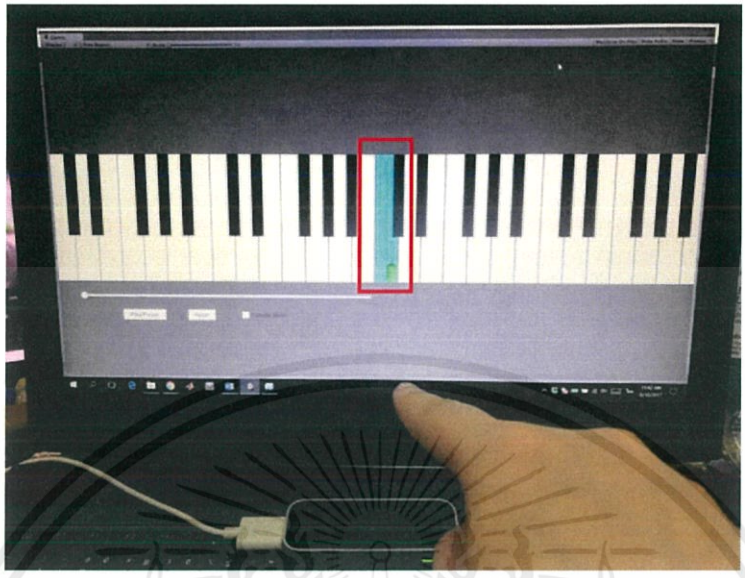
ในฟังก์ชัน update() จะใช้การวนลูปในการค้นหามาร์กเกอร์ทุกอันในลิสต์ (Marker in Marker_List) จากนั้นตรวจสอบว่าแต่ละมาร์กเกอร์เปิดใช้อยู่หรือเปล่า (Marker is enable?) ถ้าเปิดใช้งานอยู่หมายความว่า นิ้วมือของผู้ใช้อยู่ในบริเวณกล้องของอุปกรณ์ลีปโมชัน จากนั้นทำการตรวจสอบตำแหน่งของมาร์กเกอร์ว่าอยู่ในบริเวณที่กำหนดไว้หรือไม่ (Marker is in specified region?) นั่นคือตรวจสอบว่ามาร์กเกอร์อยู่บริเวณบนคีย์ใดๆ หากเข้าเงื่อนไข คีย์นั้นๆจะเปลี่ยนสถานะเป็นสถานะพร้อมกด รออินพุตจากผู้ใช้ คือความเร็วของมาร์กเกอร์ หรือความเร็วปลายนิ้วของผู้ใช้หากเข้าเงื่อนไข (Marker's velocity > threshold) ก็จะใช้คำสั่งในการเล่นเสียงของคีย์นั้นๆ

3.1.4 สถานะของคีย์

3.1.4.1) สถานะพร้อมกด สีของคีย์ถูกกำหนดให้เป็นสีฟ้ากระบวนการอยู่ตำแหน่งที่ 1 ของรูป 3.5 เมื่อมาร์กเกอร์ที่แทนตำแหน่งปลายนิ้วมืออยู่บนตำแหน่งใดๆ ในขอบเขตของคีย์ออปเจกต์ที่กำหนดไว้ในโปรแกรมยูนิตี้ คีย์เหล่านั้นจะเปลี่ยนเป็นสีฟ้าเพื่อบอกว่าเตรียมพร้อมที่จะให้ผู้เล่นกด ดังรูปที่ 3.7



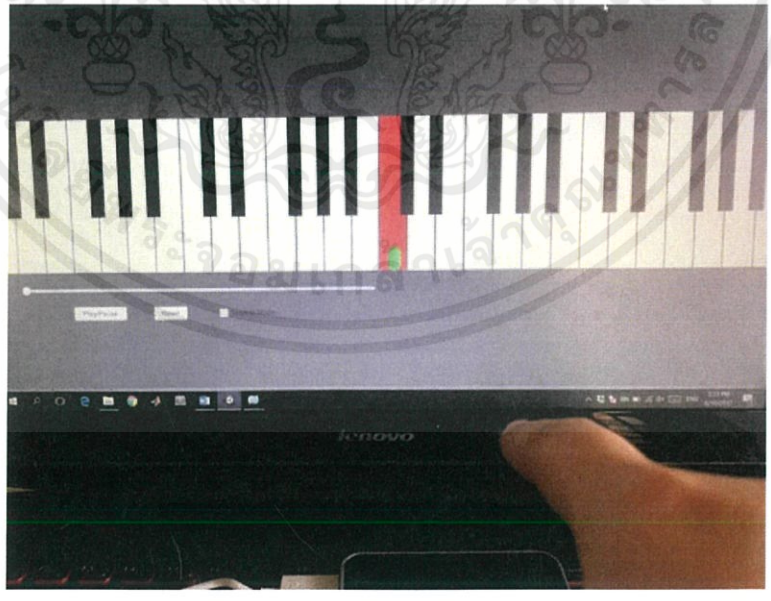
รูปที่ 3.6 ขอบเขตโดยประมาณของลีปโมชันในแนวแกน x และแกน y



รูปที่ 3.7 ตัวอย่างสถานะของคีย์ที่พร้อมจะกด

3.1.4.2) สถานะของคีย์ที่ถูกกดแล้ว

โดยสถานะของคีย์ที่ถูกกดแล้วจะถูกกำหนดให้เป็นสีแดงกระบวนการอยู่เกิดหลังตำแหน่งที่ 2 ของรูป 3.5 เมื่อเงื่อนไขถูกต้อง (true) ซึ่งจะอธิบายการออกแบบการกดในข้อหัวถัดไป ตัวอย่างคีย์ที่ถูกกด ดังรูปที่ 3.8



รูปที่ 3.8 ตัวอย่างสถานะของคีย์ที่ถูกกดแล้ว

3.1.5 การกดคีย์เพื่อแสดงการตอบสนอง

การกดคีย์เพื่อแสดงการตอบสนองนั้น ทำการออกแบบโดยกำหนดค่าคงที่ไว้ค่าหนึ่ง (threshold) และสิ่งที่ใช้ในการตัดสินใจว่าคีย์นั้นถูกกดแล้ว คือ การวัดความเร็วปลายนิ้วมือเมื่อนิ้วนั้นเคลื่อนไหวในแนวทิศทางลง (ในแนวแกน y) จากนั้นทำการพิจารณาค่าความเร็วปลายนิ้วหากมีค่ามากกว่าค่าที่กำหนดไว้จึงจะเข้าเงื่อนไขของการกดคีย์เปียโน ซึ่งกระบวนการอยู่ตำแหน่งที่ 2 ของรูป 3.5 จากนั้น คีย์จะเปล่งเสียงออกมาผ่านลำโพงและคีย์นั้นจะเป็นสีแดง

3.1.6 การสร้างเสียงให้กับเปียโน

กระบวนการทำงานของปริญญานิพนธ์นี้ เอาต์พุตคือการแสดงภาพผ่านทางจอแสดงผลและเล่นเสียงเปียโนออกทางลำโพง ในการออกแบบให้โมเดลเปียโนมีเสียงนั้นจะใช้ไฟล์ MIDI ในการกำหนดเสียงให้แต่ละคีย์ออปเจกต์ โดยการใช้ค่าพารามิเตอร์ในตาราง MIDI ของเปียโน ซึ่งแต่ละคีย์แต่ละอ็อกเทฟมีค่าพารามิเตอร์ที่แตกต่างกัน ยกตัวอย่างเช่น ต้องการเสียงโดอ็อกเทฟที่ 2 (โด สัญลักษณ์ตัวโน้ต คือ C) ดังนั้นค่าพารามิเตอร์ที่ใช้ในการกำหนดคือ 36 ดังตารางที่ 2.5

เนื่องจาก MIDI คือ การสื่อสารข้อมูลทางดนตรีแบบดิจิทัล ซึ่งมนุษย์ไม่สามารถที่จะได้ยินเสียงนั้นด้วยหูได้ จึงต้องมีการแปลงสัญญาณจากอุปกรณ์แปลงและส่งผ่านลำโพงให้หูได้ยิน ซึ่งอุปกรณ์ที่ใช้ในการแปลงสัญญาณนั่นคือ ซินธิไซเซอร์ (synthesizer) หรือเครื่องสังเคราะห์เสียง ซึ่งในปริญญานิพนธ์นี้จะใช้ซอฟต์แวร์ซินธิไซเซอร์ กระบวนการอยู่ตำแหน่งที่ 3 ของรูป 3.5 ไลบรารี่ที่ใช้มีชื่อว่า C# Synth เป็นซอฟต์แวร์ซินธิไซเซอร์ด้วยภาษา C# สามารถดาวน์โหลดได้ที่ [10] ซึ่งจะช่วยให้ไฟล์ MIDI ที่สร้างขึ้นสามารถใช้งานได้ โดยการใช้งานจำเป็นต้องกำหนดค่าพารามิเตอร์ต่าง ๆ เพื่อให้ฟังก์ชันในซอฟต์แวร์ซินธิไซเซอร์สามารถทำงานได้

```
public void
StartPlayingKey (int channel,int note,int volume,int instrument)
```

รูปที่ 3.9 พารามิเตอร์เพื่อให้ฟังก์ชันในซอฟต์แวร์ซินธิไซเซอร์สามารถทำงานได้

จากรูปที่ 3.9 ค่าพารามิเตอร์ที่จำเป็นต้องกำหนด คือ channel, note, volume และ instrument โดย channel คือ ช่องเสียงที่เลือกใช้ ซึ่งในปริญญานิพนธ์นี้จะใช้ช่องเสียงเดียวกันทั้งหมด note คือ ค่าพารามิเตอร์ที่กำหนดมาจากตารางโน้ตดนตรีของ MIDI (ตารางที่ 2.5 ในบทที่ 2) volume คือ ระดับความดังของเสียง และ instrument คือ ค่าพารามิเตอร์ตัวเลขแทนชนิดของเครื่องดนตรี (รูปที่ 3.10) ดังนั้นปริญญานิพนธ์นี้จึงมีการกำหนดค่า channel = 1, volume = 100, note ตั้งแต่ ตัวโน้ต C อ็อกเทฟที่ 1 ถึง ตัวโน้ต B อ็อกเทฟที่ 4 และ instrument = 3 ซึ่งเป็นการเลือกเสียงเครื่องดนตรีแบบ honky-tonk piano

Piano	**Bass**	**Reed**	**Synth Effects**
0 Acoustic Grand Piano	32 Acoustic Bass	64 Soprano Sax	96 FX 1 (rain)
1 Bright Acoustic Piano	33 Electric Bass (finger)	65 Alto Sax	97 FX 2 (soundtrack)
2 Electric Grand Piano	34 Electric Bass (pick)	66 Tenor Sax	98 FX 3 (crystal)
3 Honky-tonk Piano	35 Fretless Bass	67 Baritone Sax	99 FX 4 (atmosphere)
4 Electric Piano 1	36 Slap Bass 1	68 Oboe	100 FX 5 (brightness)
5 Electric Piano 2	37 Slap Bass 2	69 English Horn	101 FX 6 (goblins)
6 Harpsichord	38 Synth Bass 1	70 Bassoon	102 FX 7 (echoes)
7 Clavinet	39 Synth Bass 2	71 Clarinet	103 FX 8 (sci-fi)
8 Celesta			
Chromatic Percussion	**Strings**	**Pipe**	**Ethnic**
9 Glockenspiel	40 Violin	72 Piccolo	104 Sitar
10 Music Box	41 Viola	73 Flute	105 Banjo
11 Vibraphone	42 Cello	74 Recorder	106 Shamisen
12 Marimba	43 Contrabass	75 Pan Flute	107 Koto
13 Xylophone	44 Tremolo Strings	76 Blown bottle	108 Kalimba
14 Tubular Bells	45 Pizzicato Strings	77 Shakuhachi	109 Bagpipe
15 Dulcimer	46 Orchestral Harp	78 Whistle	110 Fiddle
	47 Timpani	79 Ocarina	111 Shanai
Organ	**Ensemble**	**Synth Reed**	**Percussive**
16 Drawbar Organ	48 String Ensemble 1	80 Lead 1 (square)	112 Tinkle Bell
17 Percussive Organ	49 String Ensemble 2	81 Lead 2 (sawtooth)	113 Agogo
18 Rock Organ	50 Synth Strings 1	82 Lead 3 (calliope)	114 Steel Drums
19 Church Organ	51 Synth Strings 2	83 Lead 4 (chiff)	115 Woodblock
20 Reed Organ	52 Choir Aahs	84 Lead 5 (charang)	116 Taiko Drum
21 Accordion	53 Voice Oohs	85 Lead 6 (voice)	117 Melodic Tom
22 Harmonica	54 Synth Choir	86 Lead 7 (fifths)	118 Synth Drum
23 Tango Accordion	55 Orchestra Hit	87 Lead 8 (bass + lead)	119 Reverse Cymbal
Guitar	**Brass**	**Synth Pad**	**Sound effects**
24 Acoustic Guitar (nylon)	56 Trumpet	88 Pad 1 (new age)	120 Guitar Fret Noise
25 Acoustic Guitar (steel)	57 Trombone	89 Pad 2 (warm)	121 Breath Noise
26 Electric Guitar (jazz)	58 Tuba	90 Pad 3 (polysynth)	122 Seashore
27 Electric Guitar (clean)	59 Muted Trumpet	91 Pad 4 (choir)	123 Bird Tweet
28 Electric Guitar (muted)	60 French Horn	92 Pad 5 (bowed)	124 Telephone Ring
29 Overdriven Guitar	61 Brass Section	93 Pad 6 (metallic)	125 Helicopter
30 Distortion Guitar	62 Synth Brass 1	94 Pad 7 (halo)	126 Applause
31 Guitar Harmonics	63 Synth Brass 2	95 Pad 8 (sweep)	127 Gunshot

รูปที่ 3.10 ค่าพารามิเตอร์ตัวเลขแทนชนิดของเครื่องดนตรี

3.1.7 โหมดการเล่นเปียโนอัตโนมัติ

การออกแบบโปรแกรมโหมดเล่นอัตโนมัติ สามารถทำได้เมื่อทราบถึงโน้ตของเพลงที่ต้องการจะเล่นและทำการแปลงโน้ตนั้นตามค่าพารามิเตอร์ในตาราง MIDI จากตารางที่ 2.5 ในบทที่ 2 และกำหนดค่าของเวลาในการเปลี่ยนไปยังโน้ตถัดไป รวมทั้งกำหนดค่าของเวลาในการกำหนดให้เสียงมีความยาวเท่าไร และซอฟต์แวร์ซินธิไซเซอร์ในการสังเคราะห์เสียงออกมา โดยการสร้างไฟล์เพลง MIDI จะจัดเก็บในลักษณะของ .txt ดังรูปที่ 3.11

```
-------------------------------------------------
channel, note, volume, instrument, duration, time
-------------------------------------------------
1,64,48,1,1.5,0
1,60,48,1,1.5,1.5
1,62,48,1,1.5,0
1,55,47,1,1.5,1.5
1,60,39,1,1.5,0
1,57,47,1,1.5,1.5
1,59,45,1,1.5,0
1,52,45,1,1.5,1.5
1,57,43,1,1.5,0
1,53,47,1,1.5,1.5
1,55,48,1,1.5,0
1,48,41,1,1.5,1.5
1,57,48,1,1.5,0
1,53,52,1,1.5,1.5
1,59,51,1,1.5,0
1,55,53,1,1.5,1.5
1,64,47,1,1.5,0
```

รูปที่ 3.11 ตัวอย่างส่วนหนึ่งของเพลง Canon ผู้แต่ง Pachelbel

ไฟล์ text ดังกล่าวดัดแปลงมาจากไฟล์ .mid ซึ่งเป็นไฟล์มาตรฐาน MIDI ซึ่งใช้ซอฟต์แวร์ MATLAB [https://github.com/kts/matlab-midi] ในการดัดแปลง ในไฟล์ text ประกอบไปด้วยชุดตัวเลขในแต่ละบรรทัด แทนการเล่นโน้ต 1 ตัว แต่ละบรรทัดประกอบไปด้วย channel, note, volume, instrument, duration และ time ซึ่งกำหนด ช่องเสียง ตัวโน้ต ความดัง ชนิดของเครื่องดนตรี (เปียโน) ระยะเวลาที่เล่นโน้ต (หากหมดเวลาจะสั่ง note off อัตโนมัติ) และเวลาในการหน่วงเพื่อกำหนดจังหวะให้โน้ตตัวถัดไป เช่นบรรทัดแรก 1,64,48,1,1.5,0 หมายความว่า ให้เล่นที่ช่องเสียง 1 โน้ตตัวที่ 64 ความดัง 48 เครื่องดนตรีเปียโนคลาสสิก ด้วยระยะเวลา 1.5 วินาที หน่วงเวลาก่อนจะเล่นโน้ตถัดไป 0 วินาที ในกรณีนี้ โน้ตบรรทัดที่ 2 จะเล่นพร้อมบรรทัดแรกเนื่องจากไม่มีการหน่วงเวลาเกิดขึ้น

3.2 เครื่องมือที่ใช้ในการทดลอง

3.2.1 อุปกรณลีปโมชัน (Leap_Motion_Orion_Setup_win_3.2.0)

รูปที่ 3.12 อุปกรณ์ลีปโมชัน

ความสูง	1.27	เซนติเมตร
ความกว้าง	3.05	เซนติเมตร
ความลึก	7.62	เซนติเมตร
น้ำหนัก	45.35	กรัม
ซอฟต์แวร์	ซอฟต์แวร์ของลีปโมชันที่[6]	
สายเคเบิล	สาย USB 2.0 ความยาว 60.96 เซนติเมตรและ 152.4 เซนติเมตร (ตัวเชื่อมต่อ ไมโคร USB 3.0)	

3.2.2 โปรแกรมยูนิตี้

ยูนิตี้เวอร์ชัน 5.4.1f1 (สำหรับ windows 64-bit)

https://unity3d.com/get-unity/download/archive

3.3 การจัดเก็บผลการทดลอง

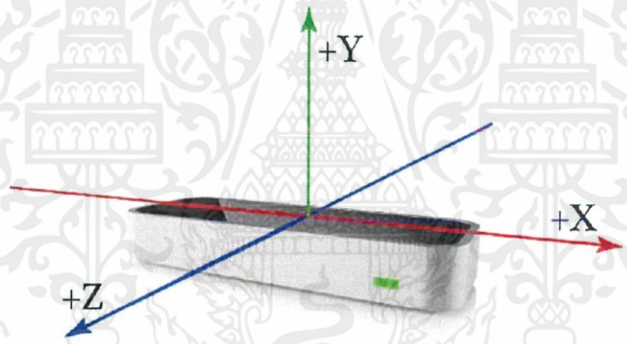
ขั้นแรกทำการทดลองเก็บค่าที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้ทั้ง 3 แกน (แกน x, y, z) จากนั้นทำการหาค่าที่เหมาะสมสำหรับโมเดลเปียโน พร้อมทั้งทำการทดสอบความสัมพันธ์ระหว่างนิ้วมือกับมาร์กเกอร์ และสุดท้ายทำการทดสอบเอาต์พุต โดยการเก็บค่าเปอร์เซ็นต์ความถูกต้องในการเล่นเพลงแต่ละเพลง โดยมีทั้งหมด 6 เพลง ประกอบด้วยเพลงหนูมาลี (มาจากเพลง mary had a little lamb แต่งโดย Sarah Josepha Haleon ปีค.ศ. 1830), Old Mcdonald (แต่งโดย F. T. Nettleingham ปี ค.ศ.1917), ช้าง (แต่งโดย คุณหญิงชิ้น ศิลปบรรเลง ปีพ.ศ. 2548), แมงมุมลาย (มาจากเพลง itsy bitsy spider แต่ไม่ปรากฏผู้แต่ง), Happy Birthday (แต่งโดย Patty and Mildred J. Hill ปีค.ศ. 1893) และค้างคาวกินกล้วย (ไม่ปรากฏผู้แต่ง) ซึ่งมีผู้เล่นในการทดสอบทั้งหมด 3 คน

บทที่ 4

ผลการทดลอง

เนื่องจากปริญญานิพนธ์เล่มนี้นำเสนอการเล่นเปียโนจำลอง โดยมีอินพุตคือ มือที่ลอยอยู่บนอุปกรณ์ลีปโมชัน จากนั้นอุปกรณ์ลีปโมชันทำการตรวจจับและแสดงรูปแบบมือที่ผู้จัดทำได้ออกแบบไว้บนหน้าต่างแสดงผล และเอาต์พุตคือเมื่อมีการสัมผัสวัตถุหรือโมเดลจำลองเปียโนจะทำให้เกิดเสียงโดยผ่านลำโพงและมีการตอบสนองของแต่ละคีย์หรือแต่ละวัตถุและทำการทดสอบความสามารถและระดับการตรวจจับของอุปกรณ์ลีปโมชันในแต่ละแกน (x, y และ z)

4.1 ผลการทดลอง หาค่าสูงและต่ำสุดที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้



รูปที่ 4.1 ระบบพิกัดของอุปกรณ์ลีปโมชัน

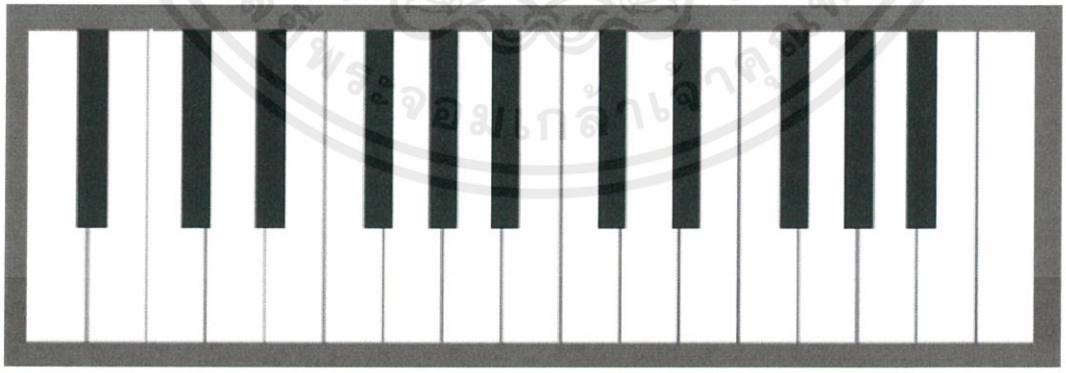
จากทฤษฎีของตัวควบคุมของอุปกรณ์ลีปโมชัน ซึ่งใช้เซ็นเซอร์แสงและแสงอินฟราเรด โดยเซ็นเซอร์มีทิศทางไปในแนวแกน y สูงขึ้นไปเมื่อตัวควบคุมอยู่ในตำแหน่งในการดำเนินงานมาตรฐาน ดังรูปที่ 4.1 และมีมุมมองประมาณ 150 องศา ระยะที่มีประสิทธิภาพของตัวควบคุมของอุปกรณ์ลีปโมชันขยายได้ประมาณ 25-600 มิลลิเมตรเหนืออุปกรณ์ จากการทดลองเมื่อทำการเก็บค่าระยะที่สามารถตรวจจับได้ พบว่า ระยะที่แกน x และ z จะ ได้ระยะสูงสุดขึ้นอยู่กับความสูงของ y แสดงค่าประมาณดังตารางที่ 4.1

ตารางที่ 4.1 ระยะโดยประมาณที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้

ระยะห่างระหว่างมือกับอุปกรณ์ลีปโมชัน (เซนติเมตร)	ระยะที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้ (เซนติเมตร)		
	แกน y	แกน x	แกน z
5	27.78 ถึง 100.12	-14.82 ถึง 13.26	-27.78 ถึง 100.12
15		-31.81 ถึง 36.31	-26.94 ถึง 50.51
35		-87.21 ถึง 90.34	-26.33 ถึง 83.28
65 ถึง 70		-91.22 ถึง 100.75	-61.62 ถึง 109.53

จากตารางที่ 4.1 เมื่อทำการเก็บค่าระยะที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้ พบว่าเมื่อระยะห่างระหว่างมือกับอุปกรณ์ลีปโมชันเพิ่มขึ้น ระยะที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้ในแกน x และ z จะเพิ่มขึ้นด้วย

4.2 ผลการทดลองการสร้างโมเดลของเปียโนโดยใช้โปรแกรมยูนิตี้

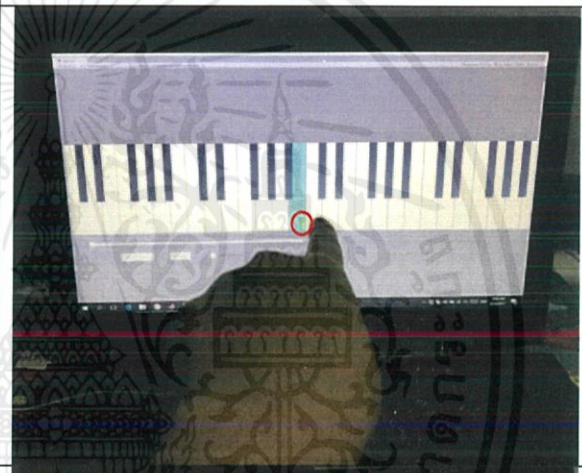
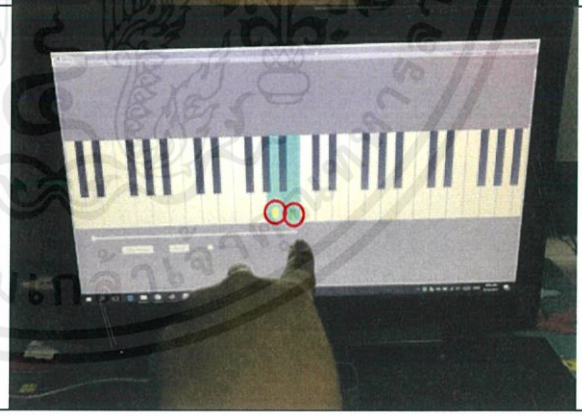


รูปที่ 4.2 โมเดลจำลองเปียโนออกแบบโดยโปรแกรมยูนิตี้

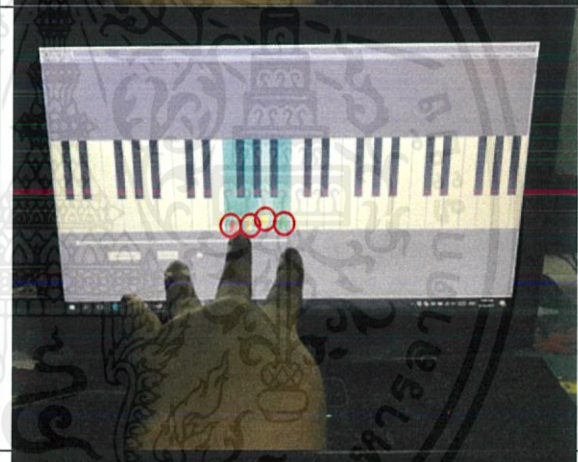
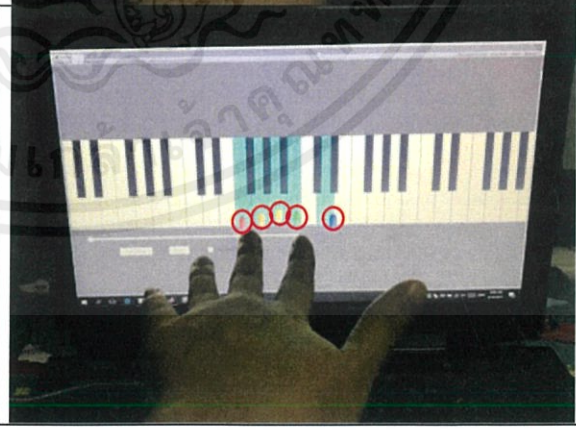
จากการทดลองสร้างโมเดลจำลองเปียโนดังรูปที่ 4.2 พบว่าค่าความกว้าง ความยาว และ ความสูง ที่เหมาะสมในการนำมาใช้งานของคีย์สีขาว มีค่าเท่ากับ 2.20 12.50 และ 1.50 เซนติเมตร ตามลำดับ โดยคีย์ฝั่งซ้ายสุดจะเริ่มที่ตำแหน่ง (-47.2, 0, 0) และคีย์สีดำ มีค่าเท่ากับ 1.10 7.20 และ 0.80 เซนติเมตร ตามลำดับ

4.3 ผลการทดลองแสดงความสัมพันธ์ระหว่างอินพุต (มือผู้ใช้งาน) กับมาร์กเกอร์

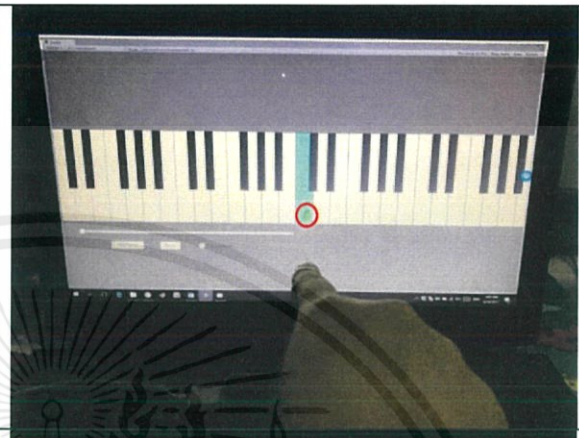
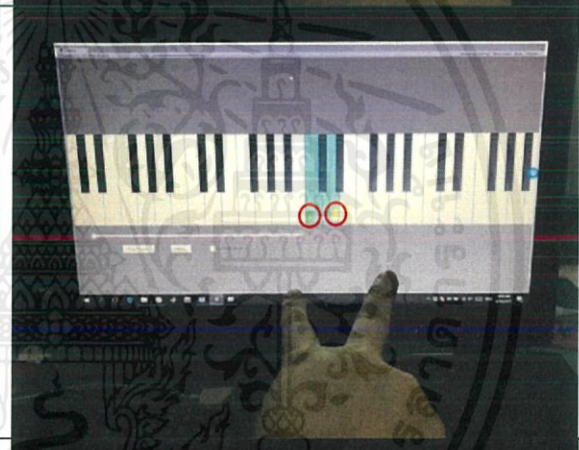
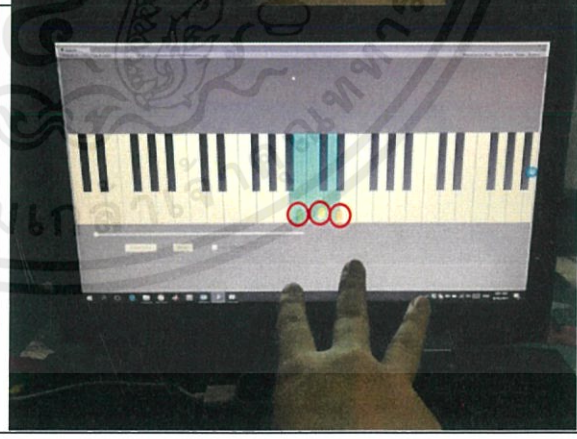
ตารางที่ 4.2 ความสัมพันธ์ระหว่างอินพุต (มือผู้ใช้งาน) กับมาร์กเกอร์

ชนิดของมือ	จำนวนนิ้วมือ	มาร์กเกอร์ที่แสดงบนหน้าจอ
มือซ้าย	1	
	2	

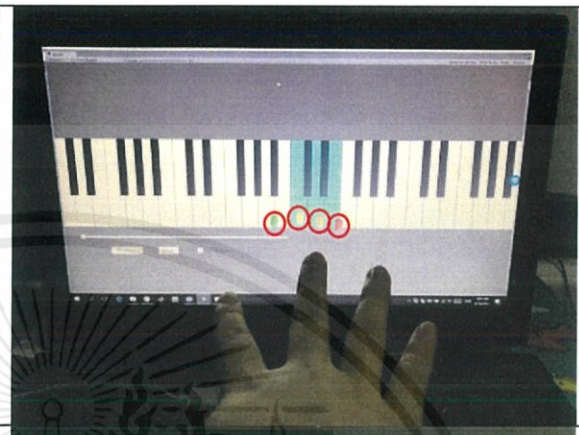
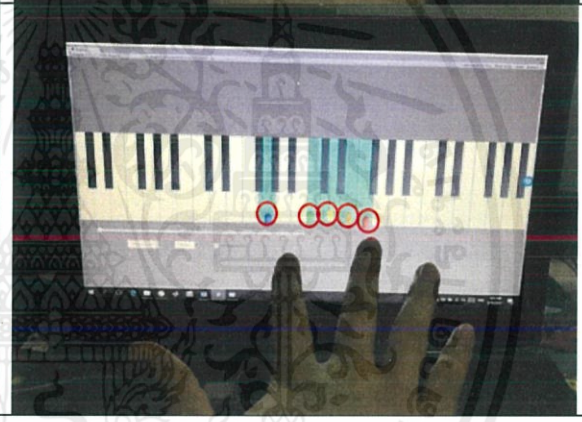
ตารางที่ 4.2 (ต่อ) ความสัมพันธ์ระหว่างอินพุต (มือผู้ใช้งาน) กับมาร์กเกอร์

ชนิดของมือ	จำนวนนิ้วมือ	มาร์กเกอร์ที่แสดงบนหน้าจอ
มือซ้าย	3	
	4	
	5	

ตารางที่ 4.2 (ต่อ) ความสัมพันธ์ระหว่างอินพุต (มือผู้ใช้งาน) กับมาร์กเกอร์

ชนิดของมือ	จำนวนนิ้วมือ	มาร์กเกอร์ที่แสดงบนหน้าจอ
มือขวา	1	
	2	
	3	

ตารางที่ 4.2 (ต่อ) ความสัมพันธ์ระหว่างอินพุต (มือผู้ใช้งาน) กับมาร์กเกอร์

ชนิดของมือ	จำนวนนิ้วมือ	มาร์กเกอร์ที่แสดงบนหน้าจอ
มือขวา	4	
	5	

จากการสร้างมาร์กเกอร์ (Marker) หรือตัวแสดงตำแหน่งปลายนิ้วมือและแอคทีฟคีย์ (Active key) หรือตัวแสดงตำแหน่งคีย์ที่กำลังจะถูกกด พบว่า มาร์กเกอร์และแอคทีฟคีย์คีย์นั้นสัมพันธ์กับอินพุตหรือมือของผู้เล่นทั้งมือซ้ายและมือขวา ดังตารางที่ 4.2

4.4 ผลการทดลองความถูกต้องของการกดคีย์เปียโนเมื่อมีการเล่นเพลง

ทำการทดสอบความถูกต้องของการกดคีย์เปียโนเมื่อมีการเล่นเพลง ทดสอบโดยให้ผู้เล่นทั้งหมด 3 คน เพลงที่ใช้ทดสอบทั้งหมด 6 เพลง เรียงลำดับตามจำนวนตัวโน้ตจากน้อยไปมาก ได้แก่ หนูมาลี (มาจากเพลง mary had a little lamb แต่งโดย Sarah Josepha Haleon ปีค.ศ. 1830), Old Mcdonald (แต่งโดย F. T. Nettleingham ปีค.ศ.1917), ช้าง (แต่งโดย คุณหญิงชิ้น ศิลปบรรเลง ปีพ.ศ. 2548), แมงมุมลาย (มาจากเพลง itsy bitsy spider แต่ไม่ปรากฏผู้แต่ง), Happy Birthday (แต่งโดย Patty and Mildred J. Hill ปีค.ศ. 1893) และค้างคาวกินกล้วย

(ไม่ปรากฏผู้แต่ง) ตามลำดับ ผลลัพธ์ที่ถูกต้องพิจารณาจากจำนวนคีย์ที่กดถูกตำแหน่งและให้เสียงถูกต้อง ผลลัพธ์แสดงตามตารางที่ 4.3

ตารางที่ 4.3 ทดสอบความถูกต้องของการกดคีย์เปียโนเมื่อมีการเล่นเพลง

เพลง	ผู้เล่น	จำนวนโน้ตทั้งหมด	จำนวนคีย์ที่กดถูกต้อง
หนูมาลี	1	25	23
	2	25	22
	3	25	20
เปอร์เซ็นความถูกต้อง (%)		86.7	
Old Mcdonald	1	26	21
	2	26	19
	3	26	22
เปอร์เซ็นความถูกต้อง (%)		79.5	
ช้าง	1	36	31
	2	36	29
	3	36	29
เปอร์เซ็นความถูกต้อง (%)		82.4	
แมงมุมลาย	1	47	42
	2	47	41
	3	47	36
เปอร์เซ็นความถูกต้อง (%)		84.4	
Happy Birthday	1	50	39
	2	50	35
	3	50	38
เปอร์เซ็นความถูกต้อง (%)		74.7	

ตารางที่ 4.3 (ต่อ) ทดสอบความถูกต้องของการกดคีย์เปียโนเมื่อมีการเล่นเพลง

เพลง	ผู้เล่น	จำนวนโน้ตทั้งหมด	จำนวนคีย์ที่กดถูกต้อง
ค้างคาวกินกล้วย	1	119	90
	2	119	83
	3	119	91
เปอร์เซ็นความถูกต้อง (%)		74.0	
เปอร์เซ็นความถูกต้องโดยรวม (%)		78.2	

จากตารางที่ 4.3 การทดสอบกดคีย์เปียโนเมื่อมีการเล่นเพลงจากผู้เล่นทั้งหมด 3 คน พบว่า การกดคีย์เพลงหนูมาลีที่มีจำนวนโน้ต 25 ตัว มีความถูกต้องประมาณ 86.67% การกดคีย์เพลง Old Mcdonald ที่มีจำนวนโน้ต 26 ตัว มีความถูกต้องประมาณ 79.49% การกดคีย์เพลงช้างที่มีจำนวนโน้ต 36 ตัว มีความถูกต้องประมาณ 82.41% การกดคีย์เพลงแมงมุมลายที่มีจำนวนโน้ต 47 ตัว มีความถูกต้องประมาณ 84.40% การกดคีย์เพลง Happy Birthday ที่มีจำนวนตัวโน้ต 50 ตัว มีความถูกต้องประมาณ 74.67% และการกดคีย์เพลงค้างคาวกินกล้วยที่มีจำนวนตัวโน้ต 119 ตัว มีความถูกต้องประมาณ 73.95% จะเห็นได้ว่าความผิดพลาดในการกดคีย์ของเพลงหนูมาลีจะมีน้อยที่สุด เนื่องจากความซับซ้อนของการเล่นเพลงนี้มีอยู่ในระดับต่ำ นั่นคือ มีจำนวนคีย์น้อยและมีคีย์ซ้ำเดิมค่อนข้างมาก และความผิดพลาดในการกดคีย์ของเพลงค้างคาวกินกล้วยจะมีมากที่สุด เนื่องจากมีความซับซ้อนของการเล่นสูงที่สุด นั่นคือ มีจำนวนคีย์มาก และใช้จำนวนอ็อกเทฟมากกว่าเพลงอื่น ความซับซ้อนของการเล่นเพลง เช่น จำนวนคีย์ จำนวนการใช้อ็อกเทฟ เป็นต้น จึงมีผลกระทบต่อการกดคีย์ที่กดแล้วให้ผลลัพธ์ถูกต้อง โดยเพลงที่มีความซับซ้อนของการเล่นในระดับที่สูงขึ้น ความผิดพลาดของการเล่นเพลงจึงเพิ่มขึ้นด้วย โดยรวมแล้วจึงสรุปได้ว่า โปรแกรมเปียโนจำลองให้ความถูกต้องในการกดคีย์เปียโนเมื่อมีการเล่นเพลงประมาณ 78.23%

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

โครงงานนี้นำเสนอการออกแบบและสร้างโปรแกรมประยุกต์เปียโนจำลองโดยใช้อุปกรณ์ต่อพ่วงลีปโมชัน (Leap Motion) ในการควบคุม และซอฟต์แวร์ยูนิตี้ (Unity) ในการออกแบบ โครงงานนี้ประกอบไปด้วยการออกแบบโมเดลเปียโนจำลองโดยโปรแกรมยูนิตี้ และทำการออกแบบมาร์กเกอร์ใช้ในแทนการระบุตำแหน่งของปลายนิ้วมือ เพื่อช่วยให้ผู้เล่นสามารถทราบถึงตำแหน่งว่าอยู่ที่คีย์ใด นอกจากนี้ทำการใส่เสียงให้แต่ละคีย์ โดยใช้ MIDI เนื่องจาก MIDIเป็นการสื่อสารข้อมูลทางดนตรีแบบดิจิตอล ซึ่งมนุษย์ไม่สามารถได้ยินได้ จึงต้องใช้ซอฟต์แวร์ซินธิไซเซอร์ในการสังเคราะห์เสียงและใช้ความเร็วปลายนิ้วเป็นตัวกำหนดให้เงื่อนไขทำงาน ซึ่งเงื่อนไขคือเมื่อความเร็วปลายนิ้วมีค่ามากกว่าค่าที่กำหนด คีย์ที่ถูกกดจะแสดงเสียงผ่านลำโพง และการทดสอบของโครงงานนี้ ทำการทดสอบการออกแบบและเอาต์พุตทั้งหมด 4 การทดลอง ซึ่งประกอบด้วย การทดสอบขอบเขตการตรวจจับของอุปกรณ์ลีปโมชัน การทดสอบค่าที่เหมาะสมในการออกแบบโมเดลเปียโน การทดสอบความสัมพันธ์ระหว่างจำนวนนิ้วมือกับมาร์กเกอร์ และการทดสอบประสิทธิภาพของโปรแกรมประยุกต์เปียโนจำลองเมื่อมีผู้เล่นจำนวนทั้งหมด 3 คน คนละ 6 เพลง

จากการทดลองที่ 4.1 การทดสอบขอบเขตการตรวจจับของอุปกรณ์ลีปโมชัน พบว่าเมื่อระยะห่างระหว่างมือกับอุปกรณ์ลีปโมชันเพิ่มขึ้น ระยะที่ลีปโมชันสามารถตรวจจับได้ในแกน x และ z จะเพิ่มขึ้นด้วย ซึ่งค่ามากสุด น้อยสุด ที่อุปกรณ์ลีปโมชันสามารถตรวจจับได้แต่ละแกน แสดงดังตารางที่ 4.1

จากการทดลองที่ 4.2 การทดสอบค่าที่เหมาะสมในการออกแบบโมเดลเปียโนความกว้าง ความยาว และ ความสูง ที่เหมาะสมในการนำมาใช้งานของคีย์สีขาว มีค่าเท่ากับ 2.20 12.50 และ 1.50 เซนติเมตร ตามลำดับ โดยคีย์ฝั่งซ้ายสุดจะเริ่มที่ตำแหน่ง (-47.2, 0, 0) และคีย์สีดำ มีค่าเท่ากับ 1.10 7.20 และ 0.80 เซนติเมตร ตามลำดับ

จากการทดลองที่ 4.3 การทดสอบความสัมพันธ์ระหว่างจำนวนนิ้วมือกับมาร์กเกอร์ พบว่า มาร์กเกอร์ สามารถแสดงความสัมพันธ์ระหว่างจำนวนนิ้วมือกับจำนวนมาร์กเกอรีได้ เช่น นิ้วมือผู้เล่นจำนวน 2 นิ้ว มาร์กเกอร์จะแสดง 2 ตัว

จากการทดลองที่ 4.4 การทดสอบประสิทธิภาพของโปรแกรมประยุกต์เปียโนจำลอง พบว่า ผลลัพธ์พิจารณาจากจำนวนคีย์ที่กดถูกตำแหน่งและให้เสียงถูกต้อง โปรแกรมเปียโนจำลองให้ความถูกต้องประมาณ 78.23% ซึ่งยังมีความผิดพลาดอยู่บ้าง โดยความซับซ้อนของการเล่นเพลง เช่น จำนวนคีย์ และจำนวนอ็อกเทฟที่ใช้ เป็นต้น จะมีผลกระทบต่อการกดคีย์ที่กดแล้วให้ผลลัพธ์ถูกต้อง ซึ่งเพลงที่มีความซับซ้อนของการเล่นในระดับที่สูงขึ้น ความผิดพลาดของการเล่นเพลงจึงเพิ่มขึ้นด้วย

ดังนั้นเปียโนจำลองสามารถเล่นได้ทั้งสองมือและเล่นเพลงที่มีคอร์ดง่ายได้ โดยผู้เล่นควรใช้เวลาทำความคุ้นเคยกับโปรแกรมประยุกต์เปียโนจำลองเสียก่อน เพื่อให้เกิดความผิดพลาดของการกดคีย์เปียโนให้น้อยที่สุด

5.2 ข้อเสนอแนะ

ปัจจุบันลีปโมชันนั้นกำลังเป็นที่นิยมอย่างแพร่หลายในการนำไปใช้งานร่วมกับ Virtual Reality (VR) ซึ่งซอฟต์แวร์ Leap Motion Orient ก็ยังพัฒนามาสำหรับ VR โดยเฉพาะอีกด้วย ทำให้การใช้งานลีปโมชันแบบ VR อาจให้ผลลัพธ์ที่ดีกว่าในแง่ของประสิทธิภาพในการตรวจจับภาพของมือและข้อนิ้วต่าง ๆ รวมถึงการแสดงผลที่น่าสนใจกว่า ทางผู้จัดทำเห็นว่าเป็นการน่าสนใจหากต่อไปจะพัฒนาโปรแกรมประยุกต์เปียโนจำลองบนแพลตฟอร์ม VR

บรรณานุกรม

[1] "เปียโน."
https://th.wikipedia.org/wiki/เปียโน.

[2] "เรียนรู้การเล่นคีย์บอร์ดเบื้องต้น."
https://sites.google.com/site/ohmygod13456/reiyn-ru-kar-len-khi-bxrd-beuxn-g-tn.

[3] "ความถี่เสียงเปียโน."
https://th.wikipedia.org/wiki/ความถี่เสียงเปียโน.

[4] "Leap Motion."
https://www.leapmotion.com/#102.

[5] "C# SDK Documentation."
https://developer.leapmotion.com/documentation/csharp/index.html.

[6] "SDK Download Leap Motion."
https://developer.leapmotion.com/get-started/.

[7] "Develop Leap Motion."
https://developer.leapmotion.com.

[8] "Unity."
https://unity3d.com/.

[9] "Leap Motion with Unity."
https://developer.leapmotion.com/documentation/unity/.

[10] "C#syn."
https://csharpsynthproject.codeplex.com.

[11] "Frame Class Reference."
https://developer.leapmotion.com/documentation/csharp/api/gencsharp/class_leap_1_1_frame.html.

[12] "Hand Class Reference."
https://developer.leapmotion.com/documentation/csharp/api/gencsharp/class_leap_1_1_hand.html.

[13] "Finger Class Reference."
https://developer.leapmotion.com/documentation/csharp/api/gencsharp/class_leap_1_1_finger.html.
[14] "มิดิ."
https://th.wikipedia.org/wiki/มิดิ.
[15] สุวิทย์ วิชัยดิษฐ. "รูปแบบ midi/vsti."
http://www.commusic4.com/studio4/2013-03-13-02-31-29.
[16] "เครื่องสังเคราะห์เสียง."
https://th.wikipedia.org/wiki/เครื่องสังเคราะห์เสียง.
[18] "SoundFont."
https://en.wikipedia.org/wiki/SoundFont.
[19] "Sample-based synthesis."
https://en.wikipedia.org/wiki/Sample-based_synthesis.
[20] "Synthesizer."
https://en.wikipedia.org/wiki/Synthesizer.
[21] "Additive synthesis."
https://en.wikipedia.org/wiki/Additive_synthesis.
[22] "Subtractive synthesis."
https://en.wikipedia.org/wiki/Subtractive_synthesis.
[23] Teng Wei Jian. "Final Project: Piano Sounds Synthesis."
http://www.acoustics.ed.ac.uk/wp-content/uploads/AMT_MSc_FinalProjects-/2012__Teng__AMT_MSc_FinalProject_PianoSynthesis.pdf.
[24] "Sampler (musical instrument)."
https://en.wikipedia.org/wiki/Sampler_(musical_instrument).

ภาคผนวก ก

คำสั่งที่ใช้ในการทำงาน

Code 1: Parameters.cs

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

[ExecuteInEditMode]
public class Parameters : MonoBehaviour { //Utils
    // Key Parameters
    [Range(0.01f, 0.05f)] public float WHITE_KEY_WIDTH = 0.02215f;
 // 22.15 mm
    [Range(1f, 8f)] public float WHITE_KEY_RATIO = 5.7006772009f;
    public float WHITE_KEY_LEN = 0.12627f;
    public float WHITE_KEY_DEPTH = 0.001f;

    [Range(0.01f, 0.05f)] public float BLACK_KEY_WIDTH = 0.01100f;
 // 11.00 mm
    [Range(1f, 8f)] public float BLACK_KEY_RATIO = 7.27272727273f;
    public float BLACK_KEY_LEN = 0.08000f;
    public float BLACK_KEY_DEPTH = 0.003f;

    public Vector3 ORIGIN = new Vector3(0, 0, 0);
    [Range(0f, 0.003f)] public float DELTA_X = 0.00127f;
    public float WKEY_DISTANCE = 0.02342f;

    public enum KeyNote { _1A = 33, _1As, _1B, _2C, _2Cs, _2D, _2D
s, _2E, _2F, _2Fs, _2G, _2Gs, _2A, _2As, _2B, _3C, _3Cs, _3D, _3Ds
, _3E, _3F, _3Fs, _3G, _3Gs, _3A, _3As, _3B, _4C, _4Cs, _4D, _4Ds,
 _4E, _4F, _4Fs, _4G, _4Gs, _4A, _4As, _4B, _5C, _5Cs, _5D, _5Ds,
_5E, _5F, _5Fs, _5G, _5Gs, _5A, _5As, _5B, _6C };
    public enum KeyType {BLACK = 0, WHITE};

    private int NumOfKey = 85;
    public int[] WhiteKeyMapper, BlackKeyMapper;

    [Range(-1000f,-100f)] public float VELOCITY_THRESHOLD = -440f;

// Finger Position Offset
    public Vector3[][] HAND_OFFSET;
    public Vector3[] R_OFFSET;
    public Vector3[] L_OFFSET;
```

Code 1: Parameters.cs (Con.)

```
void Start() {
        WhiteKeyMapper = new int[NumOfKey];
        BlackKeyMapper = new int[NumOfKey];

        WhiteKeyMapper[33] = 0;
        WhiteKeyMapper[35] = 1;
        WhiteKeyMapper[36] = 2;
        WhiteKeyMapper[38] = 3;
        WhiteKeyMapper[40] = 4;
        WhiteKeyMapper[41] = 5;
        WhiteKeyMapper[43] = 6;
        WhiteKeyMapper[45] = 7;
        WhiteKeyMapper[47] = 8;
        WhiteKeyMapper[48] = 9;
        WhiteKeyMapper[50] = 10;
        WhiteKeyMapper[52] = 11;
        WhiteKeyMapper[53] = 12;
        WhiteKeyMapper[55] = 13;
        WhiteKeyMapper[57] = 14;
        WhiteKeyMapper[59] = 15;
        WhiteKeyMapper[60] = 16;
        WhiteKeyMapper[62] = 17;
        WhiteKeyMapper[64] = 18;
        WhiteKeyMapper[65] = 19;
        WhiteKeyMapper[67] = 20;
        WhiteKeyMapper[69] = 21;
        WhiteKeyMapper[71] = 22;
        WhiteKeyMapper[72] = 23;
        WhiteKeyMapper[74] = 24;
        WhiteKeyMapper[76] = 25;
        WhiteKeyMapper[77] = 26;
        WhiteKeyMapper[79] = 27;
        WhiteKeyMapper[81] = 28;
        WhiteKeyMapper[83] = 29;
        WhiteKeyMapper[84] = 30;
```

Code 1: Parameters.cs (Con.)

```
BlackKeyMapper[34] = 0;
        BlackKeyMapper[37] = 4;
        BlackKeyMapper[39] = 6;
        BlackKeyMapper[42] = 10;
        BlackKeyMapper[44] = 12;
        BlackKeyMapper[46] = 14;
        BlackKeyMapper[49] = 18;
        BlackKeyMapper[51] = 20;
        BlackKeyMapper[54] = 24;
        BlackKeyMapper[56] = 26;
        BlackKeyMapper[58] = 28;
        BlackKeyMapper[61] = 32;
        BlackKeyMapper[63] = 34;
        BlackKeyMapper[66] = 38;
        BlackKeyMapper[68] = 40;
        BlackKeyMapper[70] = 42;
        BlackKeyMapper[73] = 46;
        BlackKeyMapper[75] = 48;
        BlackKeyMapper[78] = 52;
        BlackKeyMapper[80] = 54;
        BlackKeyMapper[82] = 56;

        R_OFFSET = new Vector3[5];
        R_OFFSET[0] = new Vector3(0, 0, 0.5f / 100);
        R_OFFSET[1] = new Vector3(0, 0, 1f / 100);
        R_OFFSET[2] = new Vector3(0, 0, 1.5f / 100);
        R_OFFSET[3] = new Vector3(0, 0, 1f / 100);
        R_OFFSET[4] = new Vector3(0, 0, 0.5f / 100);

        L_OFFSET = new Vector3[5];
        L_OFFSET[0] = new Vector3(0, 0, 0.5f / 100);
        L_OFFSET[1] = new Vector3(0, 0, 1f / 100);
        L_OFFSET[2] = new Vector3(0, 0, 1.5f / 100);
        L_OFFSET[3] = new Vector3(0, 0, 1f / 100);
        L_OFFSET[4] = new Vector3(0, 0, 0.5f / 100);

        HAND_OFFSET = new Vector3[2][];
        HAND_OFFSET[0] = L_OFFSET;
        HAND_OFFSET[1] = R_OFFSET;

}
```

Code 1: Parameters.cs (Con.)

```
void Update () {
        // Ratio should be fixed
        if (WHITE_KEY_LEN != WHITE_KEY_WIDTH * WHITE_KEY_RATIO)
            WHITE_KEY_LEN = WHITE_KEY_WIDTH * WHITE_KEY_RATIO;
        if (BLACK_KEY_LEN != BLACK_KEY_WIDTH * BLACK_KEY_RATIO)
            BLACK_KEY_LEN = BLACK_KEY_WIDTH * BLACK_KEY_RATIO;
        if (WKEY_DISTANCE != WHITE_KEY_WIDTH + DELTA_X)
            WKEY_DISTANCE = WHITE_KEY_WIDTH + DELTA_X;

        //HAND_OFFSET[0] = L_OFFSET;
        //HAND_OFFSET[1] = R_OFFSET;
    }
```

Code 2: PianoKey.cs

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

[ExecuteInEditMode]
public class PianoKey : MonoBehaviour {

    //public GameObject synthesizer;
    public string keyName;
    public Parameters.KeyNote keyNote;
    public Parameters.KeyType keyType;
    public int note;

    private Synthesizer synthesizerScript;

    private int currentVolume;
    private int currentInstrument;
    private int currentChannel;

            void Awake()
    {
        //synthesizerScript = (Synthesizer)synthesizer.GetComponen
t(typeof(Synthesizer));
        synthesizerScript = FindObjectOfType<Synthesizer>() as Syn
thesizer;
        currentChannel = 1;
        currentVolume = 100;
        currentInstrument = 3;
    }
```

Code 2: PianoKey.cs (Con.)

```
void Start()
    {
        note = (int)keyNote;
        keyName = keyNote.ToString().TrimStart('_');
        gameObject.name = keyName;
        gameObject.tag = note.ToString();
    }

    void OnMouseDown()
    {
        HandleKeyPress(currentChannel, currentVolume, currentInstru
ment);
        //StartCoroutine(Example());
    }

    void OnMouseUp()
    {
        HandleKeyRelease(currentChannel);
    }

    public void HandleKeyRelease(int channel)
    {
        StopSound(channel);

    }
void Update () {
        if (keyName != Enum.GetName(typeof(Parameters.KeyType), (in
t)keyNote))
        {
            note = (int)keyNote;
            keyName = keyNote.ToString().TrimStart('_');
            gameObject.name = keyName;
        }
    }
public void HandleKeyPress(int channel, int volume, int instrumentN
umber)
    {
        PlaySound(channel, volume, instrumentNumber);

    }

    private void PlaySound(int channel, int volume, int instrumentN
umber)
    {
        synthesizerScript.StartPlayingKey(channel, note, volume, in
strumentNumber);
    }
```

Code 3: MovingFingerTip.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Leap;
using Leap.Unity;

//[ExecuteInEditMode]
public class MovingFingerTip : MonoBehaviour {

    LeapProvider provider;

    //public GameObject ParamController;
    private Parameters param; //Utils utils;

    public enum HandType { LEFT_HAND = 0, RIGHT_HAND };

    public Color color;
    public HandType handType;
    public Finger.FingerType fingerType;

    public Vector3 Offset = new Vector3(0, 0, 0);

    public float tipVelocity = 0;
    public float sensitivity = 0;
    public float loopCount = 0;
    public float sum = 0;
    public float ave = 0;
    public float inst = 0;

    public float speed = 20.0F;
    private float journeyLength;
    static float t = 0.0f;

void Start ()
    {
        provider = FindObjectOfType<LeapProvider>() as LeapProvider
;
        param = FindObjectOfType<Parameters>() as Parameters;
        GetComponent<Renderer>().material.color = color;
        transform.position = new Vector3(0, 0, 0);
        transform.rotation = Quaternion.Euler(new Vector3(90, 0, 0)
);
        transform.localScale = new Vector3(0.010f, 0.010f, 0.005f);
        GetComponent<Renderer>().enabled = false;
        Offset = param.HAND_OFFSET[(int)handType][(int)fingerType];
    }
```

Code 3: MovingFingerTip.cs (Con.)

```
void Update () {
        Frame frame = provider.CurrentFrame;
        Hand hand;
        //Offset = param.HAND_OFFSET[(int)handType][(int)fingerType
];
        if (frame.Hands.Count > 0)
        {
            if (frame.Hands.Count > 1)
            {
                switch (handType)
                {
                    case HandType.RIGHT_HAND:
                        hand = (frame.Hands[0].IsRight) ? frame.Han
ds[0] : frame.Hands[1];
                        break;
                    case HandType.LEFT_HAND:
                        hand =  (frame.Hands[0].IsLeft) ? frame.Han
ds[0] : frame.Hands[1];
                        break;
                    default:
                        hand = frame.Hands[0];
                        break;
                }
            }
            else
            {
                switch (handType)
                {
                    case HandType.RIGHT_HAND:
                        hand = (frame.Hands[0].IsRight) ? frame.Han
ds[0] : null;
                        break;
                    case HandType.LEFT_HAND:
                        hand = (frame.Hands[0].IsLeft) ? frame.Hand
s[0] : null;
                        break;
                    default:
                        hand = frame.Hands[0];
                        break;
                }
            }
```

Code 3: MovingFingerTip.cs (Con.)

```
if (hand == null) return;
            Finger finger = hand.Fingers[(int)fingerType];
            if (finger.IsExtended)
            {
                //Leap.Vector TipPosition = finger.TipPosition;
                loopCount += 1;
                sum = sum + (finger.TipPosition.x - hand.PalmPositi
on.x);
                ave = sum / loopCount;
                //inst = Mathf.Abs(finger.TipPosition.x - hand.Palm
Position.x);
                //Debug.Log(finger.TipVelocity);
                tipVelocity = finger.TipVelocity.y * 1000; // mm pe
r sec
                if (finger.TipPosition.z > param.ORIGIN.z)
                {
                    journeyLength = Vector3.Distance(transform.posi
tion, finger.TipPosition.ToVector3());

                    //Debug.Log(journeyLength/Time.deltaTime);
                    t += speed * Time.deltaTime;

                    float PosX = Mathf.Lerp(transform.position.x, f
inger.TipPosition.x, t);

                    transform.position = new Vector3(PosX + sensiti
vity * ((finger.TipPosition.x - hand.PalmPosition.x) - ave), param.
BLACK_KEY_DEPTH, param.ORIGIN.z) + Offset;
                    GetComponent<Renderer>().enabled = true;
                }
                else if (finger.TipPosition.z <= param.ORIGIN.z)
                {
                    journeyLength = Vector3.Distance(transform.posi
tion, finger.TipPosition.ToVector3());
                    t += speed * Time.deltaTime;
                    float PosX = Mathf.Lerp(transform.position.x, f
inger.TipPosition.x, t);

                    transform.position = new Vector3(PosX + sensiti
vity * ((finger.TipPosition.x - hand.PalmPosition.x) - ave), param.
BLACK_KEY_DEPTH, -param.WHITE_KEY_LEN / 2) + Offset;
                    GetComponent<Renderer>().enabled = true;
                }
            }
```

Code 3: MovingFingerTip.cs (Con.)

```
else
            {
                GetComponent<Renderer>().enabled = false;
                resetSum();
            }
        }
        else
        {
            GetComponent<Renderer>().enabled = false;
            resetSum();
        }
    }

    private void resetSum()
    {
        sum = 0;
        ave = 0;
        inst = 0;
        loopCount = 0;
        t = 0;
    }
}
```

Code 4: KeyManager.cs

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

[ExecuteInEditMode]
public class KeyManager : MonoBehaviour {
    public GameObject ParamController;
    private Parameters param;
    private PianoKey pianoKey;
    void Start () {
        pianoKey = (PianoKey)gameObject.GetComponent(typeof(PianoKe
y));
        param = (Parameters)ParamController.GetComponent(typeof(Par
ameters));
        InitialSetting();
    }
void Update () {
        InitialSetting();
    }
```

Code 4: KeyManager.cs (Con.)

```
void InitialSetting() {
        //transform.position = param.ORIGIN;
        //transform.localScale = new Vector3(param.WHITE_KEY_WIDTH,
 param.WHITE_KEY_DEPTH, param.WHITE_KEY_LEN);

        if (pianoKey.keyType == Parameters.KeyType.WHITE)
        {
            transform.position = param.ORIGIN + GetXPositionByKeyNo
te();
            //gameObject.GetComponent<Renderer>().material.color =
Color.white;
            transform.localScale = new Vector3(param.WHITE_KEY_WIDT
H, param.WHITE_KEY_DEPTH, param.WHITE_KEY_LEN);
        }
        else if (pianoKey.keyType == Parameters.KeyType.BLACK)
        {
            transform.position = param.ORIGIN + GetXPositionByKeyNo
te();
            //gameObject.GetComponent<Renderer>().material.color =
Color.black;
            transform.localScale = new Vector3(param.BLACK_KEY_WIDT
H, param.BLACK_KEY_DEPTH, param.BLACK_KEY_LEN);
            //Debug.Log("IN MAPPER BLACK = " + param.WhiteKeyMapper
[(int)pianoKey.keyNote]);
        }
    }

    Vector3 GetXPositionByKeyNote()
    {
        if (pianoKey.keyType == Parameters.KeyType.WHITE)
            return new Vector3(param.WhiteKeyMapper[(int)pianoKey.k
eyNote] * param.WKEY_DISTANCE, 0, 0);
        if (pianoKey.keyType == Parameters.KeyType.BLACK)
            return new Vector3((1 + param.BlackKeyMapper[(int)piano
Key.keyNote]) * (param.WKEY_DISTANCE/2), 0, (param.WHITE_KEY_LEN -
param.BLACK_KEY_LEN)/2);
        //param.BlackKeyMapper[(int)pianoKey.keyNote]
        return param.ORIGIN;
    }
}
```

Code 5: Synthesizer.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using CSharpSynth.Effects;
using CSharpSynth.Sequencer;
using CSharpSynth.Synthesis;
using CSharpSynth.Midi;

public class Synthesizer : MonoBehaviour {
    public string midiFilePath = "Midis/Groove.mid";
    public string bankFilePath = "GM Bank/gm";

    private int bufferSize = 1024;
    private float[] sampleBuffer;
    private float gain = 1f;
    private MidiSequencer midiSequencer;
    private StreamSynthesizer midiStreamSynthesizer;

    void Awake()
    {
        midiStreamSynthesizer = new StreamSynthesizer(44100, 2, buf
ferSize, 40);
        sampleBuffer = new float[midiStreamSynthesizer.BufferSize];

        midiStreamSynthesizer.LoadBank(bankFilePath);

        midiSequencer = new MidiSequencer(midiStreamSynthesizer);
        midiSequencer.LoadMidi(midiFilePath, false);
        //These will be fired by the midiSequencer when a song play
s. Check the console for messages
        //midiSequencer.NoteOnEvent += new MidiSequencer.NoteOnEven
tHandler(MidiNoteOnHandler);
        //midiSequencer.NoteOffEvent += new MidiSequencer.NoteOffEv
entHandler(MidiNoteOffHandler);
    }
void Update()
    {

    }
public void StartPlayingKey(int channel, int note, int volume, int
instrumentNumber)
    {
        //UnityEditor.EditorUtility.DisplayDialog("A", volume.ToStr
ing(), "A");
        midiStreamSynthesizer.NoteOn(channel, note, volume, instrum
entNumber);
    }
```

Code 5: Synthesizer.cs (Con.)

```
private void OnAudioFilterRead(float[] data, int channels)
    {
        //This uses the Unity specific float method we added to get
 the buffer
        midiStreamSynthesizer.GetNext(sampleBuffer);
        for (int i = 0; i < data.Length; i++)
        {
            data[i] = sampleBuffer[i] * gain;
        }
    }
    public void MidiNoteOnHandler(int channel, int note, int veloci
ty)
    {
        Debug.Log("NoteOn: " + note.ToString() + " Velocity: " +
velocity.ToString());
    }
    public void MidiNoteOffHandler(int channel, int note)
    {
        Debug.Log("NoteOff: " + note.ToString());
    }
}
```

Code 6: AutoPlay.cs (Con.)

```
using System;
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class AutoPlay : MonoBehaviour {

    public string filePath; //"Assets\Scripts\text_file.txt"
    private Synthesizer synthesizerScript;
    private GameObject pianoKeyObj;
    private PianoKey pianoKey;
    private GameObject pianoKeyObj_nxt;
    private PianoKey pianoKey_nxt;
    private Slider seekbar;
    private Color defColor;
    private Color defColor_nxt;
```

Code 6: AutoPlay.cs (Con.)

```
string text;
    string[] lines;
    string line;
    string[] msg;
    string[] msg_nxt;
    int currentLine = 0;
    int nLine;

    int channel;
    int note;
    int nxt_note;
    int volume;
    int instrumentNumber;
    float Duration;
    float time;

    float tt = 0;

    private bool IsPlaying;
    public bool tutorMode;

    void Start () {

        synthesizerScript = FindObjectOfType<Synthesizer>() as Synt
hesizer;
        text = System.IO.File.ReadAllText(@filePath);
        lines = System.IO.File.ReadAllLines(@filePath);

        seekbar = GameObject.FindWithTag("Seekbar").GetComponent<Sl
ider>();
        seekbar.minValue = 0;
        seekbar.maxValue = lines.Length-1;

        IsPlaying = false;
        tutorMode = false;
    }
void Update() {

    }
```

Code 6: AutoPlay.cs (Con.)

```
private void CurrentLine()
    {

        msg = lines[currentLine].Split(',');
        msg_nxt = lines[currentLine+1].Split(',');

        seekbar.value = currentLine;

        channel = Int32.Parse(msg[0]);
        note = Int32.Parse(msg[1]);
        nxt_note = Int32.Parse(msg_nxt[1]);
        volume = Int32.Parse(msg[2]);
        instrumentNumber = Int32.Parse(msg[3]);
        Duration = float.Parse(msg[4]);
        time = float.Parse(msg[5]);

        pianoKeyObj = GameObject.FindWithTag(msg[1]);
        pianoKey = pianoKeyObj.GetComponent<PianoKey>();

        pianoKeyObj_nxt = GameObject.FindWithTag(msg_nxt[1]);
        pianoKey_nxt = pianoKeyObj_nxt.GetComponent<PianoKey>();

        if (pianoKey.keyType == Parameters.KeyType.WHITE)
        {
            defColor = Color.white;
        }
        else if (pianoKey.keyType == Parameters.KeyType.BLACK)
        {
            defColor = Color.black;
        }

        if (pianoKey_nxt.keyType == Parameters.KeyType.WHITE)
        {
            defColor_nxt = Color.white;
        }
        else if (pianoKey_nxt.keyType == Parameters.KeyType.BLACK)
        {
            defColor_nxt = Color.black;
        }

    }
```

Code 6: AutoPlay.cs (Con.)

```
IEnumerator Play()
    {
        int ch = channel;
        int nt = note;
        int vol = volume;
        int ins = instrumentNumber;
        if (!IsPlaying)
        {
            StopSound(ch, nt);
            yield return new WaitForSeconds(5);
            Debug.Log("Hi");
            StopAllCoroutines();
        }
        else
        {
            if (tutorMode)
            {
                PlaySound(ch, nt, vol, ins);
                yield return new WaitForSeconds(Duration);
                StopSound(ch, nt);
                pianoKey.GetComponent<Renderer>().material.color =
defColor_nxt;
            }
            else
            {
                pianoKey.GetComponent<Renderer>().material.color =
Color.cyan;
                PlaySound(ch, nt, vol, ins);
                yield return new WaitForSeconds(Duration);
                StopSound(ch, nt);
                pianoKey.GetComponent<Renderer>().material.color =
defColor;
            }
        }

    }
IEnumerator Timeline()
    {
        CurrentLine();

        if (channel >= 128)
        {
            Debug.Log("End of Line");
            StopCoroutine(Timeline());
            yield break;
        }
```

Code 6: AutoPlay.cs (Con.)

```
if (tutorMode)
        {
            Debug.Log("tutorMode");
            pianoKey.GetComponent<Renderer>().material.color = Colo
r.yellow;
            pianoKey_nxt.GetComponent<Renderer>().material.color =
Color.green;
            //yield return StartCoroutine(WaitForKeyDown(KeyCode.Q)
);
            yield return StartCoroutine(WaitForPressOn());
            StartCoroutine(Play());
            yield return new WaitForSeconds(time);
            currentLine += 1;
            StartCoroutine(Timeline());
        }
        else
        {
            StartCoroutine(Play());
            yield return new WaitForSeconds(time);
            currentLine += 1;
            StartCoroutine(Timeline());
        }
    }

    IEnumerator WaitForKeyDown(KeyCode keyCode)
    {
        while (!Input.GetKeyDown(keyCode))
            yield return null;
    }

    IEnumerator WaitForPressOn()
    {
        Debug.Log("Pressed " + pianoKeyObj.GetComponent<OnKey>().
pressed);
        while (!pianoKeyObj.GetComponent<OnKey>().pressed)
        {
            Debug.Log("Pressed " + pianoKeyObj.GetComponent<OnKey>
().pressed);
            yield return null;
        }
    }

    public void Play_or_Pause()
    {
        if (IsPlaying)
        {
            //pause
            IsPlaying = false;
            GameObject.FindWithTag("PlayOrPause").GetComponent<Text
>().text = "Play";
        }
```

Code 6: AutoPlay.cs (Con.)

```
else
        {
            //PLay
            StopAllCoroutines();
            IsPlaying = true;
            StartCoroutine(Timeline());
            GameObject.FindWithTag("PlayOrPause").GetComponent<Text
>().text = "Pause";
        }
    }

    public void ResetLine()
    {
        currentLine = 0;
        seekbar.value = currentLine;
        Debug.Log("Reseted!");
    }

    public void SeekBar()
    {
        currentLine = (int)seekbar.value;
        //Debug.Log("CurrentLine = " + currentLine);
    }

    public void ActiveTutorialMode(bool active)
    {
        tutorMode = active;
        Debug.Log(tutorMode);
    }

    private void PlaySound(int channel, int note, int volume, int
instrumentNumber)
    {
        synthesizerScript.StartPlayingKey(channel, note, volume,
instrumentNumber);
    }

    private void StopSound(int channel, int note)
    {
        synthesizerScript.StopPlayingKey(channel, note);
    }
}
```