**บทที่ 1**

**บทนำ**

* 1. **ความเป็นมาและความสำคัญของปัญหา**

ปัจจุบันเกมบนระบบปฏิบัติการแอนดรอยด์มีจำนวนมาก และเกมรูปแบบ 3 มิตินั้นมีจำนวนน้อย เนื่องจากใช้เวลาในการพัฒนานาน โดยเฉพาะส่วนการแสดงผล 3 มิติ ต้องใช้ภาษาโอเพนจีแอล อีเอส ซึ่งมีความซับซ้อนมาก จึงจำเป็นต้องนำเฟรมเวิร์คสำหรบพัฒนาเกมเข้ามาช่วยในการพัฒนา ซึ่งเฟรมเวิร์คสำหรับการพัฒนาเกมนั้นจะประกอบไปด้วยชุดคำสั่งหลายๆ ชุด ที่ให้ผู้ใช้เฟรมเวิร์คสามารถเรียกใช้งาน โดยแบ่งเป็นส่วนต่างๆ ที่เกี่ยวข้องกับการพัฒนาเกม เช่น ระบบกราฟิก ระบบเสียง หรือระบบฐานข้อมูล เป็นต้น ซึ่งเฟรมเวิร์คจะคอยจัดการความยุ่งยากที่ผู้ใช้เฟรมเวิร์คจะต้องทำให้ง่ายขึ้นด้วยรูปแบบที่เฟรมเวิร์คกำหนด เช่น ผู้ใช้เฟรมเวิร์คเรียกใช้คำสั่งโหลดโมเดลจากไฟล์ .FBX เฟรมเวิร์คจะทำการอ่านข้อมูล และสร้างเป็นโมเดล 3 มิติของเกมอัตโนมัติ เป็นต้น ทำให้ลดระยะการพัฒนาเกมลงไปได้มาก ด้วยเหตุนี้เฟรมเวิร์คสำหรับการพัฒนาเกมจึงมีความสำคัญเป็นอย่างมาก

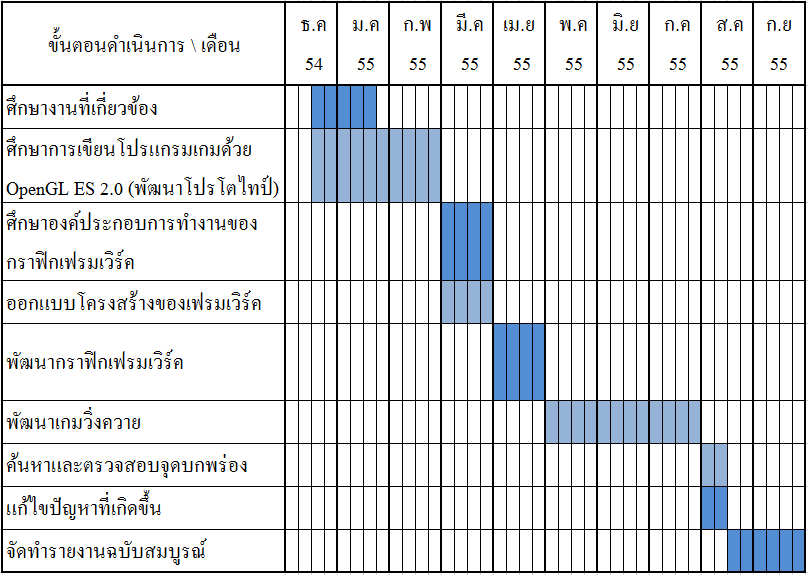
เฟรมเวิร์คบนระบบปฏิบัติการแอนดรอยด์มีจำนวนน้อย ที่พบในปัจจุบันหลายตัวไม่มีการปรับปรุงมาเป็นเวลานานแล้วอย่างเช่น Dwarf Framework หรือ catcake เป็นต้น จึงไม่เหมาะแก่การนำมาใช้ เนื่องระบบปฏิบัติการแอนดรอยด์มีการปรับปรุงอยู่ตลอดเวลา จึงจำเป็นต้องใช้เฟรมเวิร์คที่สามารถรองรับระบบใหม่ๆของระบบปฏิบัติการแอนดรอยด์ได้ เช่น ระบบปฏิบัติการแอนดรอยด์เวอร์ชันใหม่รองรับโอเพนจีแอล อีเอส เวอร์ชัน 2.0 ซึ่งในระบบปฏิบัติการแอนดรอยด์เวอร์ชันเก่าจะไม่รองรับ เป็นต้น และเฟรมเวิร์คส่วนใหญ่จะใช้ไฟล์นามสกุล .3DS .OBJ และ .MD2 แต่ไฟล์นามสกุลเหล่านี้ถูกพัฒนาขึ้นเป็นเวลานานแล้ว ซึ่งในปัจจุบันไฟล์นามสกุล .FBX ได้ถูกพัฒนาขึ้นมาเพื่อใช้เป็นสื่อกลางระหว่างโปรแกรม 3 มิติต่างๆ เนื่องจากโปรแกรม 3 มิติมีจำนวนมาก ทำให้นามสกุลของไฟล์มีความแตกต่างกัน เช่น Autodesk 3ds Max จะเป็นนามสกุล .3DS และ Wavefront จะเป็นนามสกุล .OBJ เป็นต้น นักพัฒนาส่วนใหญ่จึงหันมาใช้ไฟล์นามสกุล .FBX เป็น

จำนวนมาก แต่ไม่สามารถนำมาใช้ร่วมกับเฟรมเวิร์คสำหรับพัฒนาเกมบนแอนดรอยด์ที่มีอยู่ได้ ส่งผลให้การพัฒนาเกมทำได้ช้าลง เนื่องจากต้องทำการพัฒนาเฟรมเวิร์คที่รองรับไฟล์นามสกุล .FBX ขึ้นมาใหม่

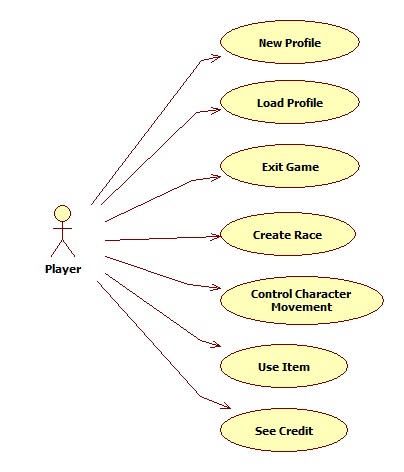
ทางผู้พัฒนาได้มีความคิดที่จะพัฒนาเฟรมเวิร์คสำหรับการพัฒนาเกมบนระบบปฏิบัติการแอนดรอยด์ ด้วยจาวา และภาษาโอเพนจีแอล อีเอส ที่ช่วยให้ผู้พัฒนาเกมลดระยะเวลาในการพัฒนา โดยจะเน้นในส่วนส่วนของกราฟิกในส่วน การโหลดโมเดล 3 มิติจากไฟล์ .FBX การจัดการโมเดล 3 มิติโดยที่ผู้ใช้เฟรมเวิร์คไม่ต้องทำในส่วนของโอเพนจีแอล อีเอส การจัดการภาพเคลื่อนไหวของโมเดล 3 มิติ การควบคุมแหล่งกำเนิดแสงและการควบคุมกล้อง และการตรวจสอบการชนของวัตถุ ด้วยรูปแบบที่เฟรมเวิร์คกำหนด และจะทำการพัฒนาเกมหนึ่งเกม ชื่อว่า “เกมวิ่งควาย ฮุยเลฮุย” ที่นำเฟรมเวิร์คเข้ามาใช้ในการพัฒนา เพื่อทดสอบระบบการทำงานของเฟรมเวิร์ค

* 1. **วัตถุประสงค์**
     1. สร้างเฟรมเวิร์คสำหรับพัฒนาเกม 3 มิติบนระบบปฏิบัติการแอนดรอยด์
     2. สร้างเฟรมเวิร์คที่รองรับการใช้งานนามสกุลไฟล์ .FBX
     3. สร้างเฟรมเวิร์คที่สามารถนำไปใช้งานได้อย่างแพร่หลาย
  2. **ขั้นตอนการดำเนินการ**
     1. ศึกษางานที่เกี่ยวข้อง
     2. ศึกษาการเขียนโปรแกรมด้วย OpenGL ES
     3. ศึกษาองค์ประกอบการทำงานของกราฟิกเฟรมเวิร์ค
     4. ออกแบบโครงสร้างของเฟรมเวิร์ค
     5. พัฒนากราฟิกเฟรมเวิร์ค
     6. พัฒนาเกมวิ่งควาย
     7. ตรวจสอบจุดบกพร่องและแก้ไขปัญหา
     8. จัดทำรายงานฉบับสมบูรณ์

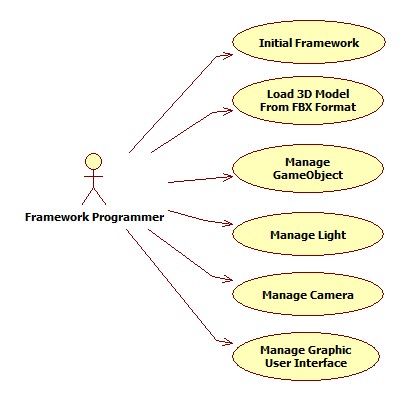
**ตารางที่ 1.1** ตารางเวลาการดำเนินการ

****

* 1. **ขอบเขตการดำเนินงาน**



**รูปที่ 1.2** ไดอะแกรมแสดงสิ่งที่ผู้เล่นสามารถทำได้



**รูปที่ 1.3** ไดอะแกรมแสดงสิ่งที่ผู้ใช้เฟรมเวิร์คสามารถทำได้

* + 1. **New Profile**

ผู้เล่นสามารถสร้างโปรไฟล์สำหรับการเก็บสถิติการแข่งในแต่ละครั้งได้

ผู้เล่นสามารถมีโปรไฟล์ได้โปรไฟล์เดียว

การสร้างโปรไฟล์ใหม่ ถ้ามีโปรไฟล์เก่าอยู่แล้วจะทำการลบโปรไฟล์เก่าทิ้ง

ในการเล่นเกมครั้งแรกผู้เล่นต้องสร้างโปรไฟล์ก่อน

* + 1. **Load Profile**

ผู้เล่นจะต้องมีโปรไฟล์ก่อนถึงจะเรียกใช้โปรไฟล์ที่บันทึกไว้ได้

ผู้เล่นสามารถเรียกใช้โปรไฟล์ที่เคยบันทึกไว้จากการเล่นได้

* + 1. **Exit Game**

ผู้เล่นสามารถออกจากเกมได้

เมื่อผู้เล่นออกจากเกมระบบจะทำการบันทึกสถิติการเล่นล่าสุดอัตโนมัติ

* + 1. **Create Race**

ผู้เล่นสามารถสร้างการแข่งได้ โดยจะต้องเลือกฉากที่จะทำการแข่ง จำนวนรอบและจำนวนของคู่แข่งที่เป็นปัญญาประดิษฐ์

ฉากสำหรับการแข่งสามารถเลือกได้ 1 ฉาก

คู่แข่งที่เป็นปัญญาประดิษฐ์จำกัดไว้ที่ 3 คู่แข่ง

จำนวนรอบกำหนดได้สูงสุด 9 รอบ

* + 1. **Control Character Movement**

ผู้เล่นสามารถบังคับตัวละครได้ในฉากการแข่งเท่านั้น

ผู้เล่นสามารถบังคับตัวละครได้ 3 ทิศทางคือ เลี้ยวซ้าย เลี้ยวขวา และถอยหลัง

ผู้เล่นสามารถบังคับตัวละครให้หยุดเคลื่อนไหวได้

* + 1. **Use Item**

ผู้เล่นสามารถใช้ไอเทมได้เฉพาะในฉากการแข่งขันเท่านั้น

ผู้เล่นไม่สามารถใช้ไอเทมได้ถ้าไม่มีไอเทมอยู่ในช่องไอเทม

เมื่อผู้เล่นใช้ไอเทมในขณะที่มีไอเทมกำลังแสดงผลอยู่จะถูกยกเลิกและแสดงผลไอเทมใหม่ทันที

* + 1. **See Credit**

ผู้เล่นสามารถดูข้อมูลคณะผู้จัดทำได้

* + 1. **Initial Framework**

ผู้ใช้เฟรมเวิร์คสามารถกำหนดค่าเริ่มต้นการทำงานของโปรแกรม เช่น การแสดงผลโมเดล 3 มิติ การแสดงส่วนติดต่อกับผู้ใช้ กล้อง หรือ แหล่งกำเนิดแสง เป็นต้น

ผู้ใช้เฟรมเวิร์คสามารถกำหนดการทำงานต่างๆโดยใช้รูปแบบที่เฟรมเวิร์คกำหนด

* + 1. **Load 3D Model From FBX Format**

ผู้ใช้เฟรมเวิร์คสามารถโหลดโมเดล 3 มิติจากไฟล์ .FBX ได้

ไฟล์ FBX จะต้อง ส่งออกจากโปรแกรม Autodesk 3ds Max 2012 เท่านั้น

ไฟล์ FBX จะต้องมีการตั้งค่า Up direction เป็น Z Axis เท่านั้น

ไฟล์ FBX จะต้องเป็นเวอร์ชัน 2012 เท่านั้น

ไฟล์ FBX ที่มีการ Rigging จะต้องทำการ Rig จาก Skin Modifier เท่านั้น

ไฟล์ FBX จะทำการแอนิเมชันแบบ Forward Kinematic เท่านั้น

ไฟล์ FBX ที่มีคีย์เฟรมแอนิเมชันจะต้องเป็นแบบ Frame-by-Frame เท่านั้น

ไฟล์ FBX ไม่รองรับโพลีกอนเป็นกลุ่มๆ ใน FBX 1 ไฟล์

* + 1. **Manage Game Object**

ผู้ใช้เฟรมเวิร์คสามารถควบคุม การเปลี่ยนตำแหน่ง การหมุนและการปรับอัตราส่วนของ Game Object ได้

ผู้ใช้เฟรมเวิร์คจะต้องควบคุม Game Object ผ่าน Game Object Manager

* + 1. **Manage Light**

ผู้ใช้เฟรมเวิร์คสามารถสร้างแหล่งกำเนิดแสงผ่านเฟรมเวิร์คได้

ผู้ใช้เฟรมเวิร์คสามารถตั้งค่าสีของแสดงได้

ผู้ใช้เฟรมเวิร์คสามารถตั้งค่าความเข้มแสงได้

* + 1. **Manage Camera**

ผู้ใช้เฟรมเวิร์คสามารถควบคุมกล้องเพื่อฉายภาพวัตถุในมุมมองต่างๆ

ผู้ใช้เฟรมเวิร์คสามารถปรับระยะการมองเห็นในแนวลึก (Depth) ได้

* + 1. **Manage Graphic User Interface**

ผู้ใช้เฟรมเวิร์คสามารถกำหนดข้อความเพื่อแสดงบนหน้าจอได้

ผู้ใช้เฟรมเวิร์คสามารถสร้างปุ่มโต้ตอบกับผู้ใช้งานด้วยไฟล์ภาพ .png ได้

* 1. **อุปกรณ์และเครื่องมือที่ใช้พัฒนา**
     + 1. คอมพิวเตอร์ 1 เครื่อง
       2. แท็บเล็ต Samsung Galaxy Tab P1000 1 เครื่อง
       3. โปรแกรม Autodesk 3ds Max 2012
       4. โปรแกรม Eclipse IDE
       5. โปรแกรม Nero Wave Editor
  2. **ประโยชน์ที่คาดว่าจะได้รับ**
     + 1. ได้ศึกษาการเขียนโปรแกรมเกมด้วยภาษาระดับต่ำ
       2. ได้ความรู้ในการพัฒนาเกมบนระบบปฏิบัติการแอนดรอยด์
       3. ได้ทักษะการวางแผนงาน
       4. ได้เฟรมเวิร์คสำหรับพัฒนาเกมที่สามารถนำไปใช้ได้ในอนาคต

**บทที่ 2**

**ทฤษฎี หลักการและงานที่เกี่ยวข้อง**

ทฤษฎีที่เกี่ยวข้องกับโครงงานมีดังต่อไปนี้

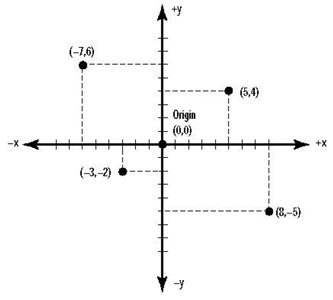
* 1. **ทฤษฎีที่เกี่ยวข้อง**
     1. **คอมพิวเตอร์กราฟิก (Computer Graphic)**
        + 1. **ระบบพิกัดฉาก (Cartesian Coordinate System) [1]**

ระบบพิกัดในคอมพิวเตอร์กราฟิกนั้นมีหลากหลายรูปแบบ ระบบพิกัดคาร์ทีเชียนเป็นหนึ่งระบบที่มีความสำคัญมาก ซึ่งแบ่งออกเป็น 2 แบบ คือ

* 1. ระบบพิกัดคาร์ทีเชียน 2 มิติ

ในระบบ 2 มิตินั้นประกอบไปด้วยแกน 2 แกน คือ แกน X และ แกน Y โดย แกน X จะบอกพิกัดในแนวนอน (ความกว้าง) ส่วนแกน Y จะบอกพิกัดในแนวตั้ง (ความสูง) จุดกำเนิดของแกนทั้งสอง อยู่ที่ X=0, Y=0 โดยทั้งสองแกนมีความยาวทางบวกและทางลบได้เป็นอนันต์ ดังรูปที่ 2.1

จากภาพ 2.1 จะเห็นว่า แกน X ค่าที่เป็นบวกจะอยู่ทางขวา ค่าลบอยู่ทางซ้าย ส่วนแกน Y ค่าบวกจะอยู่ด้านบน ค่าลบอยู่ด้านล่าง



**รูปที่ 2.1** แกนพิกัด 2 มิติ

* 1. ระบบพิกัดคาร์ทีเชียน 3 มิติ

ระบบ 3 มิตินั้นจะเพิ่มแกน อีกหนึ่งแกน จาก 2 แกนในระบบ 2 มิติ นั่นคือ แกน Z โดยแกน Z จะบอกพิกัดในแนวลึก โดยจุดกำเนิดอยู่ที่ z=0 และแกน Z มีความยาวทางบวกและทางลบได้เป็นอนันต์ ดังรูปที่ 2.2



**รูปที่ 2.2** แกนพิกัด 3 มิติ

* + - * 1. **การเขียนโปรแกรม Shader (Shader Programming) [13]**

โปรแกรม Shader ถูกพัฒนาขึ้นมา เมื่อเกมในในปัจจุบันได้ก้าวเข้าสู่ยุคของเกมสามมิติที่ภาพภายในเกมมีความสมจริงมาก จนบางครั้งแยกไม่ออกเลยว่าเป็นภาพจริงหรือถูกสร้างขึ้นมา ซึ่งความสมจริงของภาพนั้นมาจากความสามารถของโปรแกรม Shader นั่นเอง

โปรแกรม Shader จะมีรูปแบบการทำงานโดยการเขียนโค้ดเพื่อสั่งให้ GPU (Graphics Processing Unit หรือ การ์ดจอ) ทำการแสดงผลภาพตามโค้ดที่ป้อนเข้ามา ซึ่งปัจจุบันแบ่งรูปแบบการสั่งการให้ GPU ประมวลผลได้ 3 แบบ คือ

* Vertex Shader
* Pixel Shader (Fragment Shader)
* Geometry Shader

ภาษาที่ใช้ในการเขียนโปรแกรม Shader มีทั้งหมด 3 ภาษาดังนี้

1. GLSL (OpenGL Shading Language)

ใช้พื้นฐานของภาษา C ในการเขียนคำสั่งต่างๆ ซึ่งถูกพัฒนาโดยกลุ่ม OpenGL ARB และถูกพัฒนาขึ้นเพื่อให้สามารถรองรับกับความละเอียดของกราฟิกที่เพิ่มมากขึ้นในปัจจุบัน โดยภาษานี้จะสามารถแสดงผลได้บน OpenGL เท่านั้น

1. HLSL (High Level Shading Language)

เป็นภาษา Shading ที่อยู่ในกรรมสิทธ์ของบริษัท Microsoft โดยใช้โปรแกรมชื่อว่า Microsoft Direct3D API ซึ่ง HLSL มีลักษณะคล้ายกับ OpenGL (GLSL) และ NVIDIA (Cg) มาก แต่ HLSL แสดงผลได้ด้วย DirectX เท่านั้น

1. Cg (Programming Language)

เป็นภาษา Shading ที่ถูกพัฒนาโดย NVIDIA ซึ่งได้รับการอุปถัมภ์จากบริษัท Microsoft ทำให้มีลักษณะคล้ายกับ HLSL แต่ Cg สามารถแสดงผลได้ทั้ง OpenGL และ DirectX ซึ่งรูปแบบของภาษาจะมีพื้นฐานจากภาษา C

* + - * 1. **ซีนกราฟ (Scene Graph)**

ซีนโหนด (Scene Node)ใช้ในการจัดการวัตถุภายในเกม ที่สามารถเคลื่อนย้ายได้ในฉากของเกม โดยข้อมูลการเคลื่อนย้าย (การเปลี่ยนตำแหน่ง การหมุนและการปรับอัตราส่วน) จะถูกเก็บในรูปแบบของเมตริกซ์

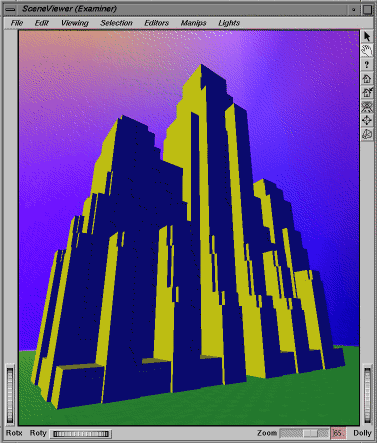
ซีนกราฟ (Scene Graph) คือ การเก็บโหนด (Node) ในรูปของกราฟต้นไม้ ซึ่งแต่ละโหนดสามารถมีโหนดลูก (Child Node) ได้มากมาย แต่จะมีโหนดแม่ (Parent Node) ได้เพียงโหนดเดียวเท่านั้น เมื่อโหนดแม่มีการเปลี่ยนแปลงต่างๆ จะทำให้โหนดลูกมีการเปลี่ยนแปลงไปด้วย เช่น การเปลี่ยนตำแหน่ง การหมุนวัตถุและการปรับอัตราส่วน ของวัตถุ เป็นต้น

* + - * 1. **แสงในคอมพิวเตอร์กราฟิก [6]**

แสงในคอมพิวเตอร์กราฟิก มีทั้งหมด 4 ประเภท ซึ่งแต่ละประเภทจะมีลักษณะที่แตกต่างกันออกไปดังนี้

* 1. Directional Light

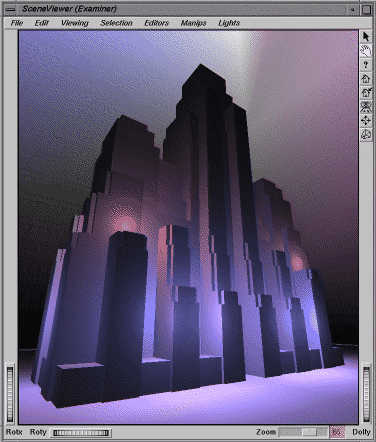
แสงประเภทนี้เปรียบเสมือนแสงจากดวงอาทิตย์ ซึ่งแสงมีระยะเป็นอนันต์ ปริมาณความเข้มของแสงจะเท่ากันในทุกๆตำแหน่ง และมีทิศทางของแสงไปในทางเดียวกันดัง รูปที่ 2.3

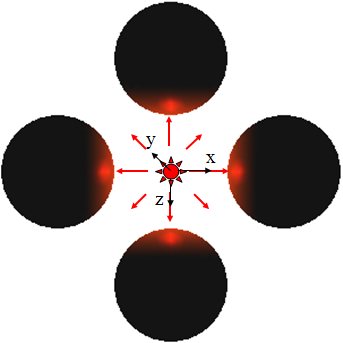


**รูปที่ 2.3** Directional Light

* 1. Point Light

แสงประเภทนี้เปรียบเสมือนแสงเทียน ซึ่งแสงจะกระจายไปรอบทิศทาง และแสงมีระยะที่จำกัด ยิ่งระยะไกลมากเท่าใด ปริมาณความเข้มแสงจะยิ่งน้อย ดังตัวอย่างในรูปที่ 2.4 และ 2.5

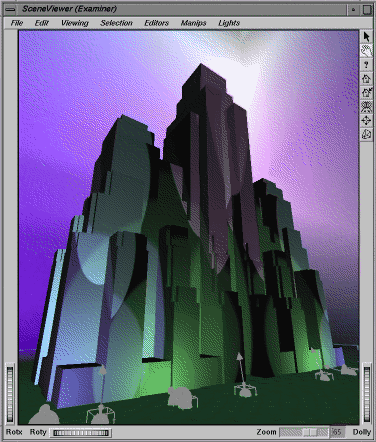


****

**รูปที่ 2.4** Point Light

* 1. Spot Light

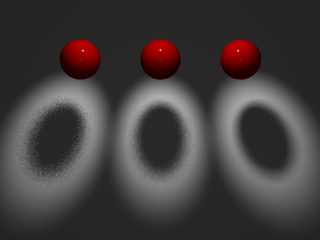
แสงประเภทนี้เปรียบเสมือนแสงจากไฟฉาย ซึ่งลำแสงจะมีทิศทางและขอบเขตที่จำกัด มีลักษณะของแสงเป็นรูปกรวย ยิ่งระยะไกลมากเท่าใด ปริมาณความเข้มแสงจะยิ่งน้อย ดังรูปตัวอย่างที่ 2.6



**รูปที่ 2.5** Spot Light

* 1. Area Light

แสงประเภทนี้เปรียบเสมือนแสงจากแผงไฟ ซึ่งแสงจะพุ่งไปยังทุกๆวัตถุที่แสงเคลื่อนผ่าน รูปที่ 2.7



**รูปที่ 2.6** Area Light

* + 1. **ปัญญาประดิษฐ์ (Artificial Intelligent : AI) [2]**

ปัญญาประดิษฐ์หรือ เอไอ (AI) เป็นการสร้างความคิดให้กับสิ่งไม่มีชีวิต โดยสามารถแบ่งได้เป็น 4 ประเภท สามารถมองในแบบ 2 มิติ ได้แก่ นิยามที่เน้นระบบที่เลียนแบบมนุษย์ กับ นิยามที่เน้นระบบที่มีเหตุผล และ นิยามที่เน้นความคิด กับ นิยามที่เน้นการกระทำ ซึ่งแต่ละนิยามข้างต้นมีความหมายดังนี้

1. ระบบความคิดเหมือนมนุษย์

คือ การพยายามที่จะทำให้เครื่องจักรมีความคิดที่เหมือนกับมนุษย์ แต่ก่อนที่จะทำให้เครื่องคิดเหมือนมนุษย์ได้ ต้องทำการวิเคราะห์วิธีการคิดของมนุษย์เสียก่อน ซึ่งปัจจุบันก็ยังไม่ทราบแน่ชัดว่ามนุษย์มีระบบความคิดอย่างไร

1. ระบบการกระทำที่เหมือนมนุษย์

คือ การให้เครื่องจักรทำงานในสิ่งมนุษย์ทำได้ สามารถสื่อสารด้วยภาษาที่มนุษย์ใช้ และมีการเรียนรู้โดยการตรวจจับรูปแบบเหตุการณ์ที่เกิดขึ้นได้ สามารถปรับตัวเข้ากับสิ่งแวดล้อมได้

1. ระบบที่คิดอย่างมีเหตุผล

คือ การสร้างเครื่องจักรที่ใช้หลักตรรกศาสตร์ในการคิดหาคำตอบอย่างถูกต้อง

1. ระบบที่กระทำอย่างมีเหตุผล

คือ การทำให้เครื่องจักทำงานอย่างมีเป้าหมายเช่น ให้ระบบขับรถอัตโนมัติ ซึ่งมีเป้าหมายที่จะไปด้วยเส้นทางที่สั้นที่สุด เป็นต้น

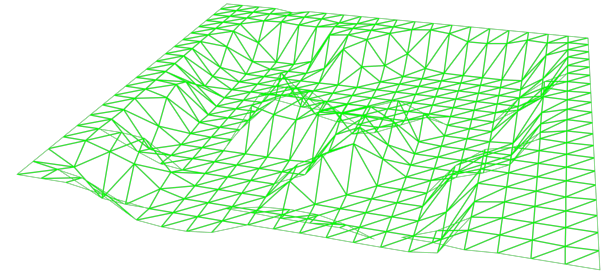
* 1. **หลักการที่เกี่ยวข้อง**
     1. **การสร้างภูมิประเทศด้วยไฮต์แมพ (Heightmap)**

การสร้างภูมิประเทศ (Terrain) มีหลายวิธีเช่น สร้างจากโปรแกรมสำหรับงานสามมิติและเซฟเป็นไฟล์นามสกุลต่างๆ และสร้างจากภาพสองมิติที่มีค่าสีเป็นขาวดำหรือเรียกว่า ไฮต์แมพเป็นต้น

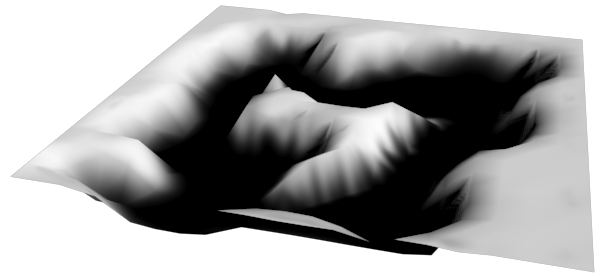
การสร้างภูมิประเทศจากไฮต์แมพมีข้อดีคือสามารถลดขั้นตอนการตรวจสอบการชนของวัตถุต่างๆกับภูมิประเทศได้ โดยใช้ค่าสีในการคำนวณ ซึ่งค่าสีสามารถบอกความสูงของภูมิประเทศได้ โดยใช้สีขาวแทนความสูงที่สูงที่สุด ส่วนสีดำจะเป็นส่วนที่ลึกที่สุด เช่น ภูเขาของภูมิประเทศนี้มีความสูงอยู่ 50 หน่วย ส่วนที่เป็นสีขาวจะมีความสูง 50 หน่วย ถ้าเป็นสีเทาจะมี 25 หน่วย (คำนวณจากอัตราส่วนระหว่างความสูงกับค่าสี) และถ้าเป็นสีดำจะมีความสูงเป็น 0 หน่วย เป็นต้น โดยรูปที่ 2.8 จะเป็นรูปตัวอย่างของไฮต์แมพและรูปที่ 2.9, 2.10, 2.11 จะเป็นภูมิประเทศที่ถูกสร้างขึ้นจากไฮต์แมพในโลกของคอมพิวเตอร์กราฟิก

****

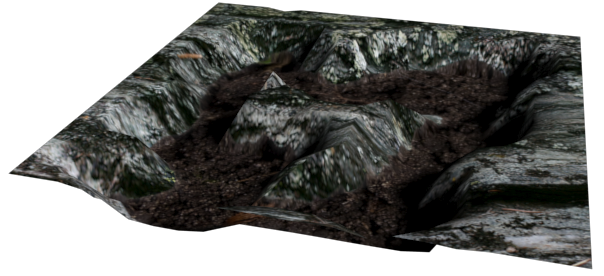
**รูปที่ 2.7** HeightmapSource

****

**รูปที่ 2.8** การสร้างภูมิประเทศจากไฮต์แมพ (แสดงผลแบบเส้น)

****

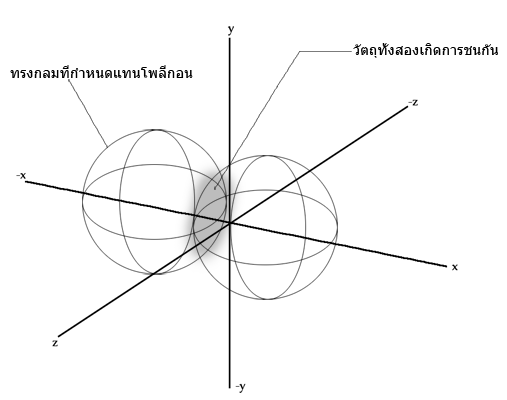
**รูปที่ 2.9** การสร้างภูมิประเทศจากไฮต์แมพ (แสดงผลแบบโพลีกอน)

****

**รูปที่ 2.10** การสร้างภูมิประเทศจากไฮต์แมพ (แสดงผลแบบโพลีกอนโดยใส่เทกเจอร์)

* + 1. **การตรวจสอบการชน [3]**

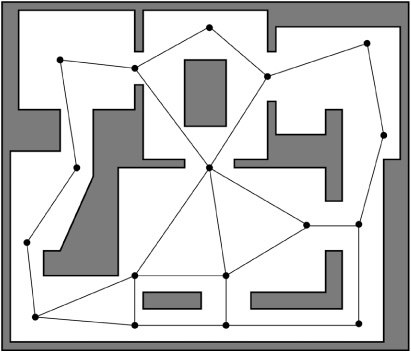
การตรวจสอบการชนส่วนมากจะพบในวีดีโอเกม โดยนำหลักการพีชคณิตเชิงเส้นและเรขาคณิตเชิงคำนวณ เข้ามาใช้ในการตรวจสอบการชนระหว่างวัตถุตั้งแต่ 2 วัตถุขึ้นไป ซึ่งวัตถุในคอมพิวเตอร์จะถูกมองว่าเป็นรูปหลายเหลี่ยม (Polygons) การที่จะคำนวณนั้นทำได้ยาก จึงมีการกำหนดทรงกลมขึ้นมาล้อมรอบวัตถุหลายเหลี่ยมในขนาดที่ใกล้เคียงกับวัตถุหลายเหลี่ยมนั้นๆ และทำการตรวจสอบว่าจากทรงกลมนั้นว่ามีการทับซ้อนกันหรือไม่ สามารถคำนวณได้จากระยะห่างระหว่างจุดศูนย์กลางของทรงกลมทั้งสองอันว่าได้ผลลัพธ์น้อยกว่ารัศมีของทั้งสองบวกกันหรือไม่ ถ้าคำตอบได้ค่าน้อยกว่าแสดงว่ามีการชนกัน) ดังรูปที่ 2.3 ซึ่งสามารถคำนวณสมการดังนี้ ((x1-x2)2 + (y1-y2)2 + (z1-z2)2 ≤ r12 + r22) แต่การทำแบบนี้ยังไม่ละเอียดมาก ถ้าต้องการความแม่นยำสูงกว่านี้ คือการแบ่งทรงกลมเป็นหลายๆก้อนสำหรับหนึ่งวัตถุ แล้วนำมาคำนวณจะได้ผลลัพธ์ที่แม่นยำมากกว่า



**รูปที่ 2.11** การชนกันของวัตถุ

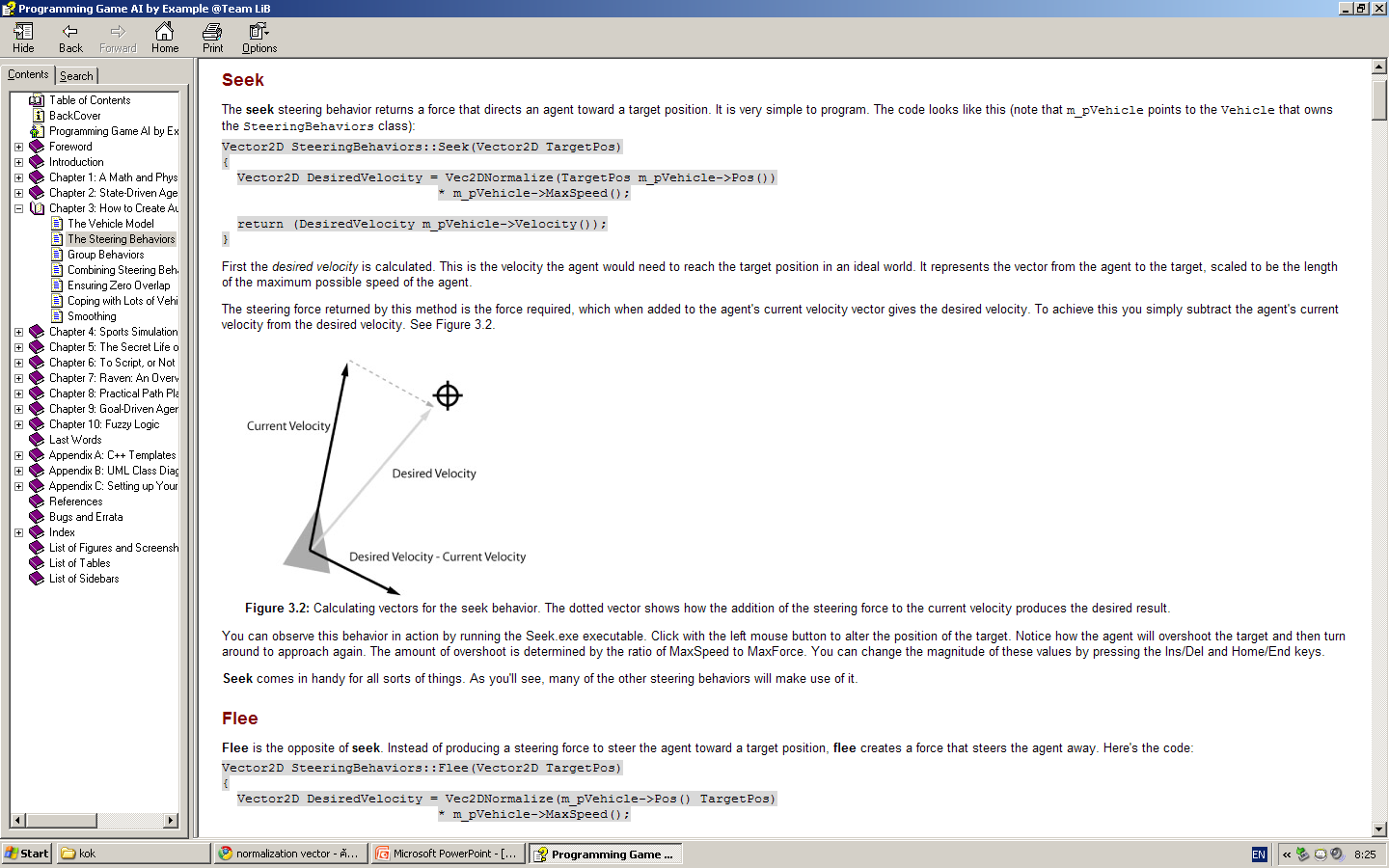
* + 1. **การค้นหาเส้นทางและเคลื่อนที่ไปตามเส้นทางของปัญญาประดิษฐ์ [4]**

การเคลื่อนที่ไปตามเส้นทางนั้นจะต้องมีเส้นทางก่อน ซึ่งสามารถสร้างเส้นทางได้หลายวิธี โดยวิธีที่นำมาใช้คือ Point of Visibility การกำหนดจุดหลักที่คิดว่าปัญญาประดิษฐ์สามารถเคลื่อนที่ไปได้ดังรูปที่ 2.4



**รูปที่ 2.12** ตัวอย่าง Point of Visibility

หลังจากกำหนดจุดเรียบร้อย ปัญญาประดิษฐ์สามารถเคลื่อนที่ไปตามจุดที่กำหนดโดยการหาทิศทางที่จะเคลื่อนที่จากจุดที่ปัญญาประดิษฐ์อยู่กับจุดที่เป็นเป้าหมาย ดังภาพที่ 2.5 เมื่อได้ทิศทางแล้วจะทำการเคลื่อนที่ไปยังทิศทางนั้นด้วยความเร็วของปัญญาประดิษฐ์จนไปถึงจุดเป้าหมาย เมื่อถึงจุดเป้าหมายแล้วปัญญาประดิษฐ์ก็จะทำการหาทิศทางของจุดต่อไปเรื่อยๆ

****

**รูปที่ 2.13** การหาทิศทางของจุดเป้าหมายกับปัญญาประดิษฐ์

* 1. **งานที่เกี่ยวข้อง**
     1. **เฟรมเวิร์คสำหรับพัฒนาเกมบนระบบปฏิบัติการแอนดรอยด์**

เฟรมเวิร์คบนระบบปฏิบัติการแอนดรอยด์ที่พบในปัจจุบันมีดังต่อไปนี้

* + - 1. **min3D Framework**

min3D Framework เป็นเฟรมเวิร์คสำหรับกราฟิกสามมิติที่ถูกพัฒนาขึ้นเพื่อพัฒนากราฟิกสามมิติบนระบบปฏิบัติการแอนดรอยด์ ด้วยภาษา Java และ OpenGL ES โดยเปิดให้นักพัฒนาสามารถนำไปใช้ได้อย่างอิสระ

จุดเด่น

* + - เป็นเฟรมเวิร์คที่มีขนาดเล็ก
    - รองรับโมเดล 3 มิติที่มี Keyframe Animation
    - รองรับการใช้งาน Accelerometer

ข้อจำกัด

* + - รองรับ OpenGL ES 1.0, 1.1 เท่านั้น
    - รองรับโมเดลสามมิตินามสกุล .3DS, .OBJ, .MD2 เท่านั้น
      1. **Rajawali Framework**

Rajawali Framework เป็นเฟรมเวิร์คที่ถูกพัฒนาขึ้นมาโดยเน้นไปทางด้านการทำ Live Wallpaper บนระบบปฏิบัติการ์แอนดรอยด์ ซึ่งถูกพัฒนาด้วยภาษา Java และ OpenGL ES โดยเปิดให้นักพัฒนาสามารถนำไปใช้ได้อย่างอิสระ

จุดเด่น

* + - รองรับ OpenGL ES 2.0
    - ง่ายต่อการสร้าง Live Wallpaper
    - สามารถทำ Environment Mapping ได้

ข้อจำกัด

* + - รองรับโมเดลสามมิตินามสกุล .OBJ เท่านั้น
      1. **Dwarf Framework [9]**

Dwarf Framework เป็นเฟรมเวิร์คที่ถูกพัฒนาขึ้นเมื่อปี ค.ศ 2009 โดยมีเป้าหมายหลักคือใช้ในการพัฒนาเกมสามมิติ บนระบบปฏิบัติการแอนดรอยด์ แต่เฟรมเวิร์คนี้ไม่มีการพัฒนาต่อมาเป็นเวลานานแล้ว

จุดเด่น

* + - รองรับการใช้งาน Scene Graph
    - รองรับ keyframe Animation

ข้อจำกัด

* + - รองรับเพียง OpenGL ES 1.0, 1.1 เท่านั้น
    - รองรับโมเดลสามมิตินามสกุล .OBJ เท่านั้น
    - ไม่รองรับการติด texture บนโมเดลสามมิติ
      1. **Catcake [10]**

เฟรมเวิร์คนี้ถูกพัฒนาขึ้นเพื่อใช้ในการพัฒนาเกมสองมิติและสามมิติบนระบบปฏิบัติการต่างๆ โดยภาษาที่ใช้จะเป็น Java Native Code แต่เฟรมเวิร์คนี้ไม่มีการพัฒนาต่อมาเป็นเวลานานแล้ว

จุดเด่น

* รองรับการพัฒนาแบบ Cross-Platform (Windows, Android, Linux)
* รองรับการเขียนโปรแกรม Shader
* รองรับการใช้งาน Scene Graph
* รองรับ keyframe Animation
* สร้างแสงแบบ Directional Light และ Point Light ได้
* สร้าง Bounding Box ได้หลายรูปแบบเช่น Sphere, Box, Triangle เป็นต้น

ข้อจำกัด

* รองรับโมเดลสามมิตินามสกุล .3DS เท่านั้น
  + - 1. **Libgdx [11]**

เฟรมเวิร์คนี้ถูกพัฒนาขึ้นเพื่อใช้ในการพัฒนาเกมสองมิติและสามมิติบนระบบปฏิบัติการต่างๆ เช่น Windows, Linux, Android เป็นต้น ซึ่งจุดเด่นและข้อจำกัดต่างๆของ libgdx มีดังนี้

จุดเด่น

* รองรับการพัฒนาแบบ Cross-Platform (Windows, Android, Linux)
* รองรับ OpenGL ES 1.0, 1.1, 2.0
* รองรับการเขียนโปรแกรม Shader
* รองรับ keyframe Animation
* รองรับไฟล์เสียงนามสกุล .WAV, .MP3 และ OGG
* มีคำสั่งคำนวณเวกเตอร์และเมตริกซ์
* สร้าง Bounding Box ได้หลายรูปแบบเช่น Sphere, Box, Triangle เป็นต้น

ข้อจำกัด

* รองรับโมเดลสามมิตินามสกุล .OBJ, .MD5 เท่านั้น
* ไลบรารี่การพัฒนาเกมรูปแบบสามมิติแบบยังไม่สมบูรณ์
  + 1. **ไฟล์โมเดลสามมิตินามสกุล FBX**

การเก็บข้อมูลภายในไฟล์โมเดลสามมิตินามสกุล .FBX แบ่งเป็นหัวข้อหลักๆได้ 2 หัวข้อดังนี้

* + - 1. **Object properties**

Vertex Table : ตารางจุดในแกนพิกัด x, y, z ที่จะใช้ในการสร้างโมเดล 3 มิติ

Face Table : ตารางการเรียกใช้จุดจาก Vertex Table เพื่อสร้างเป็นโมเดล 3 มิติ

Normals : เก็บเวกเตอร์ปกติของทุกๆจุด

UV : เก็บข้อมูลพิกัดในการใส่เทกเจอร์ให้กับโมเดล 3 มิติ

UV Table : ตารางการเรียกใช้จุดจาก UV เพื่อใส่เทกเจอร์ให้กับโมเดล 3 มิติ

Node : เก็บตำแหน่งของโมเดล 3 มิติ และตำแหน่งของกระดูก ในแกนพิกัด x, y, z

Deformer : เก็บข้อมูลของกระดูกว่าแต่ละกระดูกควบคุมจุดใน Vertex Table ใดบ้าง

AnimationCurve : เก็บเมตริกซ์ของอนิเมชันแต่ละเฟรม

* + - 1. **Object connections**

ส่วนนี้เป็นส่วนที่บ่งบอกความสัมพันธ์ของข้อมูลแต่ละชนิดใน Object Properties เช่นความสัมพันธ์ระหว่าง Deformer กับ AnimationCurve หรือ ความสัมพันธ์ระหว่าง Node กับ Deformer เป็นต้น

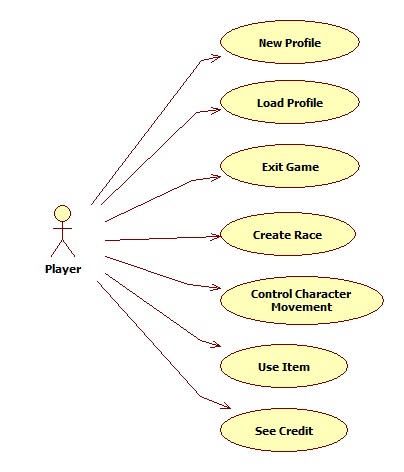
**บทที่ 3**

**การออกแบบระบบ**

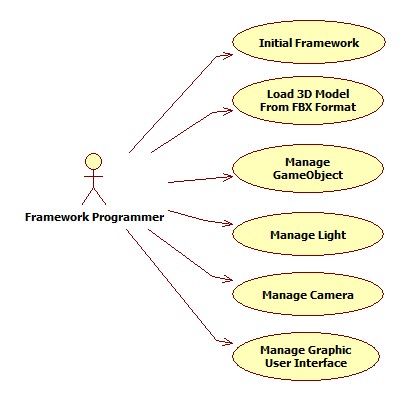
การพัฒนาเฟรมเวิร์คนั้นจะต้องมีการออกแบบระบบของโปรแกรมเสียก่อน ดังนั้นก่อนการเริ่มทำงานจึงต้องทำการวิเคราะห์ระบบของเฟรมเวิร์ค โดยจำแนกความต้องการของระบบและสามารถเขียนให้อยู่ในรูปแบบของไดอะแกรมต่างๆได้ เช่น Use Case Diagram, Activity Diagram, Sequence Diagram และ Class Diagram โดยแต่ละไดอะแกรมมีรูปแบบดังต่อไปนี้

* 1. **การออกแบบและพัฒนา**
     1. **ไดอะแกรมแสดงการใช้งาน (Use Case Diagram)**

สามารถแบ่งไดอะแกรมแสดงฟังก์ชันการทำงานของเกมและเฟรมเวิร์คออกเป็นกลุ่มการทำงานได้ ดังนี้

****

**รูปที่ 3.1** ไดอะแกรมแสดงสิ่งที่ผู้เล่นสามารถทำกับเกมได้

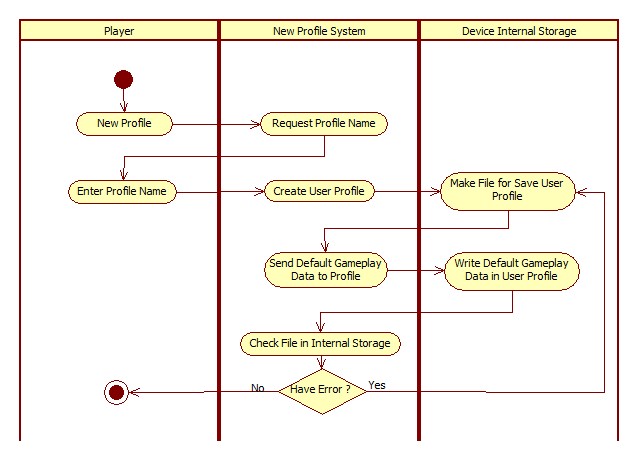


**รูปที่ 3.2** ไดอะแกรมแสดงการใช้งานของเฟรมเวิร์ค

* + 1. **ไดอะแกรมต่างๆ**

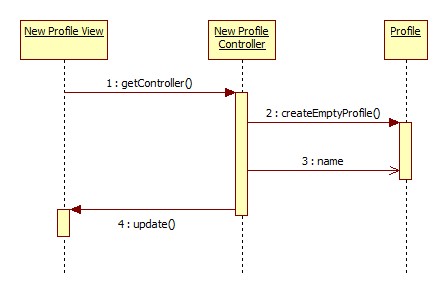
**New Profile**

* 1. Activity Diagram



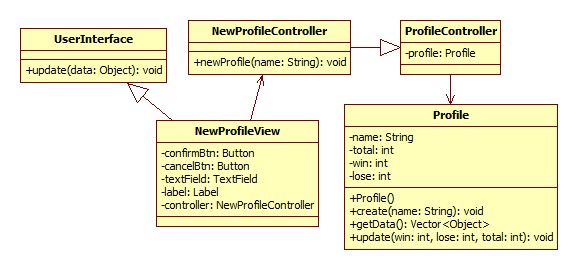
**รูปที่ 3.3** แสดงกิจกรรมการสร้างโปรไฟล์

* 1. Sequence Diagram



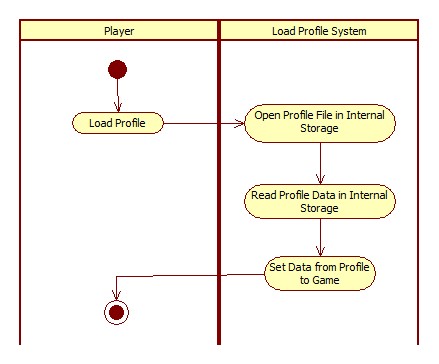
**รูปที่ 3.4** ไดอะแกรมแสดงลำดับการทำงานของการสร้างโปรไฟล์

* 1. Class Diagram



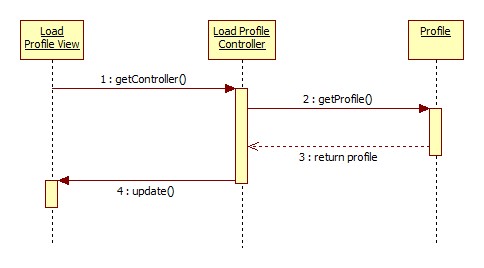
**รูปที่ 3.5** ไดอะแกรมแสดงลำดับคลาสการสร้างโปรไฟล์

* + - 1. **Load Profile**
  1. Activity Diagram



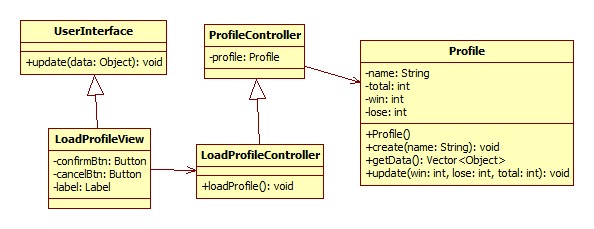
**รูปที่ 3.6** ไดอะแกรมแสดงกิจกรรมของการโหลดโปรไฟล์

* 1. Sequence Diagram



**รูปที่ 3.7** ไดอะแกรมแสดงลำดับการทำงานของการโหลดโปรไฟล์

* 1. Class Diagram



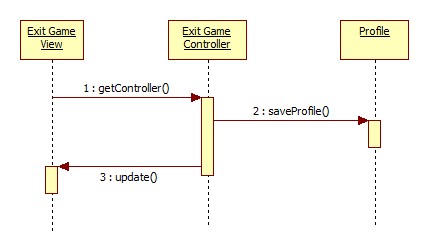
**รูปที่ 3.4** ไดอะแกรมแสดงคลาสของการโหลดโปรไฟล์

* + - 1. **Exit Game**
  1. Activity Diagram



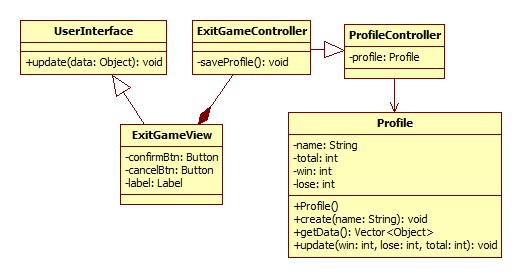
**รูปที่ 3.8** ไดอะแกรมแสดงกิจกรรมของการออกจากเกมและบันทึกโปรไฟล์

* 1. Sequence Diagram



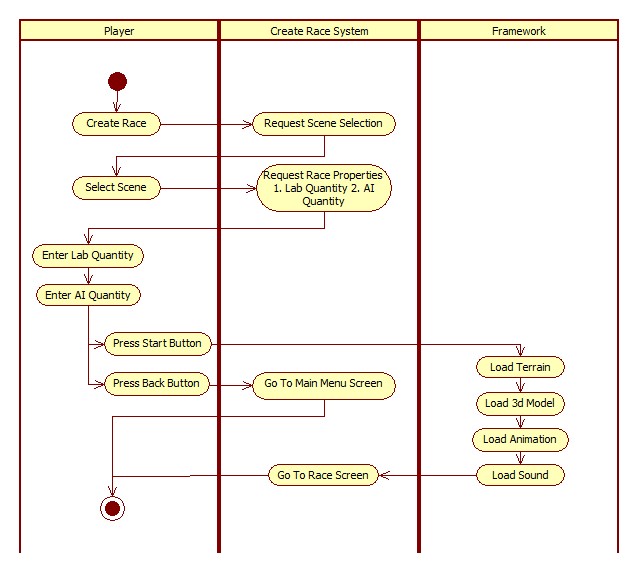
**รูปที่ 3.9** ไดอะแกรมแสดงลำดับการทำงานของการออกจากเกมและบันทึกโปรไฟล์

* 1. Class Diagram

****

**รูปที่ 3.10** ไดอะแกรมแสดงคลาสของการออกจากเกมและบันทึกโปรไฟล์

* + - 1. **Create Race**
  1. Activity Diagram



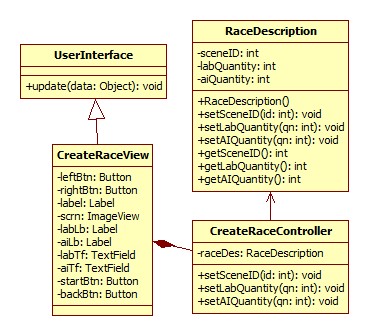
**รูปที่ 3.11** ไดอะแกรมแสดงกิจกรรมของการออกสร้างการแข่ง

* 1. Sequence Diagram



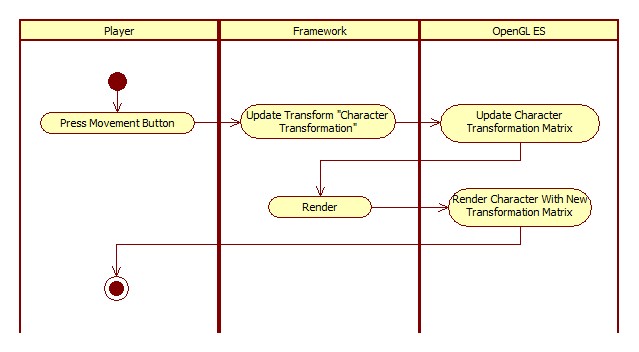
**รูปที่ 3.12** ไดอะแกรมลำดับการทำงานของการสร้างการแข่ง

* 1. Class Diagram



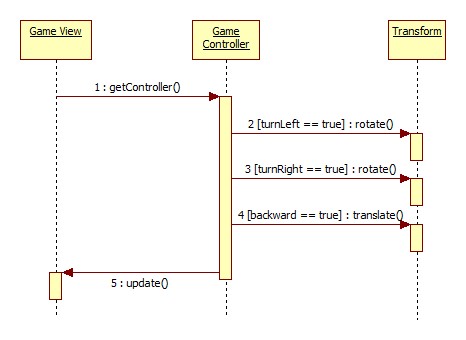
**รูปที่ 3.13** ไดอะแกรมแสดงคลาสของการสร้างการแข่ง

1. **Control Character Movement**
   1. Activity Diagram



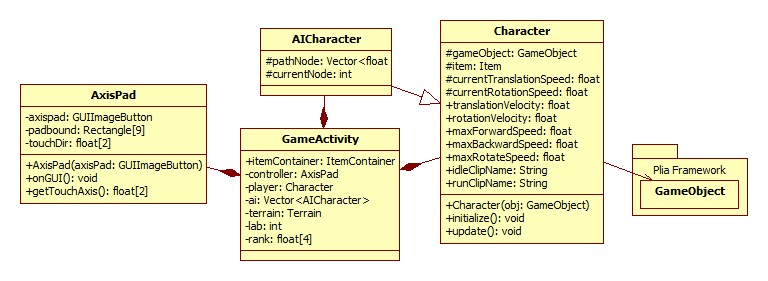
**รูปที่ 3.14** ไดอะแกรมแสดงกิจกรรมของการออกบังคับตัวละคร

* 1. Sequence Diagram



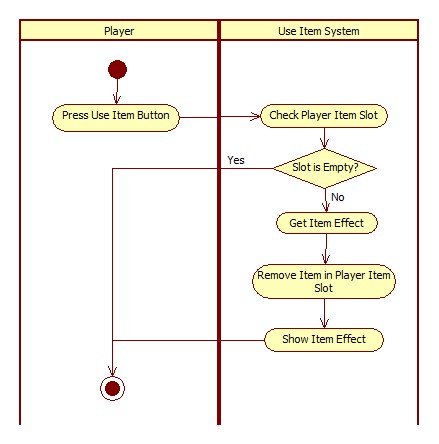
**รูปที่ 3.15** ไดอะแกรมแสดงลำดับการทำงานของการออกบังคับตัวละคร

* 1. Class Diagram



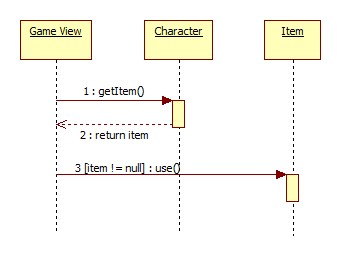
**รูปที่ 3.16** ไดอะแกรมแสดงคลาสของการออกบังคับตัวละคร

1. **Use Item**
   1. Activity Diagram



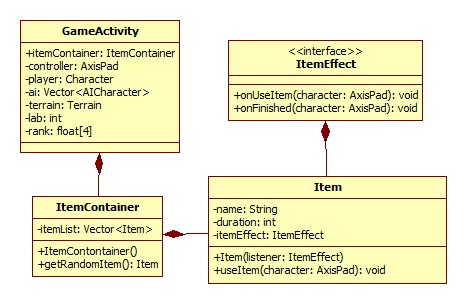
**รูปที่ 3.16** ไดอะแกรมแสดงกิจกรรมของการใช้ไอเทม

* 1. Sequence Diagram



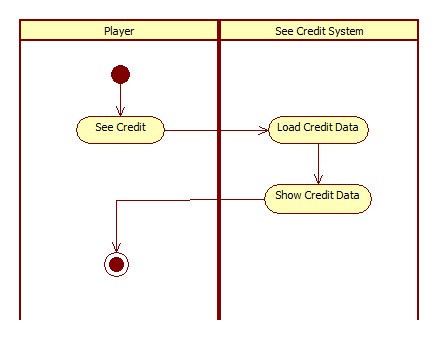
**รูปที่ 3.17** ไดอะแกรมแสดงลำดับการทำงานของการใช้ไอเทม

* 1. Class Diagram



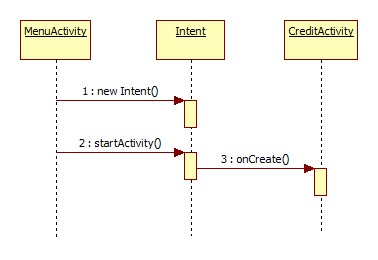
**รูปที่ 3.18** ไดอะแกรมแสดงคลาสของการใช้ไอเทม

1. **See Credit**
   1. Activity Diagram



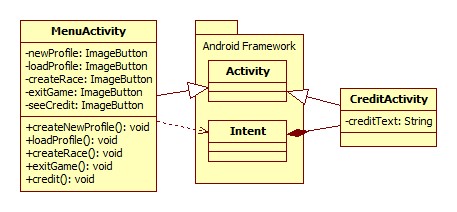
**รูปที่ 3.19** ไดอะแกรมแสดงกิจกรรมของการดูเครดิต

* 1. Sequence Diagram



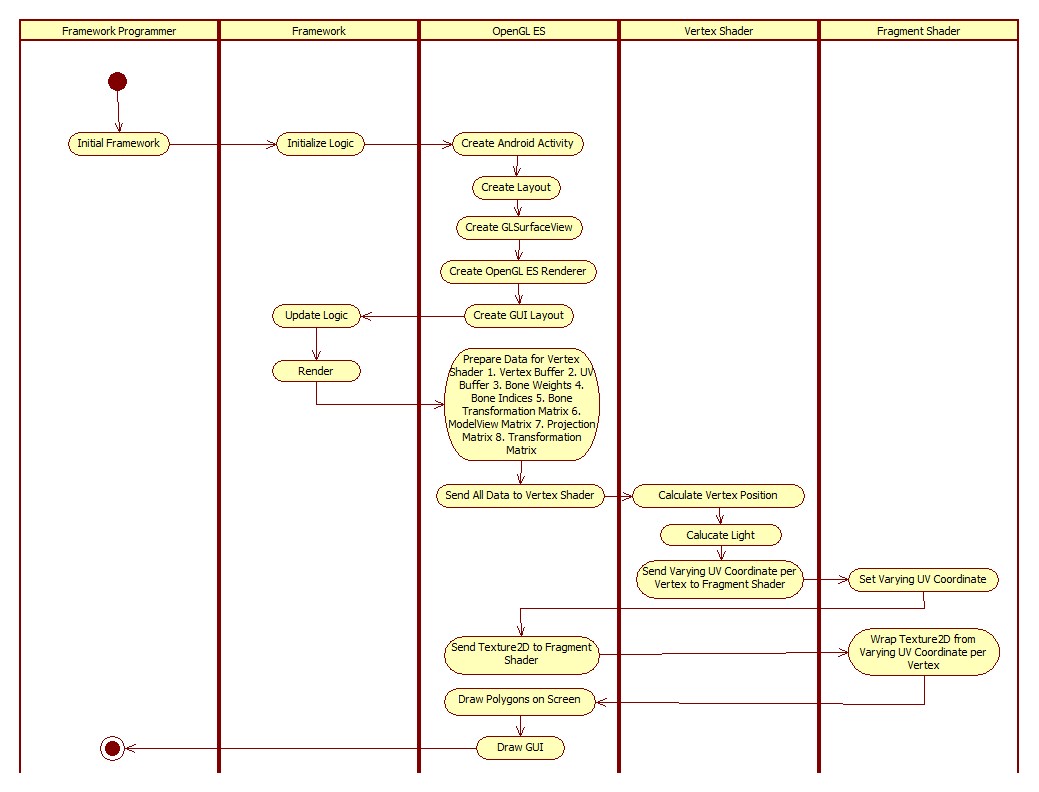
**รูปที่ 3.20** ไดอะแกรมแสดงลำดับการทำงานของการดูเครดิต

* 1. Class Diagram



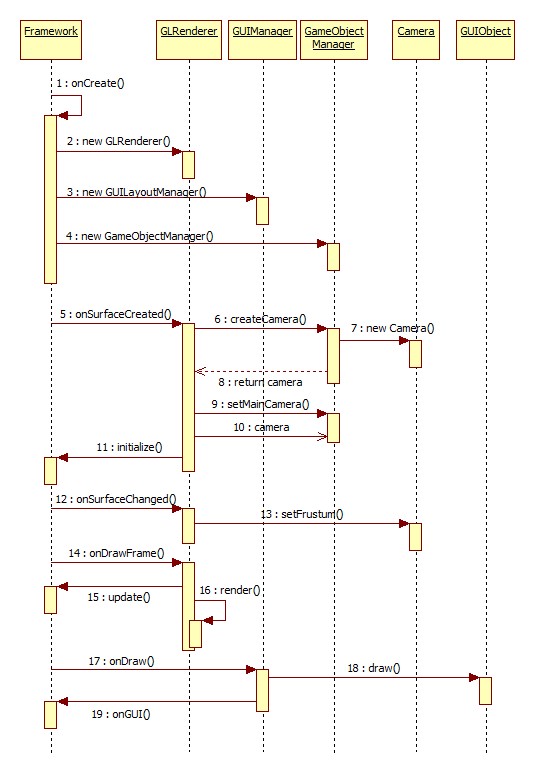
**รูปที่ 3.21** ไดอะแกรมแสดงคลาสของการดูเครดิต

1. **Initial Framework**
   1. Activity Diagram



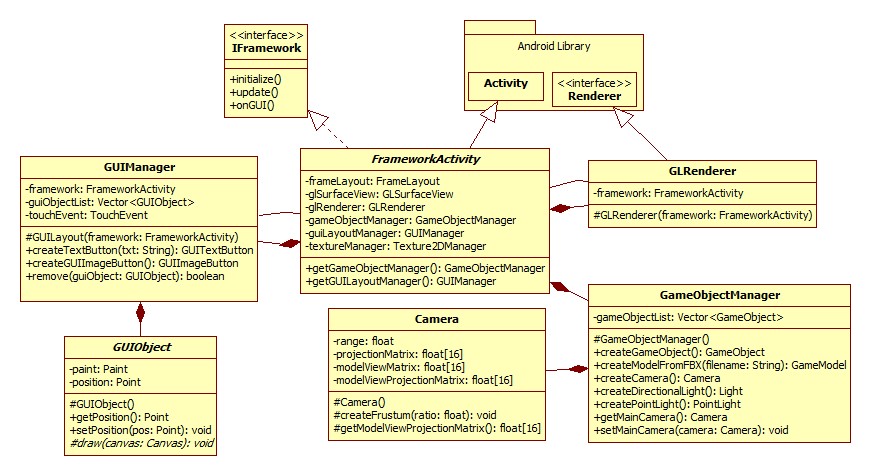
**รูปที่ 3.22** ไดอะแกรมแสดงกิจกรรมของการกำหนดค่าเริ่มต้นของเฟรมเวิร์ค

* 1. Sequence Diagram



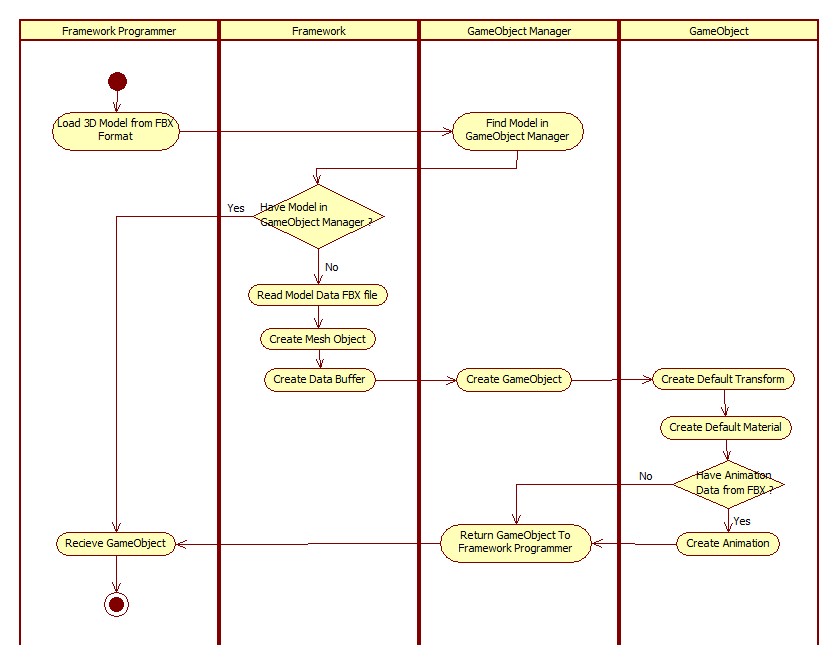
**รูปที่ 3.23** ไดอะแกรมแสดงลำดับการทำงานของการกำหนดค่าเริ่มต้นของเฟรมเวิร์ค

* 1. Class Diagram



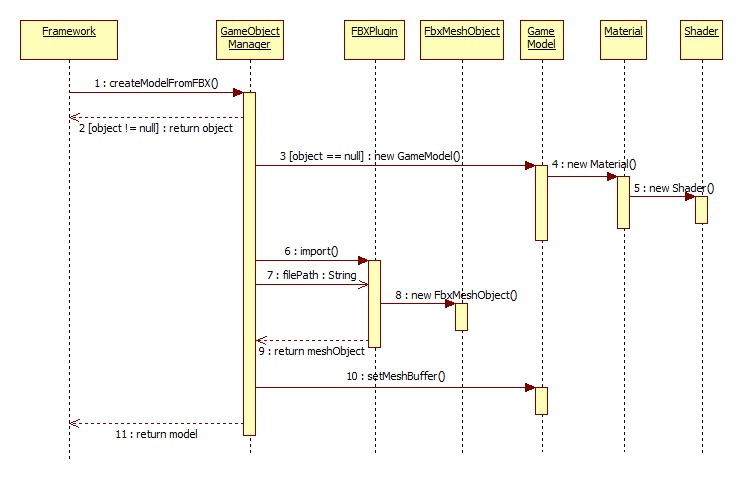
**รูปที่ 3.24** ไดอะแกรมแสดงคลาสของการกำหนดค่าเริ่มต้นของเฟรมเวิร์ค

1. **Load 3D Model From FBX Format**
   1. Activity Diagram



**รูปที่ 3.25** ไดอะแกรมแสดงกิจกรรมของการโหลดโมเดลจากไฟล์ .FBX

* 1. Sequence Diagram



**รูปที่ 3.26** ไดอะแกรมแสดงลำดับการทำงานของการโหลดโมเดลจากไฟล์ .FBX

* 1. Class Diagram



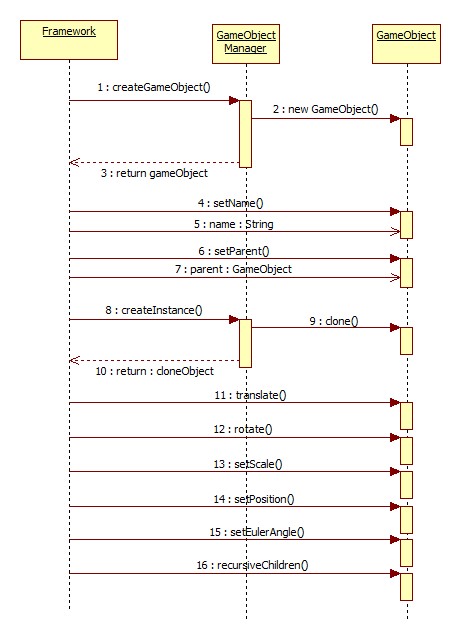
**รูปที่ 3.27** ไดอะแกรมแสดงคลาสของการโหลดโมเดลจากไฟล์ .FBX

1. **Manage Game Object**
   1. Activity Diagram



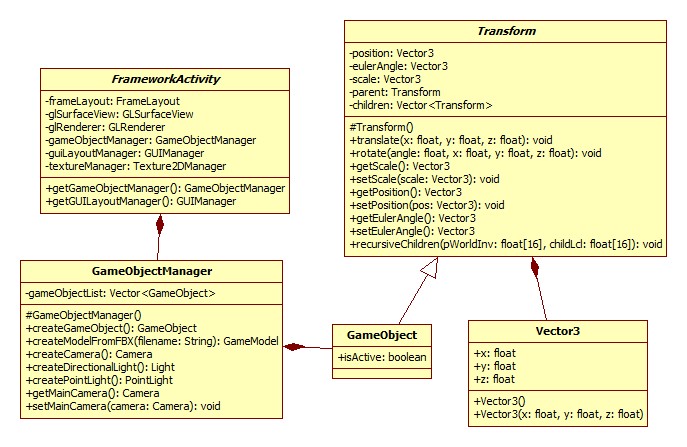
**รูปที่ 3.28** ไดอะแกรมแสดงกิจกรรมของการจัดการ GameObject

* 1. Sequence Diagram



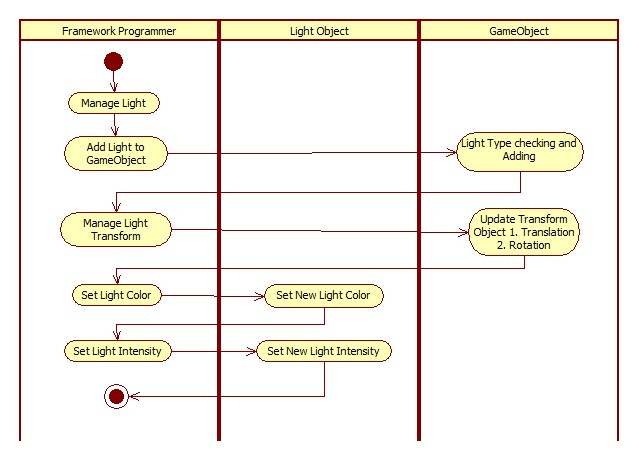
**รูปที่ 3.29** ไดอะแกรมแสดงลำดับการทำงานของการจัดการ GameObject

* 1. Class Diagram



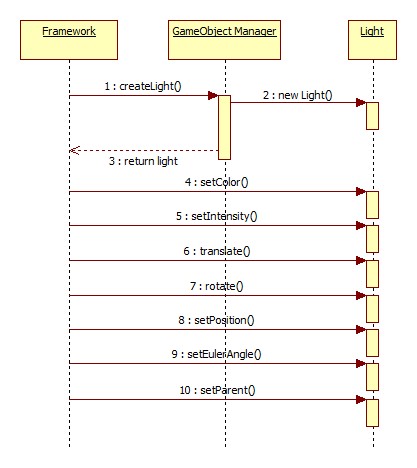
**รูปที่ 3.30** ไดอะแกรมแสดงคลาสของการจัดการ GameObject

1. **Manage Light**
   1. Activity Diagram



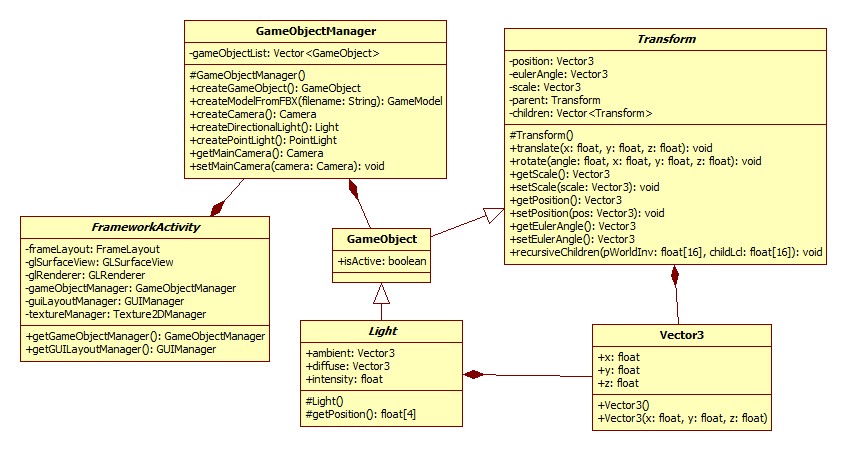
**รูปที่ 3.31** ไดอะแกรมแสดงกิจกรรมของการจัดการแสง

* 1. Sequence Diagram



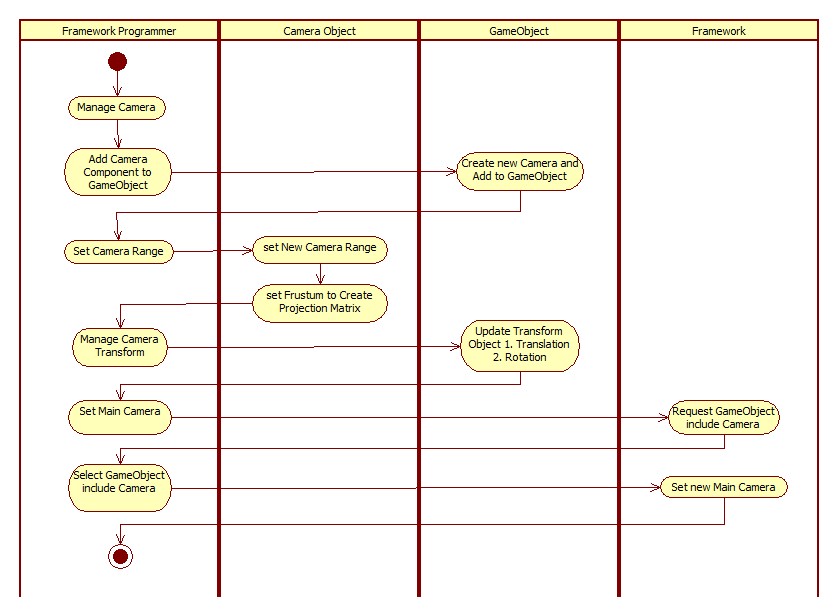
**รูปที่ 3.32** ไดอะแกรมแสดงลำดับการทำงานของการจัดการแสง

* 1. Class Diagram



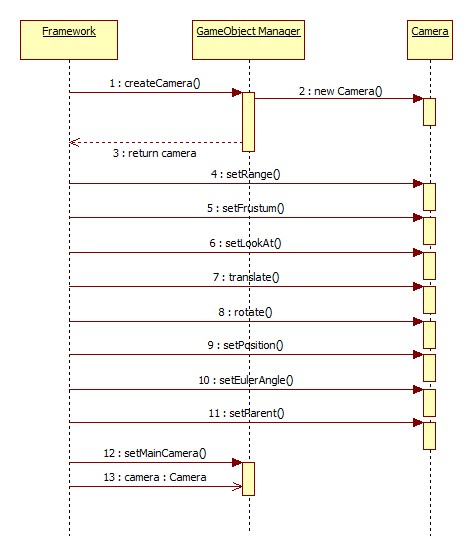
**รูปที่ 3.32** ไดอะแกรมแสดงคลาสของการจัดการแสง

1. **Manage Camera**
   1. Activity Diagram



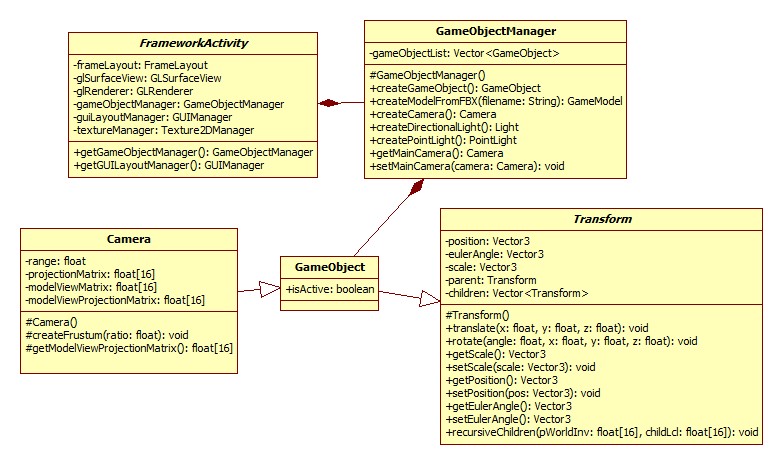
**รูปที่ 3.33** ไดอะแกรมแสดงกิจกรรมของการจัดการกล้อง

* 1. Sequence Diagram



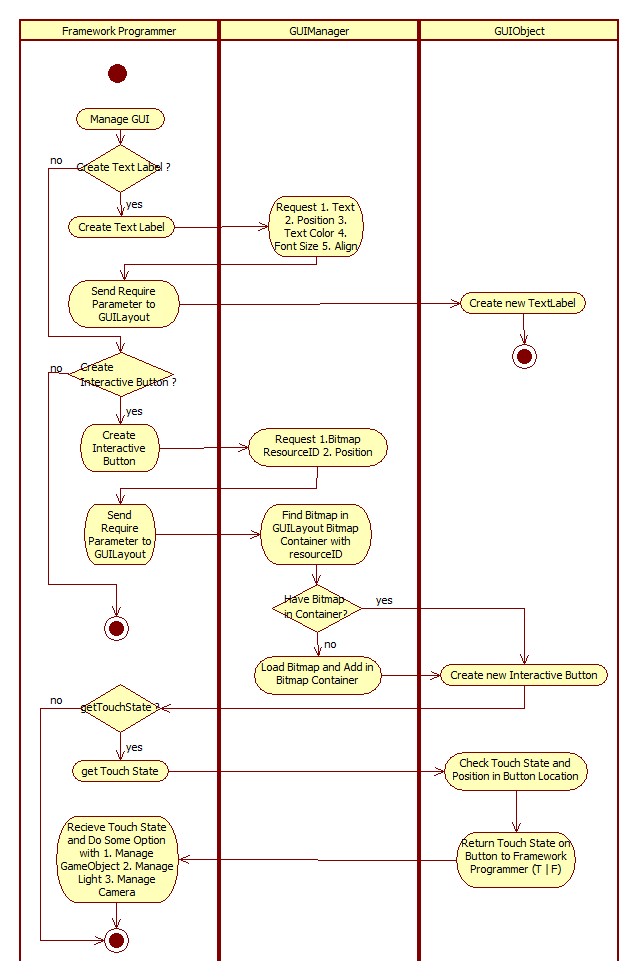
**รูปที่ 3.34** ไดอะแกรมแสดงลำดับการทำงานของการจัดการกล้อง

* 1. Class Diagram



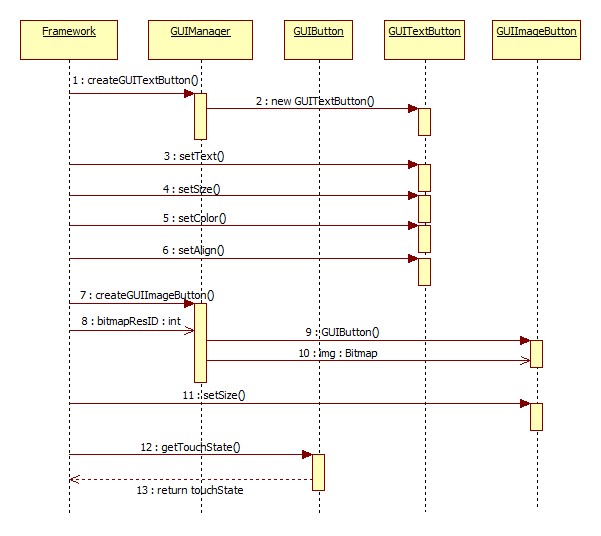
**รูปที่ 3.35** ไดอะแกรมแสดงคลาสของการจัดการกล้อง

1. **Manage Graphic User Interface**
   1. Activity Diagram



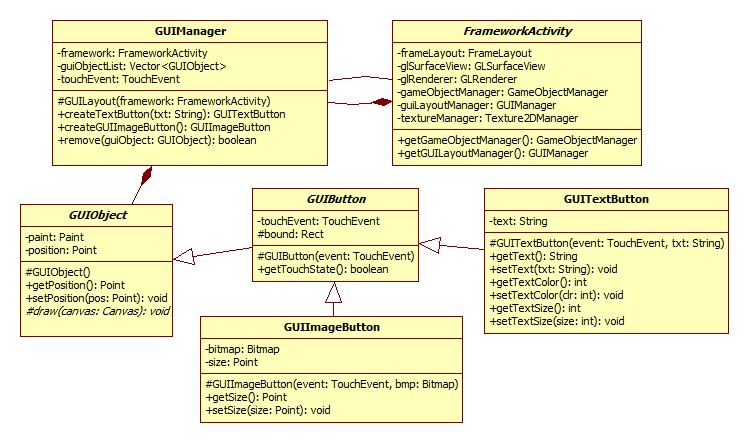
**รูปที่ 3.36** ไดอะแกรมแสดงกิจกรรมของการจัดการ Graphic User Interface

* 1. Sequence Diagram



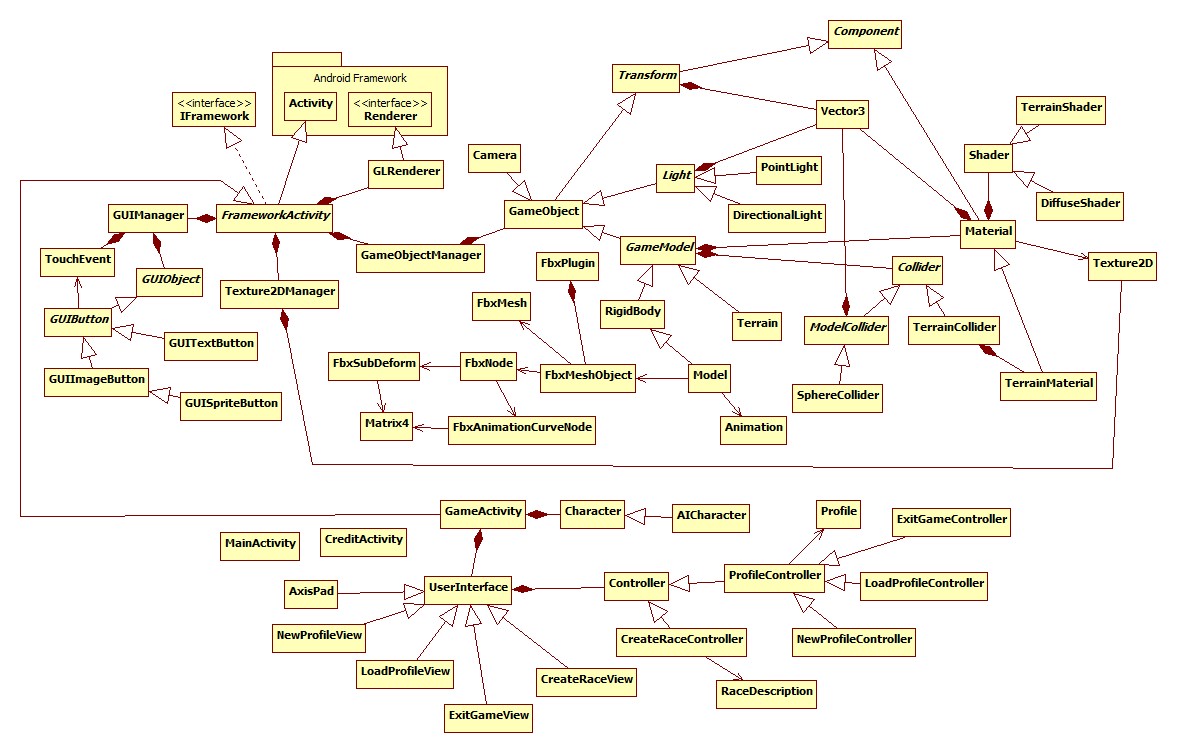
**รูปที่ 3.38** ไดอะแกรมแสดงลำดับการทำงานของการจัดการ Graphic User Interface

* 1. Class Diagram



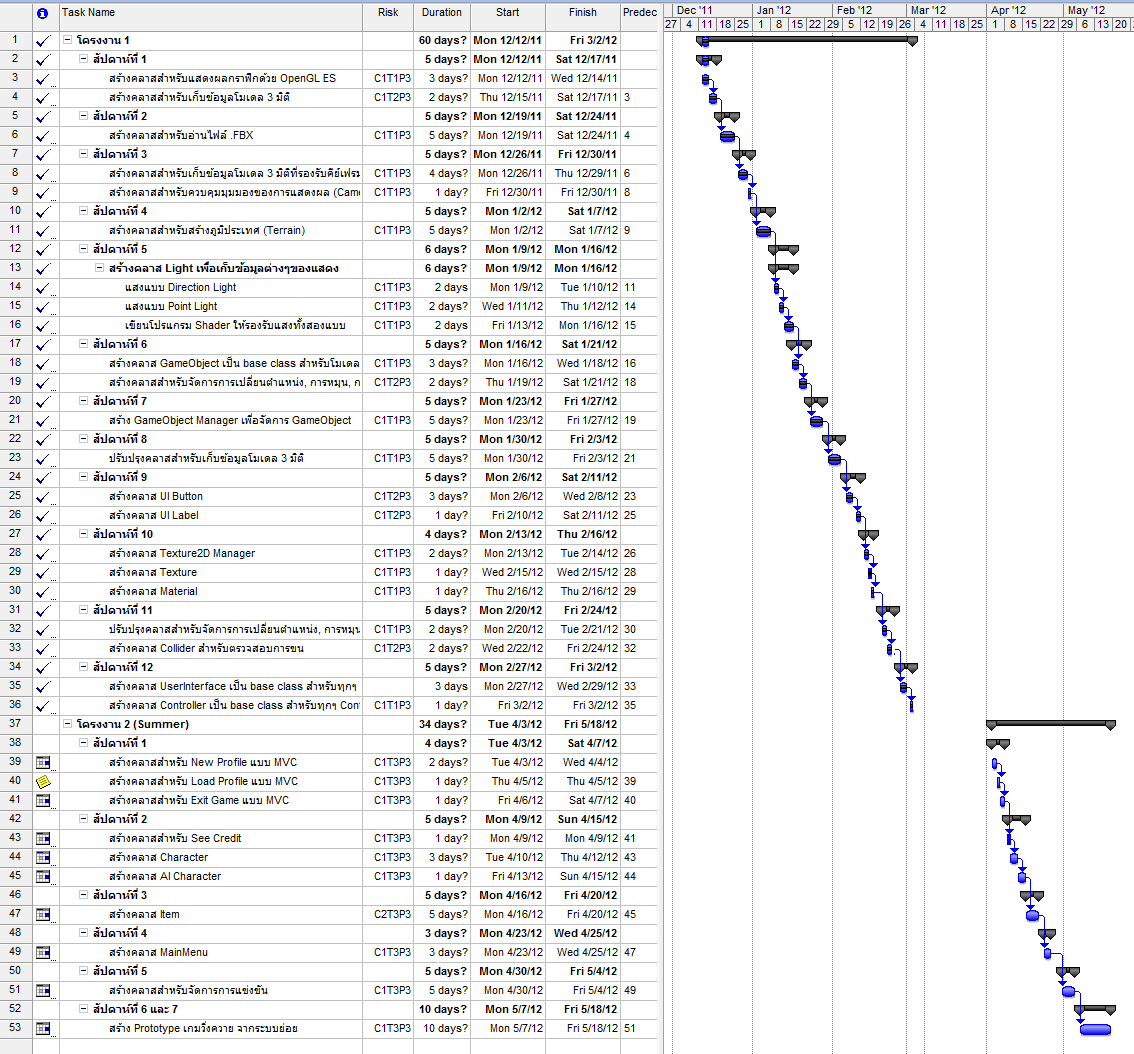
**รูปที่ 3.39** ไดอะแกรมแสดงคลาสของการจัดการ Graphic User Interface

จากคลาสไดอะแกรมที่กล่าวมาข้างต้น สามารถนำมาแสดงอยู่ในรูปแบบของระบบโดยรวมได้ดังรูปที่ 3.40

****

**รูปที่ 3.40** ภาพรวมของเฟรมเวิร์คและเกมวิ่งควาย

โดยวางแผนระยะเวลาในการดำเนินงานตามขั้นตอนต่างๆ ดังตารางที่ 3.1**ตารางที่ 3.1** ตารางดำเนินงานในการสร้างเฟรมเวิร์คและเกมวิ่งควาย

****

**บทที่ 4**

**แนวทางการพัฒนา**

* 1. **สถาปัตยกรรมของระบบ**

ในส่วนของสถาปัตยกรรมของระบบของเฟรมเวิร์ค สามารถแบ่งออกได้เป็น 3 ส่วนดังนี้

* + 1. **ลักษณะของเฟรมเวิร์ค**

เฟรมเวิร์คนี้ถูกพัฒนาขึ้นด้วยภาษา OpenGL ES 2.0 และ Android Framework ซึ่งจะเน้นไปที่ส่วนของกราฟิก 3 มิติเป็นหลัก โดยภายในการทำงานของเฟรมเวิร์คจะประกอบไปด้วย GLSurfaceView (เป็นคลาสสำหรับสร้างการทำงานกับ OpenGL ES) และ Activity (เป็นคลาสที่เรียกใช้งาน GLSurfaceView)

* + 1. **การทำงานของเฟรมเวิร์ค**

การทำงานของเฟรมเวิร์คแบ่งได้เป็น 4 ฟังก์ชันหลักๆ ที่เป็นส่วนติดต่อหลักกับผู้ใช้ในการพัฒนาเกม ดังรูปที่ 4.1

Initialize

Update

Render

Destroy

**รูปที่ 4.1** การทำงานของเฟรมเวิร์ค

จากรูปที่ 4.1 สามารถอธิบายได้ดังนี้

1. **ฟังก์ชัน Initialize**

เป็นส่วนของการกำหนดค่าเริ่มต้นของเฟรมเวิร์ค เช่น การโหลดโมเดล 3 มิติ การโหลดเสียง การสร้างแหล่งกำเนิดแสง หรือการสร้างกล้อง เป็นต้น

1. **ฟังก์ชัน Update**

เป็นส่วนการอัพเดทข้อมูลการทำงานของเฟรมเวิร์ค ซึ่ง Update จะเป็นเธรด 1 เธรดที่เป็นการแยกการทำงานออกมา เนื่องจาก OpenGL ES จะมีเพียงเธรดของการเรนเดอร์เพียงอย่างเดียว จึงต้องสร้างเธรดของการอัพเดทขึ้นมา เพื่อลดภาระของเธรดการเรนเดอร์ และการอัพเดทประกอบไปด้วย

* 1. ตำแหน่งของ GameObject (GameObject จะกล่าวถึงในหัวข้อที่ 4.1.3)
  2. การเคลื่อนไหวของกระดูกในแต่ละ GameObject
  3. การตรวจสอบการชนกันของแต่ละ GameObject

1. **ฟังก์ชัน Render**

ในส่วนนี้จะถูกควบคุมโดยเฟรมเวิร์ค ผู้ใช้เฟรมเวิร์คจะไม่สามารถเข้ามาแก้ไขการทำงานได้

เป็นส่วนของการแสดงผล GameObject ที่ถูกการสร้างขึ้นมา โดยการแสดงผลนั้นจำเป็นจะต้องมีกล้องเป็นส่วนประกอบ เพื่อกำหนดมุมมองในการแสดงผล

1. **ฟังก์ชัน Destroy**

ในส่วนนี้จะถูกควบคุมโดยเฟรมเวิร์ค ผู้ใช้เฟรมเวิร์คจะไม่สามารถเข้ามาแก้ไขการทำงานได้ แต่สามารถเรียกใช้ได้

เป็นส่วนของการคืนหน่วยความจำ เช่นการลบ GameObject ที่ถูกสร้างขึ้นมา เป็นต้น

* + 1. **ส่วนประกอบของเฟรมเวิร์ค**

**GameObject Manager**

ภายในของ GameObject Manager จะประกอบไปด้วย GameObject ที่มีจำนวนตั้งแต่ศูนย์ขึ้นไป โดย GameObject Manager จะทำการตรวจสอบการโหลดโมเดลจากผู้ใช้เฟรมเวิร์คว่า เคยโหลดโมเดลที่ผู้ใช้เฟรมเวิร์คสั่งให้โหลดหรือเปล่า ถ้าเคยโหลดแล้วจะไม่ทำการโหลดซ้ำอีก เพื่อเป็นการลดความซ้ำซ้อนของข้อมูล

**GameObject**

GameObject เป็นคลาสที่มีความสำคัญของเฟรมเวิร์คนี้ เนื่องจาก GameObject เป็นคลาสที่ประกอบไปด้วย โมเดลสามมิติ ภูมิประเทศ แสง กล้อง และ คลาสย่อยๆ หลายคลาสดังนี้

**Mesh Object**

คลาสนี้เป็นคลาสที่มีข้อมูลโพลีกอนของโมเดล 3 มิติ โดยจะถูกควบคุมโดยเฟรมเวิร์ค

**Transform**

คลาสนี้เป็นคลาสสำหรับเก็บตำแหน่ง องศาการหมุน และอัตราส่วนของโมเดล 3 มิติ ซึ่งผู้ใช้สามารถเรียกใช้งานด้วยคำสั่งต่างๆเพื่อกำหนดค่าของตำแหน่ง องศาการหมุน และอัตราส่วน ดังนี้

* + **Translate(float x, float y, float z);**

เป็นการเคลื่อนที่ GameObject ด้วยค่า x,y,z (Cartesian Coordinate)

* + **Rotate(float angle, float x, float y, float z);**

เป็นการสั่งให้ GameObject หมุนด้วยทิศทาง angle และกำหนดแกนการหมุนด้วย x,y,z

* + **Scale(float x, float y, float z);**

เป็นการปรับอัตราส่วนของ GameObject ด้วย x,y,z

**Animation**

คลาสนี้เป็นคลาสสำหรับการเก็บคีย์เฟรมแอนิเมชันของโมเดลสามมิติ และสามารถแบ่งช่วงของคีย์เฟรมเพื่อให้คีย์เฟรมแอนิเมชันเล่นเฉพาะช่วงที่ต้องการได้ ซึ่งผู้ใช้สามารถเรียกใช้งานด้วยคำสั่งต่างๆเพื่อกำหนดค่าคีย์เฟรมแอนิเมชัน ดังนี้

* + **AddAnimationClip (String name, int startf, int endf);**

เป็นการกำหนดช่วงของอนิเมชัน โดยพารามิเตอร์คือ ชื่อของช่วง และคีย์เฟรม เริ่มต้น-สิ้นสุด

* + **RemoveAnimationClip(String clipName);**

ทำการลบข้อมูลช่วงของอนิเมชันออกโดย มีพารามิเตอร์คือ ชื่อของช่วงที่ต้องการลบ

* + **Play(String clipName, boolean loop);**

เป็นการสั่งให้เล่นคีย์เฟรมอนิเมชันจากชื่อของช่วง และกำหนดว่าต้องการให้เล่นแบบวนหรือเปล่า

* + **Stop(String clipName);**

เป็นการสั่งให้หยุดเล่นแอนิเมชันจากชื่อของช่วง

**Collider**

คลาสนี้จะเป็นคลาสสำหรับเก็บ Bounding Box ของ GameObject ที่ใช้ในการตรวจสอบการชนกับวัตถุอื่นๆภายในเกม ซึ่งจะแบ่งเป็น 2 รูปแบบ คือ แบบทรงกลม (Sphere Collider) และ Heightmap (Terrain Collider) โดยมีคำสั่งให้เรียกใช้ดังนี้

* + **Intersect(Collider a, Collider b);**

เป็นการตรวจสอบการชนด้วย Collider ซึ่งจะทำการคืนค่าเป็น true false

**Camera**

คลาสนี้จะเป็นคลาสที่เก็บ ระยะการแสดงผลของกล้อง ทั้งแนวกว้างและแนวลึก โดยมีคำสั่งต่างๆให้เรียกใช้ดังนี้

* + **setRange(float range);**

เป็นคำสั่งสำหรับตั้งระยะการแสดงผลในแนวลึก โดยมีพารามิเตอร์เป็นระยะการแสดงผล

**Light**

* + **setLightDensity(float density);**

เป็นคำสั่งกำหนดความเข้มของแสง โดยมีพารามิเตอร์เป็นค่าความเข้มแสงที่ต้องการ

* + **setLightColor(float r, float g, float b);**

เป็นคำสั่งกำหนดค่าสีของแสง โดยมีพารามิเตอร์เป็นสี r,g,b มีค่าตั้งแต่ 0 ถึง 1 สามารถเป็นเลขทศนิยมได้

* + **setLightType (String type);**

เป็นคำสั่งสำหรับกำหนดรูปแบบของแสง โดยพารามิเตอร์จะเป็นรูปแบบของแสงที่ต้องการ ซึ่งมีคีย์เวิร์ดดังนี้ “Directional”, “Point”

* + - **Directional** เป็นแสงแบบพระอาทิตย์
    - **Point** เป็นแสงแบบหลอดไฟหรือแสงเทียน

**Material**

เป็นคลาสสำหรับเก็บค่าสีของโมเดล และเก็บเทกเจอร์ โดยมีคำสั่งให้เรียกใช้ดังนี้

* + **setColor(float r, float g, float b, float a);**

เป็นคำสั่งกำหนดค่าสีของโพลีกอน โดยมีพารามิเตอร์คือ สีแดง สีเขียว สีน้ำเงิน และค่าความโปร่งแสง

* + **setTexture2D(Bitmap bitmap);**

เป็นคำสั่งกำหนดเทกเจอร์ของโพลีกอน โดยมีพารามิเตอร์คือ ภาพบิตแมป

**Parent GameObject**

เก็บ GameObject ที่เป็น Parent เพื่อให้เกิด Scene Graph

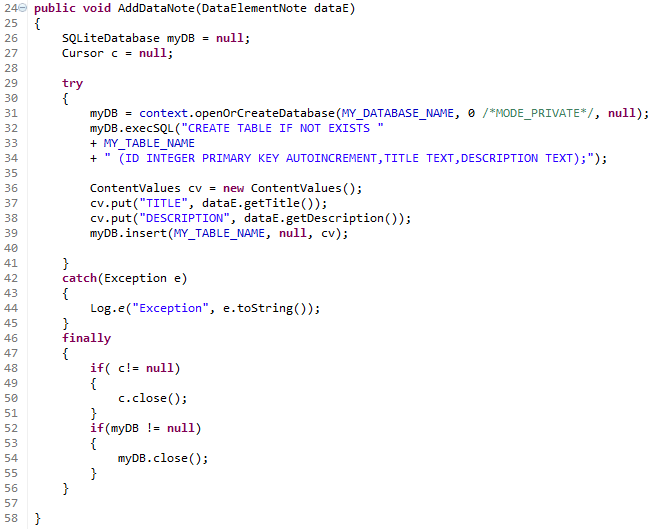
**ChildList GameObject**

เก็บ GameObject ที่เป็น Child ในรูปแบบ List เพื่อให้เกิด Scene Graph

**Scene Graph**

ซีนกราฟในเฟรมเวิร์คจะนำมาใช้กับ GameObject เพราะว่า GameObject นั้นเป็นคลาสหลักของเฟรมเวิร์คที่มีทั้งคลาสโมเดล กล้อง และแสง ซึ่งการใช้งานมีดังตัวอย่าง เช่น กำหนดกล้องเป็นโหนดลูกของโมเดล เมื่อมีการเปลี่ยนตำแหน่งจะทำให้กล้องเปลี่ยนตำแหน่งด้วย หรือการทำแอนิเมชันที่มีการขยับกระดูกที่เป็นโหนดแม่ จะทำให้โหนดลูกขยับตาม

* 1. **แนวทางการพัฒนา**
     1. **New Profile**
        + **การเพิ่มข้อมูลลงใน SQLite**

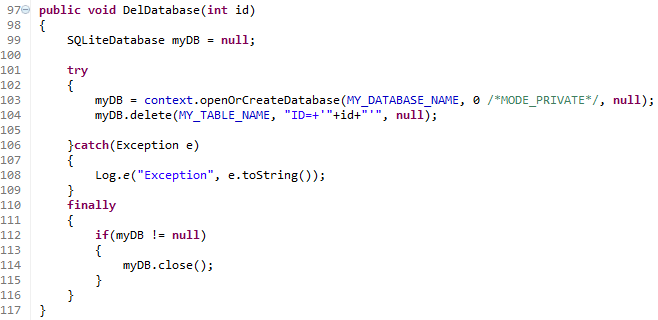
****

**รูปที่ 4.2** การเพิ่มข้อมูลลงใน SQLite

จากโค๊ดตัวอย่างรูปที่ 4.2 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 26 | ประกาศออบเจกต์ของฐานข้อมูล SQLite โดยให้ค่าเป็น null |
| 31-34 | เรียกฐานข้อมูล SQLite จากหน่วยความจำของเครื่องไว้ที่ตัวแปร myDB  ถ้าไม่มีจะสร้างฐานข้อมูลขึ้นมาใหม่ |
| 36-39 | ใส่ข้อมูลลงในฐานข้อมูล |
| 46-56 | ทำการปิดการเชื่อมต่อฐานข้อมูล |

* + - * **การลบข้อมูลใน SQLite**

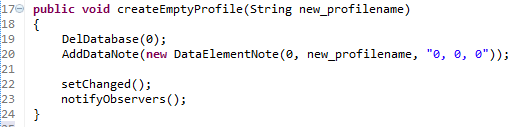
****

**รูปที่ 4.3** การลบข้อมูลใน SQLite

จากโค๊ดตัวอย่างรูปที่ 4.3 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 99 | ประกาศออบเจกต์ของฐานข้อมูล SQLite โดยให้ค่าเป็น null |
| 103 | เรียกฐานข้อมูล SQLite จากหน่วยความจำของเครื่องไว้ที่ตัวแปร myDB |
| 104 | ลบข้อมูลในฐานข้อมูล |
| 110-116 | ทำการปิดการเชื่อมต่อฐานข้อมูล |

* + - * **ฟังก์ชันสร้างโปรไฟล์ ในส่วนของคอนโทรลเลอร์**

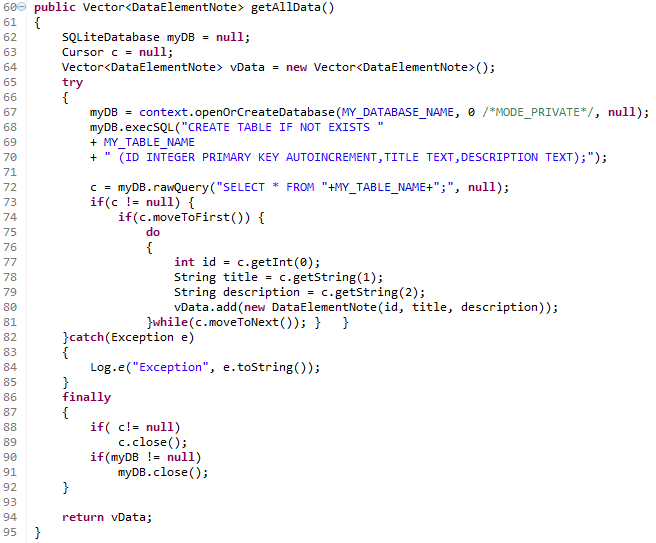
****

**รูปที่ 4.4** การสร้างโปรไฟล์ในส่วนของคอนโทรลเลอร์

จากโค๊ดตัวอย่างรูปที่ 4.4 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 19 | ลบโปรไฟล์เก่า |
| 20 | สร้างโปรไฟล์ใหม่ |
| 22-23 | อัพเดทวิว |

* + 1. **Load Profile**
       - **การเรียกข้อมูลในฐานข้อมูล SQLite**

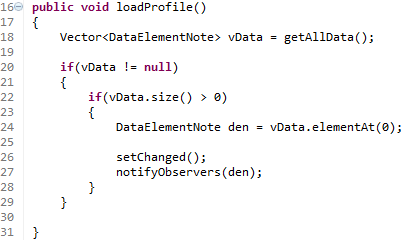
****

**รูปที่ 4.5** การเรียกข้อมูลในฐานข้อมูล SQLite

จากโค๊ดตัวอย่างรูปที่ 4.5 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 62-64 | ประกาศออบเจกต์ของฐานข้อมูล SQLite โดยให้ค่าเป็น null, และตัวแปรสำหรับรับข้อมูลจากฐานข้อมูล |
| 67-70 | เรียกฐานข้อมูล SQLite จากหน่วยความจำของเครื่องไว้ที่ตัวแปร myDB |
| 72-81 | ขอข้อมูลจากฐานข้อมูล และเก็บไว้ที่ตัวแปร vData |
| 86-92 | ทำการปิดการเชื่อมต่อฐานข้อมูล |

* + - * **ฟังก์ชันโหลดโปรไฟล์**

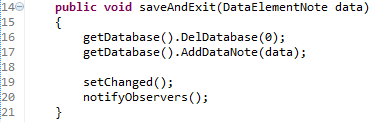
****

**รูปที่ 4.6** การโหลดโปรไฟล์

จากโค๊ดตัวอย่างรูปที่ 4.6 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 18 | ขอข้อมูลจากฐานข้อมูล |
| 20-29 | อัพเดทวิว โดยส่งข้อมูลโปรไฟล์ไปด้วย |

* + 1. **Exit Game**

****

**รูปที่ 4.7** การเซฟเกมก่อนการออกจากเกม

จากโค๊ดตัวอย่างรูปที่ 4.7 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 16 | ลบโปรไฟล์เก่า |
| 17 | สร้างโปรไฟล์ใหม่ โดยใช้ข้อมูลการเล่นล่าสุด |
| 19-20 | อัพเดทวิว |

* + 1. **Create Race**

**การเขียนโปรแกรมการกดปุ่มเพื่อการเริ่มการแข่ง**

****

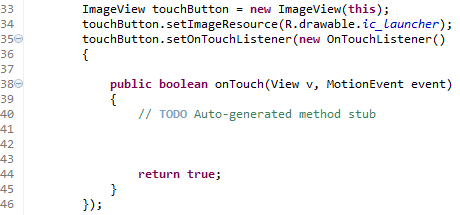
**รูปที่ 4.8** ตัวอย่างการเขียนโปรแกรมการกดปุ่มเพื่อเริ่มการแข่ง

จากโค๊ดตัวอย่างรูปที่ 4.8 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 18-30 | สร้างวิวจากคลาส ImageView (วิวในแอนดรอยด์มีทั้งการแสดงข้อความและรูปภาพ) |
| 24-29 | ทำการตรวจสอบการกดปุ่ม |
| 27-28 | ทำการเรียก Activity TestProjectActivity (Game Activity) |

* + 1. **Control Character Movement**

การบังคับตัวละครในเกมนั้น ผู้เล่นจะต้องกดปุ่มบังคับทิศทางบนหน้าจอแสดงผล ซึ่งการสร้างปุ่มมีแนวทางการสร้างดังรูปที่ 5.5

****

**รูปที่ 4.9** ตัวอย่างการสร้างปุ่มจากรูป และตรวจจับการกดปุ่ม **(**แบบ Touch**)**

จากโค๊ดตัวอย่างรูปที่ 4.9 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 77 | สร้างวิวจากคลาส ImageView (วิวในแอนดรอยด์มีทั้งการแสดงข้อความและรูปภาพ) |
| 78 | ทำการใส่ภาพที่จะใช้เป็นปุ่มให้กับตัว ImageView ที่สร้างขึ้นมา |
| 80 | ติดตั้งการตรวจจับการกดปุ่มให้กับ ImageView |
| 82 | เป็นชื่อฟังก์ชั่นสำหรับตรวจสอบการกด โดยแอนดรอยด์จะเรียกอัตโนมัติ |
| 84 | ใส่การทำงานให้กับปุ่ม เมื่อมีการกดปุ่ม โดยสามารถนำ MotionEvent มาเพิ่มการกฎการตรวจสอบได้ เช่น การกดค้าง การกดแล้วสไลด์ |

* + 1. **Use Item**

แนวทางการทำปุ่มสำหรับไอเทมนั้นจะมีความคล้ายกับเรื่องการบังคับตัวละครในข้อ 5.5 แต่สิ่งที่แตกต่างคือหลังจากที่มีการกดปุ่มไปแล้วนั่นเอง ซึ่งสามารถลำดับขั้นตอนการทำงานคร่าวๆได้ดังนี้

1. ตรวจสอบไอเทมของผู้เล่น
2. ถ้ามีไอเทม ดูข้อ 3 แต่ถ้าไม่มีไอเทมการทำงานจะจบลงเท่านี้
3. ใช้ไอเทม

สามารถเขียนเป็น Flow Chart ได้ดังรูปที่ 5.4

ใช้ไอเทม

ไม่พบไอเทม

พบไอเทม

ตรวจสอบ  
ไอเทม

**รูปที่ 4.10** Flowchart การใช้ไอเทม

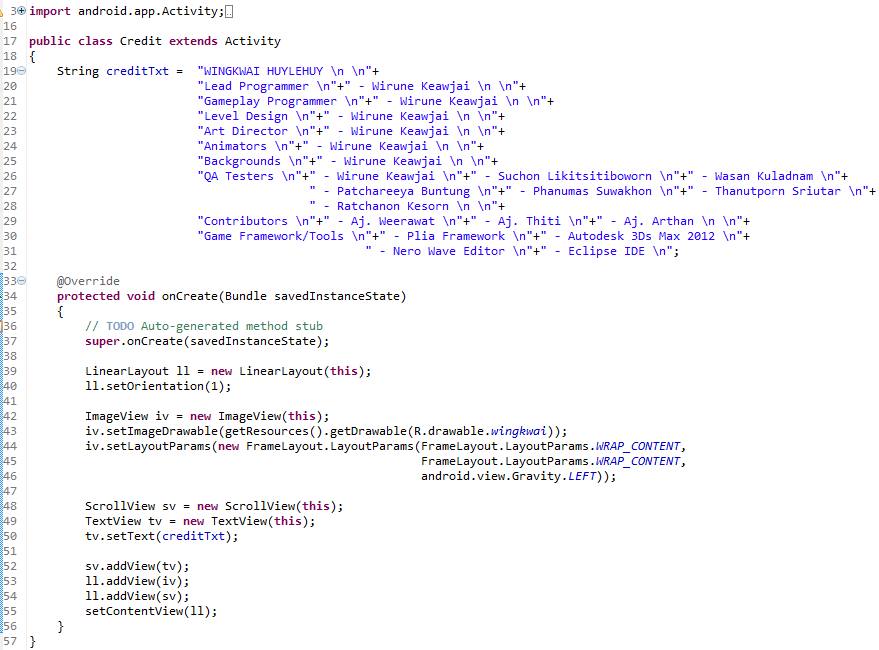


**รูปที่ 4.10** ตัวอย่างการสร้างปุ่มจากรูป และตรวจจับการกดปุ่ม

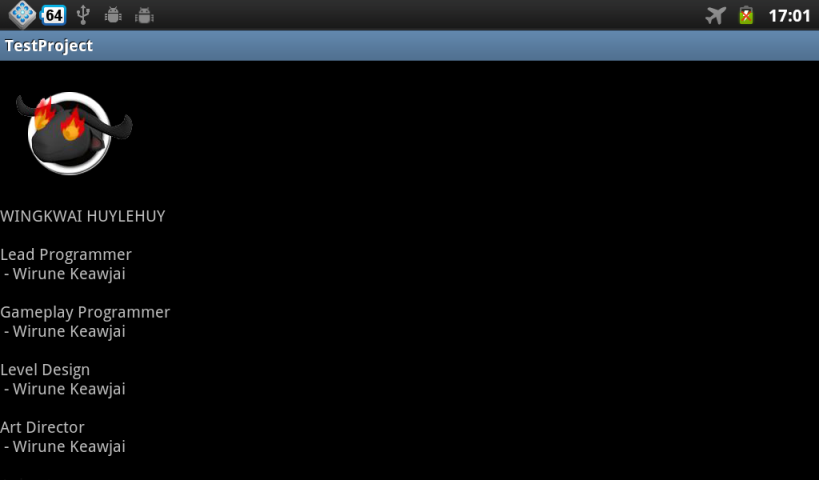
จากโค๊ดตัวอย่างรูปที่ 4.10 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 77 | สร้างวิวจากคลาส ImageView (วิวในแอนดรอยด์มีทั้งการแสดงข้อความและรูปภาพ) |
| 78 | ทำการใส่ภาพที่จะใช้เป็นปุ่มให้กับตัว ImageView ที่สร้างขึ้นมา |
| 80 | ติดตั้งการตรวจจับการกดปุ่มให้กับ ImageView |
| 82 | เป็นชื่อฟังก์ชั่นสำหรับตรวจสอบการกด โดยแอนดรอยด์จะเรียกอัตโนมัติ |
| 84 | ใส่การทำงานให้กับปุ่ม เมื่อมีการกดปุ่ม |

* + 1. **See Credit**

****

**รูปที่ 4.11** ตัวอย่างการสร้างหน้าต่างแสดง Credit

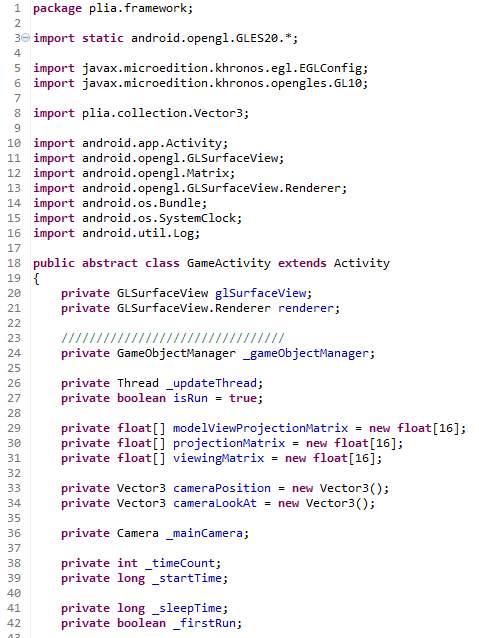


**รูปที่ 4.11** ผลลัพธ์การสร้างหน้าต่างแสดง Credit

จากโค๊ดตัวอย่างรูปที่ 4.11 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 19-32 | ประกาศตัวแปรสตริง เพื่อเก็บรายละเอียดของนักพัฒนา |
| 39 | สร้าง layout หลักของการแสดงผล |
| 42-47 | ใส่ข้อมูลภาพโลโก้เกม |
| 48-50 | ใส่รายละเอียดทีมพัฒนาลงไปใน layout โดยใช้ข้อมูลจากตัวแปรสตริงที่ประกาศไว้ |

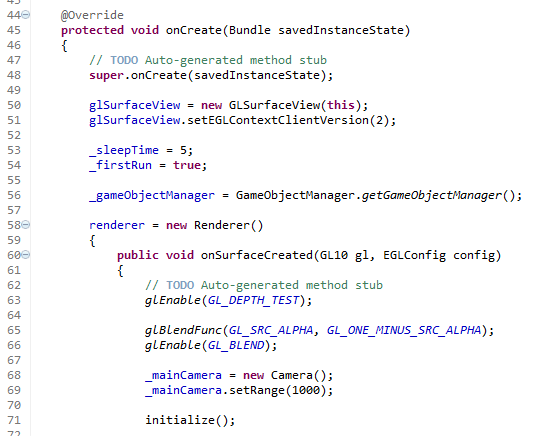
* + 1. **Initial Framework**
* **การเขียนโปรแกรมทางฝั่งผู้พัฒนาเฟรมเวิร์ค**

****

**รูปที่ 4.12** ประกาศตัวแปรคลาส GameActivity

จากโค๊ดตัวอย่างรูปที่ 4.12 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 1-16 | Import คลาสต่างๆที่จำเป็นเข้ามาในคลาสนี้ |
| 18-42 | ประกาศตัวแปรที่จำเป็น เช่น  1. GLSurfaceView, Renderer จะเป็นคลาสสำหรับเชื่อมต่อกับ OpenGL ES  2. GameObjectManager จะเป็นคลาสที่เก็บโมเดลทั้งหมด  3. Matrix (float[]) จะเก็บเมตริกซ์สำหรับการ Projection และ ModelView  เป็นต้น |

****

**รูปที่ 4.13** ฟังก์ชันเริ่มต้นของ Activity

จากโค๊ดตัวอย่างรูปที่ 4.13 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 45-56 | ประกาศ Object ต่างๆ จากตัวแปรในบรรทัดที่ 18-42 |
| 18-42 | ประกาศ Object Renderer และทำการ Implements ฟังก์ชันพื้นฐานของ OpenGL ES |

****

**รูปที่ 4.14** สร้างเธรด

จากโค๊ดตัวอย่างรูปที่ 4.14 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 73-119 | สร้างเธรดสำหรับการ Update |
| 90-106 | นับเฟรมเรท |

****

**รูปที่ 4.15** การตั้งค่า Aspect Ratio และการวาดภาพ

จากโค๊ดตัวอย่างรูปที่ 4.15 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 121-129 | เป็นฟังก์ชันพื้นฐานของ OpenGL ES ชื่อว่า onSurfaceChanged โดยจะทำงานเมื่อมีการปรับเปลี่ยนขนาดหน้าจอ หรือ มีการเปลี่ยนแปลง Aspect Ratio |
| 130-145 | เป็นฟังก์ชันวาดรูป หรือวาดโมเดลสามมิติ โดยเริ่มต้นการทำงานต้องมีการ เคลียร์สีพื้นหลังก่อน หลังจากนั้นค่อยวาด |

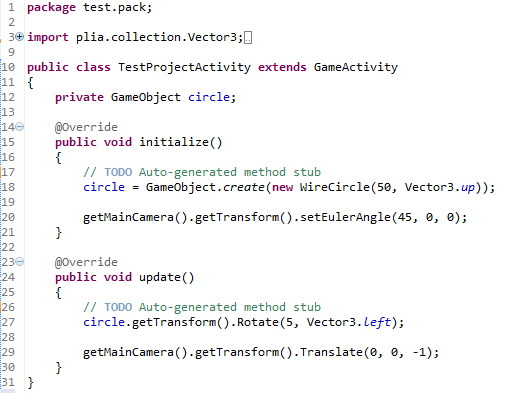
****

**รูปที่ 4.16** ประกาศฟังก์ชัน Initialize, Update และ Getter/Setter ของกล้อง

จากโค๊ดตัวอย่างรูปที่ 4.16 สามารถนำมาอธิบายได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 151 | ประกาศฟังก์ชัน Initialize ไว้ให้ผู้ใช้เฟรมเวิร์คได้ใส่การทำงานลงไป |
| 152 | ประกาศฟังก์ชัน Update ไว้ให้ผู้ใช้เฟรมเวิร์คได้ใส่การทำงานลงไป |
| 154-157 | ฟังก์ชันเรียกใช้งานกล้องหลัก |
| 159-162 | ฟังก์ชันติดตั้งกล้องใหม่แทนกล้องเก่า |

* **การเขียนโปรแกรมทางผู้ใช้เฟรมเวิร์ค**

****

**รูปที่ 4.17** ฟังก์ชันพื้นฐานของเฟรมเวิร์ค

จากโค๊ดตัวอย่างรูปที่ 4.17 สามารถนำมาอธิบายได้ดังนี้

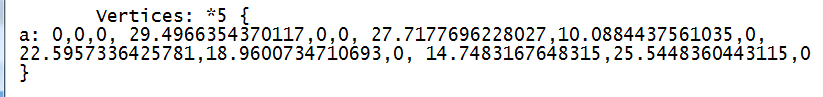
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 15-21 | สร้าง GameObject ขึ้นมา 1 Object และกำหนดมุมมองของกล้องที่ แกน X = 45 องศา |
| 27 | กำหนดให้ GameObject หมุนรอบแกน X ทุกๆเฟรม ทีละ 5 องศา |
| 29 | กำหนดให้ กล้อง เคลื่อนที่ถอยหลังเฟรมละ 1 หน่วย |

* + 1. **Load 3D Model from FBX file format**

ในการอ่านข้อมูลจากไฟล์ FBX จะทำการอ่านข้อมูลทีละบรรทัด โดยจะยกตัวอย่างการอ่าน Vertex Table จากไฟล์ FBX ดังรูปที่ 5.5



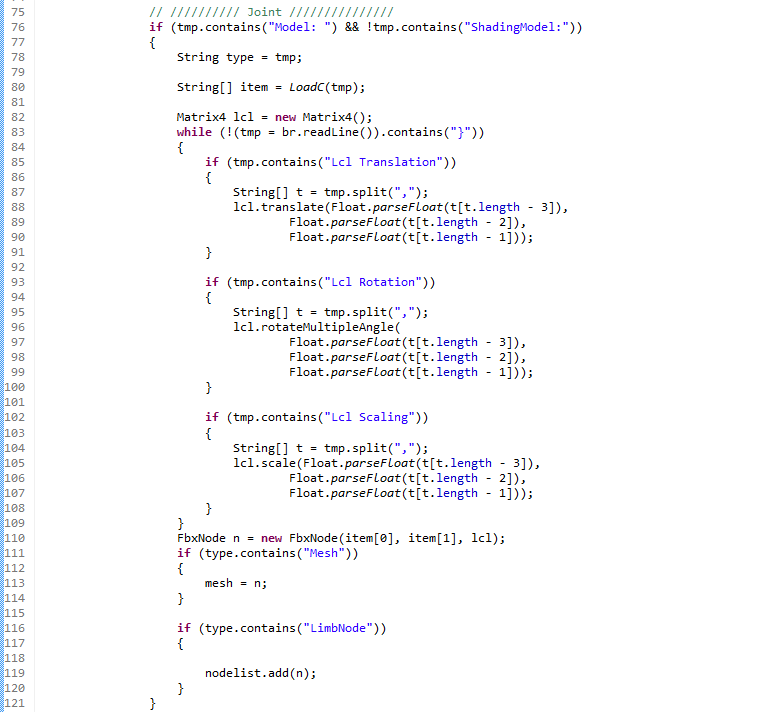
**รูปที่ 4.18** ตัวอย่างการอ่านข้อมูล FBX และเก็บลงในโปรแกรม



**รูปที่ 4.19** ตัวอย่าง Vertex Table ในไฟล์ FBX

จากโค๊ดตัวอย่างรูปที่ 4.18 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 5 - 18 | เปิดไฟล์ FBX format |
| 21-42 | สร้างตัวแปรเก็บข้อมูลของโมเดล |
| 45 | ใช้ While Loop ในการอ่านข้อมูลทีละบรรทัด (อ่านจนกว่าจะไม่มีข้อมูล หรือได้ค่า null) |
| 47-73 | อ่านข้อมูลของ Mesh |

****

**รูปที่ 4.20** ตัวอย่างการอ่านข้อมูล Node (Mesh, Bone Local Transformation)

จากโค๊ดตัวอย่างรูปที่ 4.20 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 76-110 | อ่านข้อมูล Local Transformation ของ Mesh และ Bones |
| 111-119 | กำหนดว่า Local Transformation ที่อ่านมา เป็นของ Mesh หรือ Bone |

****

**รูปที่ 4.21** ตัวอย่างการอ่านข้อมูล Bind Pose Matrix

จากโค๊ดตัวอย่างรูปที่ 4.21 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 123-153 | อ่านข้อมูล BindPose |

****

**รูปที่ 4.22** ตัวอย่างการอ่านข้อมูล Deformer

จากโค๊ดตัวอย่างรูปที่ 4.22 สามารถอธิบายคำสั่งได้ดังนี้

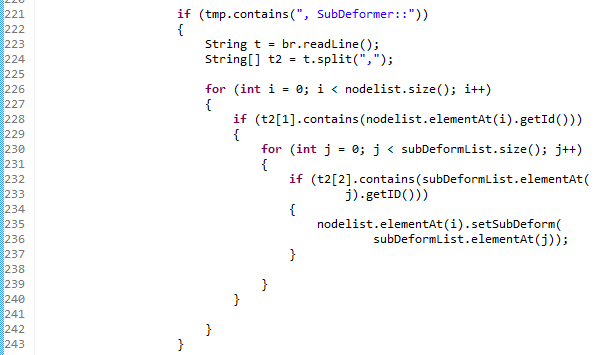
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 155 - 186 | เก็บข้อมูล Deformer ประกอบด้วย BoneIndex, BoneWeight, TransformMatrix, TransformLink Matrix |

****

**รูปที่ 4.23** ตัวอย่างการอ่านข้อมูล Scene Graph

จากโค๊ดตัวอย่างรูปที่ 4.23 สามารถอธิบายคำสั่งได้ดังนี้

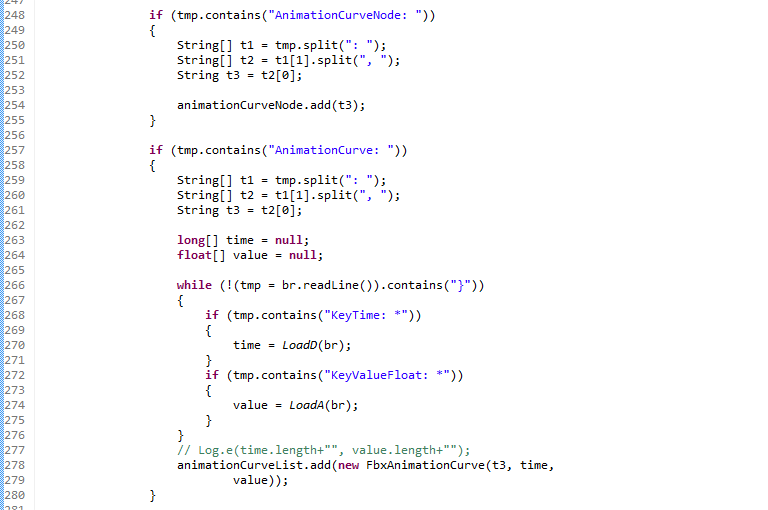
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 190-194 | เก็บข้อมูลรหัส RootNode |
| 196-217 | อ่านข้อมูล Child Node และ Parent Node |

****

**รูปที่ 4.24** ตัวอย่างการอ่านข้อมูล SubDeformer

จากโค๊ดตัวอย่างรูปที่ 4.24 สามารถอธิบายคำสั่งได้ดังนี้

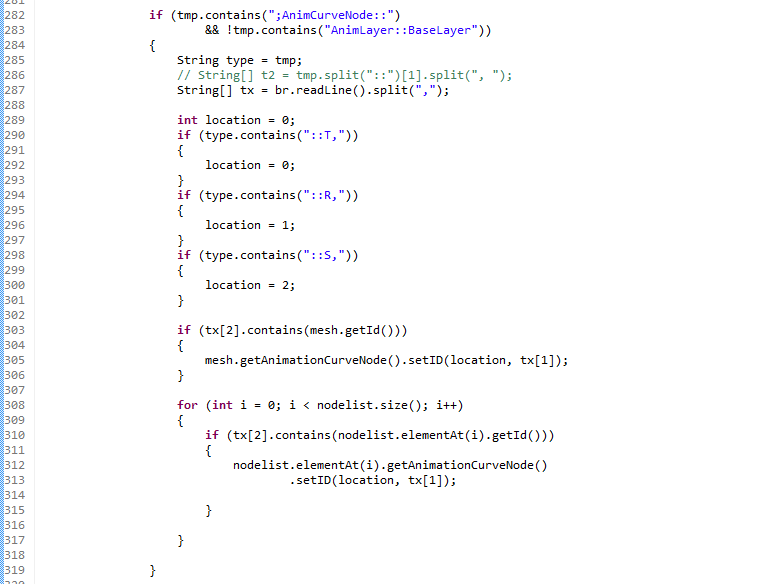
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 155 - 186 | เก็บข้อมูล SubDeformer |

****

**รูปที่ 4.25** ตัวอย่างการอ่านข้อมูล AnimationCurve

จากโค๊ดตัวอย่างรูปที่ 4.25 สามารถอธิบายคำสั่งได้ดังนี้

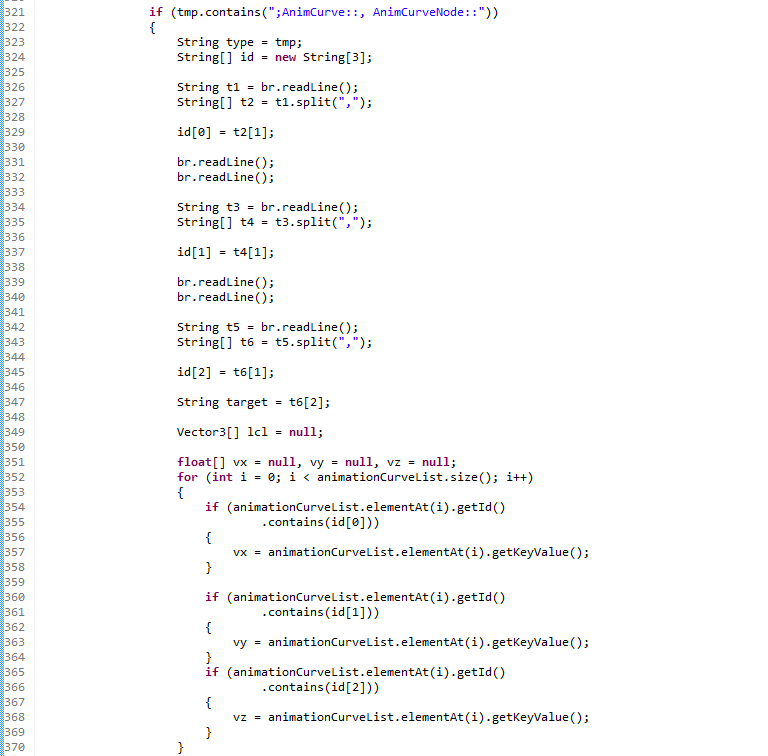
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 248-255 | เก็บข้อมูล Animation Curve Node |
| 257-280 | อ่านข้อมูล Animation Curve |
| 266-278 | เก็บข้อมูล Keytime และ Value ของ Animation Curve |

****

**รูปที่ 4.26** ตัวอย่างการอ่านข้อมูลความสัมพันธ์ระหว่าง Node กับ Animation Curve Node

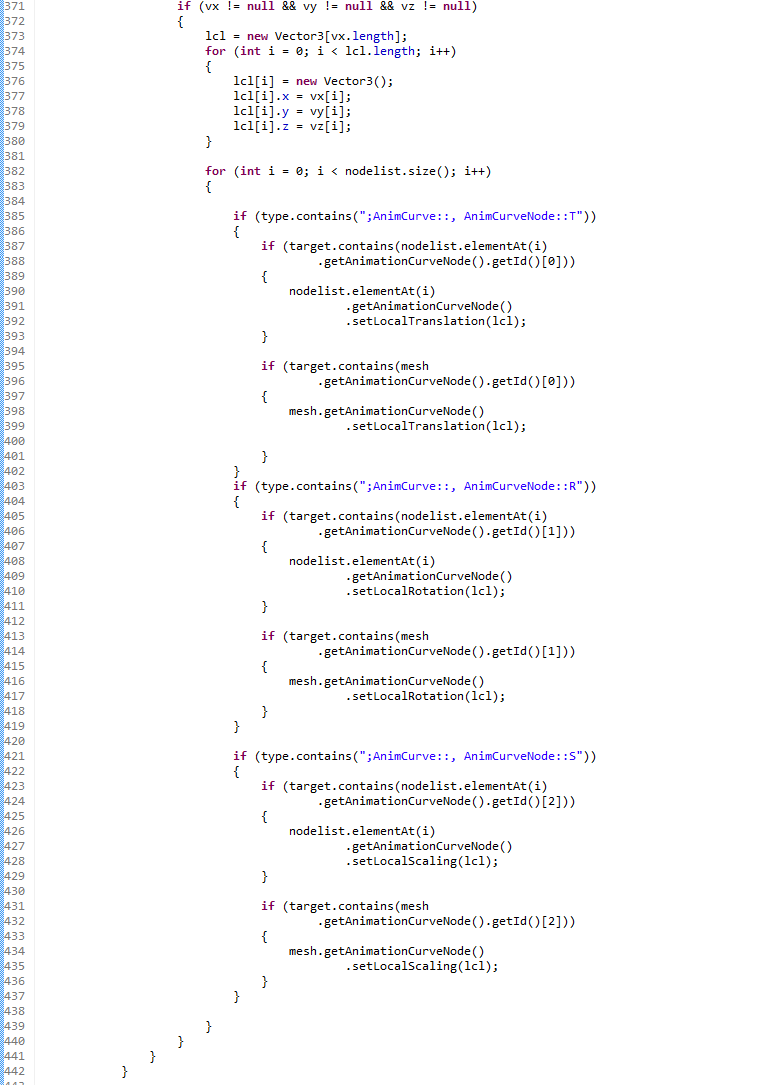
จากโค๊ดตัวอย่างรูปที่ 4.26 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 282-319 | เก็บข้อมูลความสัมพันธ์ระหว่าง Node กับ Animation Curve Node |
| 290-301 | ตรวจสอบว่า Animation Curve Node เป็นประเภทใด   1. Translation 2. Rotation 3. Scaling |

**รูปที่ 4.27** ตัวอย่างการอ่านข้อมูลความสัมพันธ์ระหว่าง Animation Curve Node กับ Animation Curve

จากโค๊ดตัวอย่างรูปที่ 4.27 สามารถอธิบายคำสั่งได้ดังนี้

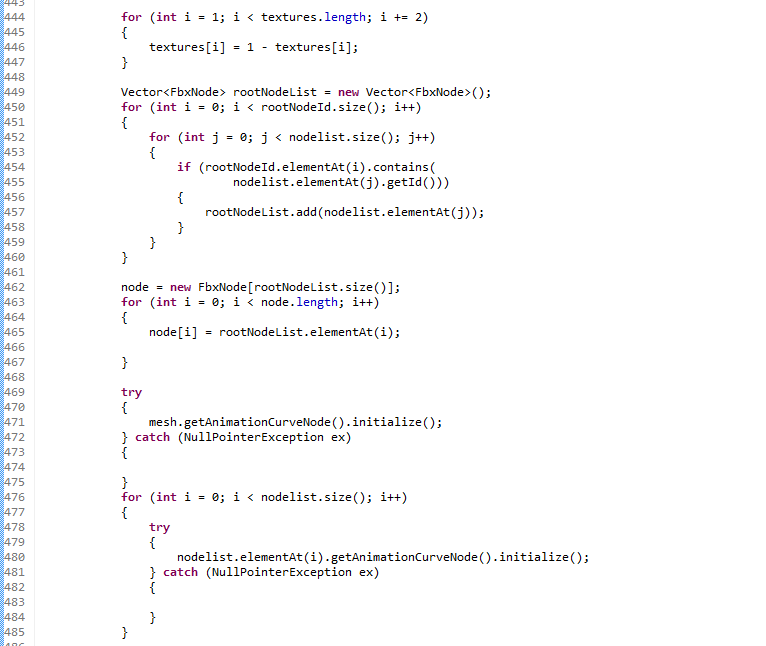
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 321-370 | เก็บข้อมูลความสัมพันธ์ระหว่าง Animation Curve Node กับ Animation Curve |

****

**รูปที่ 4.28** ตัวอย่างการรวม Animation Curve ไว้ที่ Animation Curve Node

จากโค๊ดตัวอย่างรูปที่ 4.28 สามารถอธิบายคำสั่งได้ดังนี้

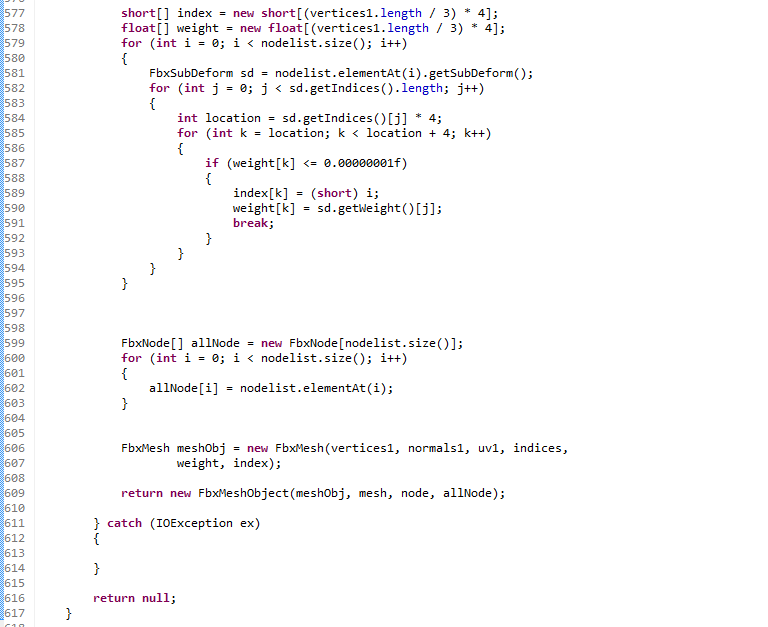
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 371-442 | นำ Animation Curve 3 ประเภท 1. Translation 2. Rotation 3.Scaling  มารวมกันเป็น 1 Animation Curve Node |

****

**รูปที่ 4.29** ตัวอย่างการเรียงข้อมูลที่อ่านมาจากไฟล์ FBX

จากโค๊ดตัวอย่างรูปที่ 4.30 สามารถอธิบายคำสั่งได้ดังนี้

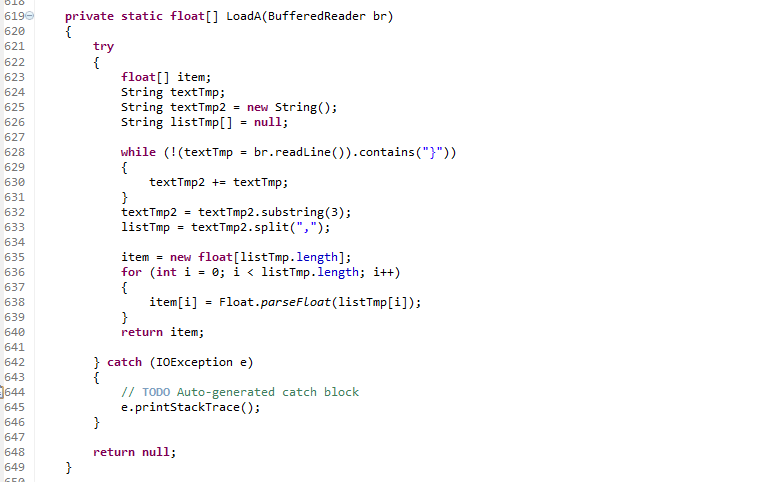
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 444-447 | กลับด้านแกนพิกัด แกน v (จากแกน uv) เนื่องจากข้อมูลเริ่มต้นเมื่อนำมาใช้จะกลับด้านแกน v ทำให้ก่อนนำไปใช้ต้องทำการกลับด้านเสียก่อน |
| 449-460 | ทำการสร้างซีนกราฟของกระดูก โดยมี RootNode เป็นโหนดเริ่มต้นของแต่ละกราฟ |

****

**รูปที่ 4.31** ตัวอย่างการเรียงข้อมูลที่อ่านจาก FBX

จากโค๊ดตัวอย่างรูปที่ 4.31 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 577-595 | สร้าง Object FbxSubDeform สำหรับเก็บข้อมูล BoneIndex, BoneWeight ที่อ่านมาได้ |
| 606 | สร้าง Object ของ Mesh |

****

**รูปที่ 4.32** ตัวอย่างการอ่านข้อมูลอาร์เรย์

จากโค๊ดตัวอย่างรูปที่ 4.32 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 619-649 | เป็นฟังก์ชันวนลูปอ่านข้อมูลจากรูป 5.15 เพื่อเก็บตัวแปรที่โหลดมาเป็น Array แล้วส่งค่ากลับ |

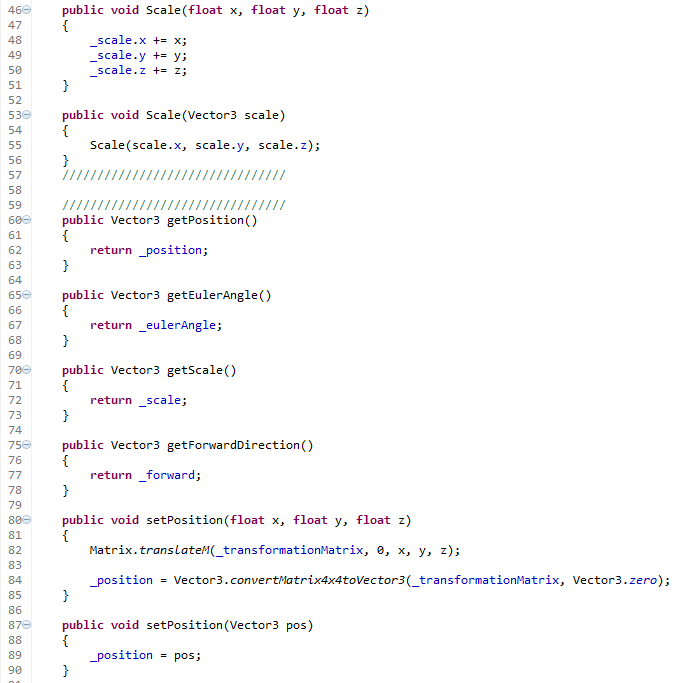
* + 1. **Manage Game Object**
* **การเขียนโปรแกรมควบคุมตำแหน่ง , การหมุน และอัตราส่วนของ GameObject บน OpenGL ES**

****

**รูปที่ 4.33** ตัวอย่างการสร้างคลาส Transform

จากโค๊ดตัวอย่างรูปที่ 4.33 สามารถอธิบายคำสั่งได้ดังนี้

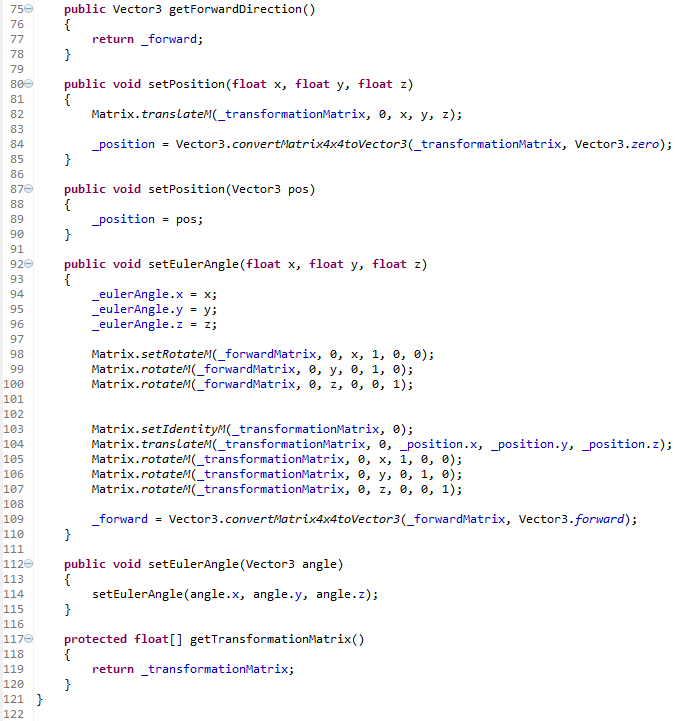
|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 9-16 | ประกาศตัวแปรที่จำเป็นของ Transform |
| 18-28 | ฟังก์ชัน Translate GameObject |
| 29-44 | ฟังก์ชัน Rotate GameObject |

****

**รูปที่ 4.34** ตัวอย่างการสร้างคลาส Transform 2

จากโค๊ดตัวอย่างรูปที่ 4.34 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 46-56 | ฟังก์ชันกำหนดอัตราส่วนของ GameObject |
| 60-63 | ฟังก์ชันเรียกดูข้อมูลพิกัดของ GameObject |
| 65-68 | ฟังก์ชันเรียกดูทิศทางการหมุนของ GameObject |
| 70-73 | ฟังก์ชันเรียกดูอัตราส่วนของ GameObject |
| 75-78 | ฟังก์ชันเรียกดู เวกเตอร์ทิศทาง ด้านหน้าของ GameObject |

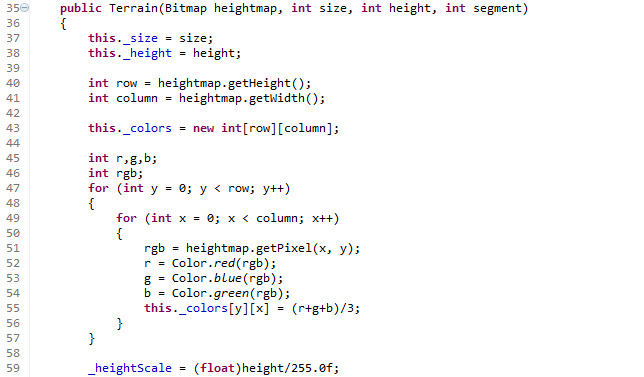


**รูปที่ 4.35** ตัวอย่างการสร้างคลาส Transform 3

จากโค๊ดตัวอย่างรูปที่ 4.35 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 80-90 | ฟังก์ชันกำหนดพิกัดของ GameObject |
| 92-116 | ฟังก์ชันกำหนดการหมุนของ GameObject |
| 117-120 | ฟังก์ชันเรียกดู Transformation Matrix |

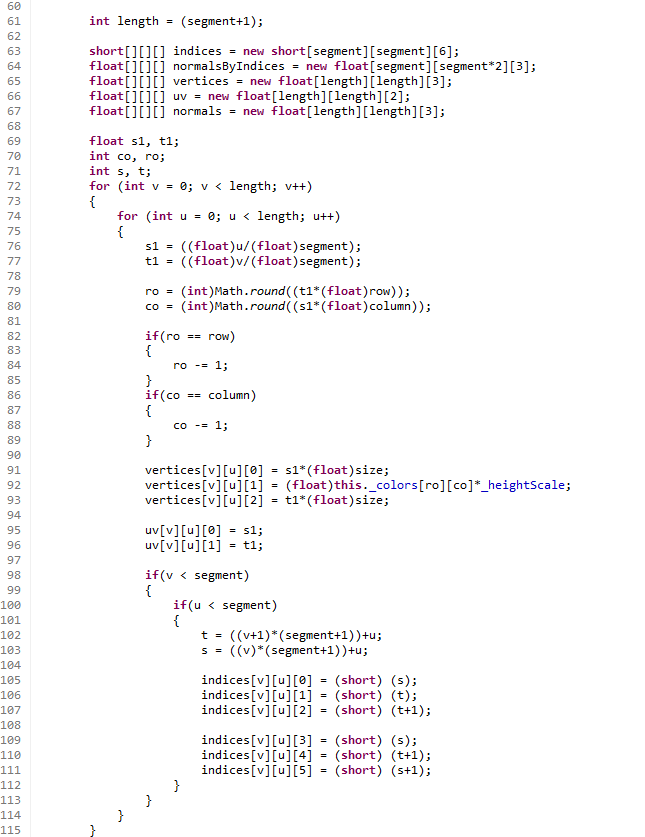
* **การสร้างภูมิประเทศจาก Heightmap**

****

**รูปที่ 4.36** ตัวอย่างการตรวจสอบค่าสีจาก Heightmap

จากโค๊ดตัวอย่างรูปที่ 4.36 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 40-43 | คำนวณหาแถวและหลักของภาพที่ส่งเข้ามา และสร้างอาร์เรย์ที่มีขนาดเดียวกับจำนวนพิกเซล |
| 47-57 | วนลูปตรวจสอบค่าสีและเก็บไว้ในอาร์เรย์ |
| 59 | คำนวณอัตราส่วนความสูงที่ผู้ใช้กำหนดกับค่าสีขาว |

****

**รูปที่ 4.37** ตัวอย่างการเขียนโปรแกรมสร้างพื้นผิวให้กับภูมิประเทศ

จากโค๊ดตัวอย่างรูปที่ 4.37 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 63-67 | ประกาศตัวแปรสำหรับการสร้างพื้นผิว |
| 91-93 | กำหนดพิกัดของจุดแต่ละจุดของพื้นผิว |
| 95-97 | กำหนดพิกัดของ uv ของแต่ละจุด |
| 98-113 | สร้างตาราง face table ของแต่ละโพลิกอน |

* **การสร้างภูมิประเทศจาก Heightmap ด้วย Vertex Shader**

จากการศึกษาและทดลองสร้างภูมิประเทศจาก Heightmap ในข้อที่ 5.10.2 ทำให้พบปัญหาดังนี้

* + - 1. ใช้เวลาในการสร้าง Buffer นานและมีขนาดใหญ่มาก
      2. การส่งข้อมูลจาก Buffer ที่มีขนาดใหญ่ไปที่ Vertex Shader จะทำได้ช้า และส่งผลต่อเฟรมเรทในการแสดงผลด้วย
      3. ถ้า Segment มีจำนวนมากจะทำให้หน่วยความจำเต็มได้

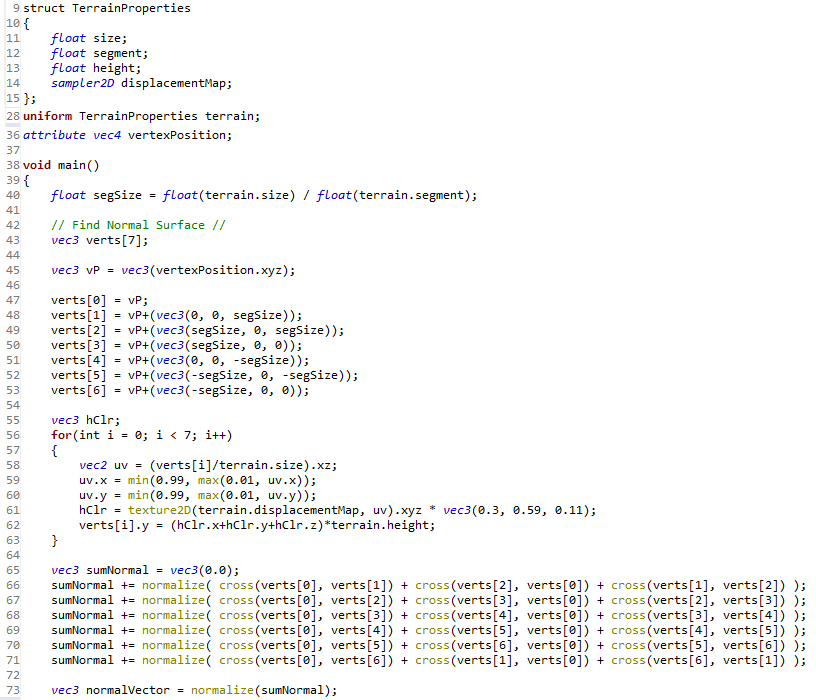
แนวทางการแก้ไขปัญหามีดังนี้



**รูปที่ 4.38** ตัวอย่างการเขียนโปรแกรมสร้างพื้นผิวให้กับภูมิประเทศ

จากโค๊ดตัวอย่างรูปที่ 4.38 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 38-48 | ประกาศตัวแปรสำหรับการสร้างพื้นผิว |
| 50-51 | สร้าง Buffer ของ Vertices และ Face Table |
| 53-79 | กำหนดพิกัดของจุดแต่ละจุดของพื้นผิว และใส่ลงใน Buffer |

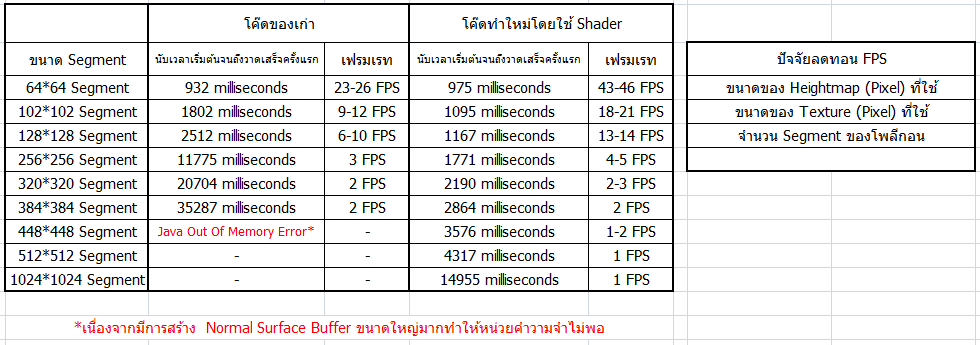


**รูปที่ 4.39** ตัวอย่างการคำนวณเวกเตอร์ปกติของแต่ละจุดใน Vertex Shader

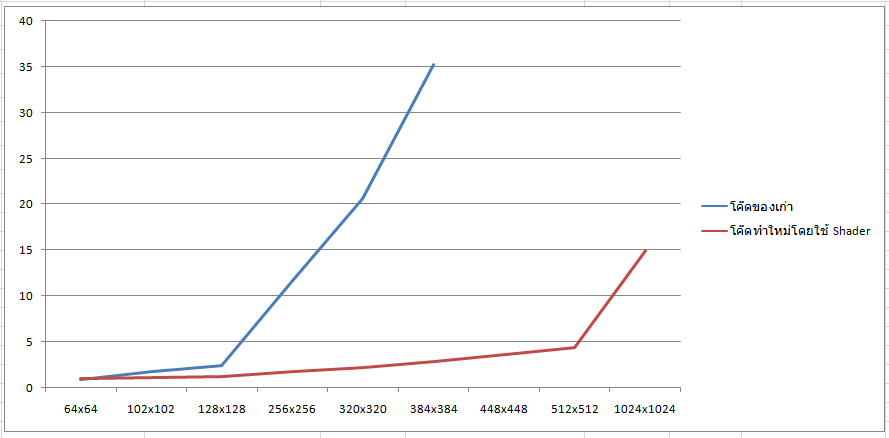
จากโค๊ดตัวอย่างรูปที่ 4.39 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 9-36 | ประกาศตัวแปรสำหรับการสร้างพื้นผิว |
| 47-53 | หาจุดเพื่อนบ้านในแกน x, z |
| 55-63 | หาจุดเพื่อนบ้านในแกน y |
| 65-73 | คำนวณหาเวกเตอร์ปกติของแต่ละโพลิกอนจากจุดที่ได้มาในบรรทัดที่ 55-63 และนำมารวมกัน หลังจากนั้นทำเป็นเวกเตอร์ปกติอีกครั้งก็จะได้ เวกเตอร์ปกติของจุด |

จากการศึกษาและทดลองและนำมาเปรียบเทียบกับหัวข้อที่ 5.10.2 ได้ผลลัพธ์ดังนี้



**รูปที่ 4.40** ตารางเปรียบเทียบประสิทธิภาพระหว่างหัวข้อที่ 5.10.2 และ 5.10.3

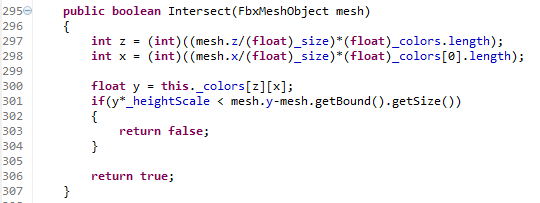


**รูปที่ 4.41** กราฟเปรียบเทียบประสิทธิภาพระหว่างหัวข้อที่ 5.10.2 และ 5.10.3

สรุปผลการทดลอง

สาเหตุที่ทำให้การทำงานในหัวข้อ 5.10.2 ช้ากว่า 5.10.3 มาจากการที่ 5.10.2 มีการสร้าง Buffer ของเวกเตอร์ปกติของจุดทุกจุด (ยิ่งจำนวน Segment มีจำนวนมาก การเติบโตทางเวลายิ่งมาก) แล้วจึงส่งไปให้ Vertex Shader ใช้งาน แต่ในหัวข้อที่ 5.10.3 จะไม่มีการคำนวณเวกเตอร์ปกติเลย แต่จะใช้การคำนวณใน Vertex Shader โดยตรง

* **การตรวจสอบการชนระหว่างวัตถุกับภูมิประเทศ**

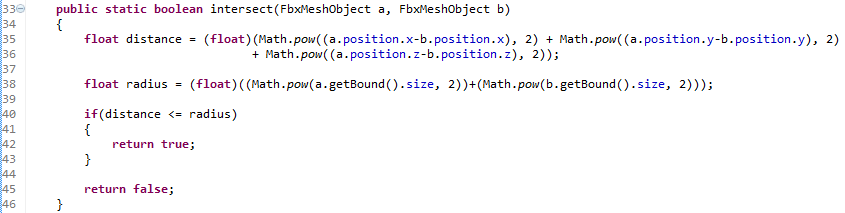
****

**รูปที่ 4.42** ตัวอย่างการตรวจสอบการชนระหว่างวัตถุกับภูมิประเทศ

จากโค๊ดตัวอย่างรูปที่ 4.42 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 297-300 | คำนวณตำแหน่งแกนพิกัดระหว่างวัตถุกับภูมิประเทศ |
| 301-306 | ตรวจสอบความสูงระหว่างวัตถุกับภูมิประเทศ |

* **การตรวจสอบการชนระหว่างทรงกลมสองทรงกลม**

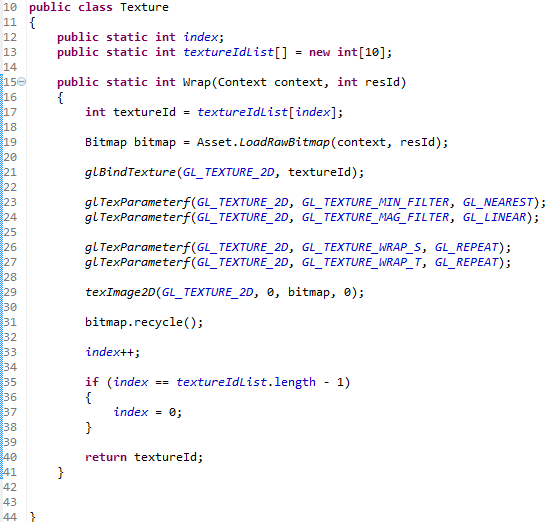
****

**รูปที่ 4.43** ตัวอย่างการตรวจสอบการชนระหว่างทรงกลมสองวงกลม

จากโค๊ดตัวอย่างรูปที่ 4.43 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 35-36 | คำนวณหาระยะห่างระหว่างสองทรงกลม |
| 38 | คำนวณหารัศมีของทั้งสองทรงกลม |
| 40-43 | ตรวจสอบว่าระยะห่างของสองทรงกลมว่ามีการชนกันอยู่ |

* **การเขียนโปรแกรมกำหนดเทกเจอร์ ให้กับ GameObject**

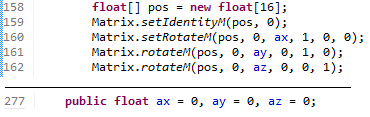
****

**รูปที่ 4.44** ตัวอย่างการสร้างคลาสสำหรับโหลด Texture

จากโค๊ดตัวอย่างรูปที่ 4.44 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 10-41 | Static Class สำหรับโหลด Texture และสร้าง Texture ID เพื่อนำไปใช้ในการติดตั้งเทกเจอร์ให้กับโมเดล |

* + 1. **Manage Light**
       - **เขียนโปรแกรมควบคุมทิศทางของแสง**

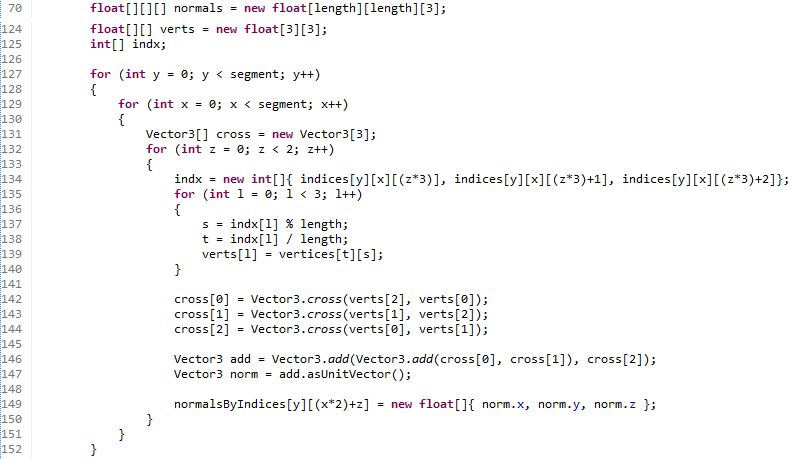
****

**รูปที่ 4.45** ตัวอย่างการเขียนโปรแกรมคำนวณทิศทางของแสง

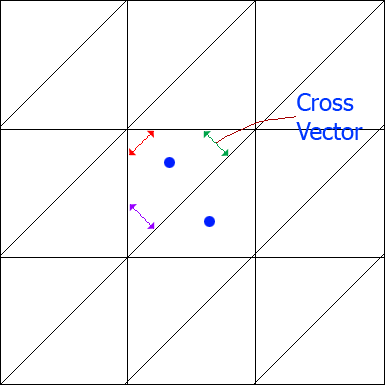
จากโค๊ดตัวอย่างรูปที่ 4.45 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 277 | ประกาศตัวแปรสำหรับกำหนดองศาการหมุนในแกน x,y,z |
| 158-159 | สร้างเมตริกซ์เอกลักษณ์ |
| 160-162 | ทำการหมุนทิศทางตามแกน x,y,z |

* + - * **การเขียนโปรแกรมคำนวณเวกเตอร์ปกติของภูมิประเทศเพื่อใช้คำนวณแสง**

****

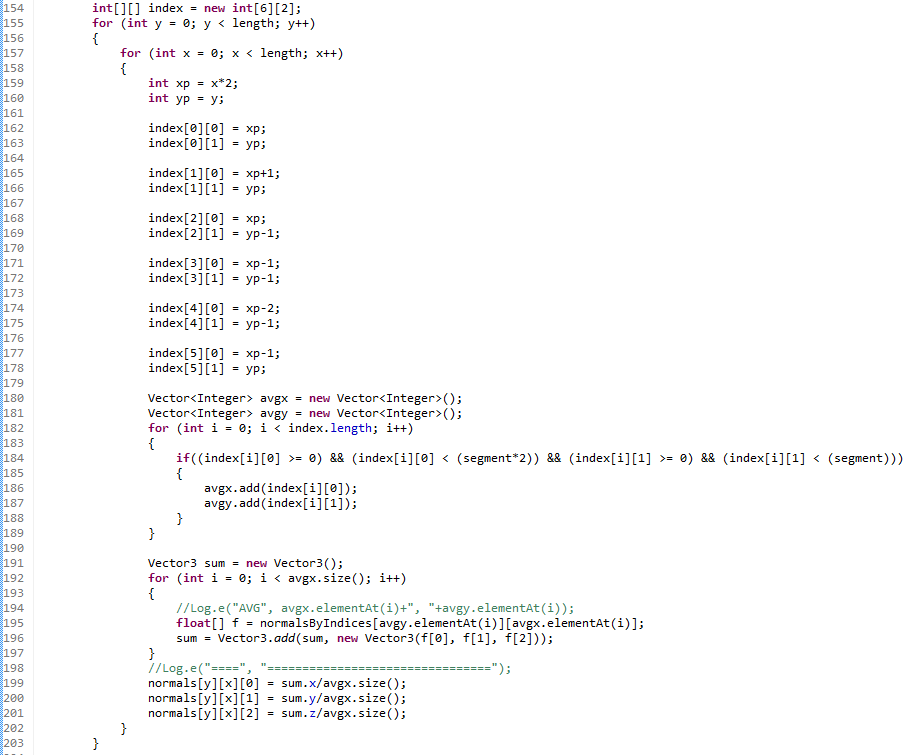
**รูปที่ 4.46** ตัวอย่างการเขียนโปรแกรมหา Normal Surface



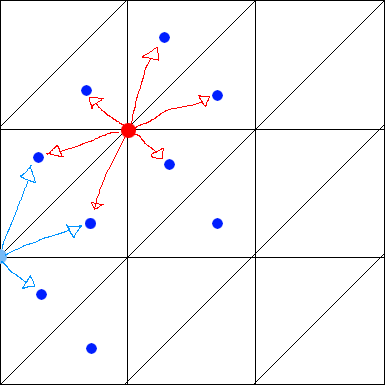
**รูปที่ 4.47** ตัวอย่างการหา Normal Surface (รูปประกอบ)

จากโค๊ดตัวอย่างรูปที่ 4.46 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 127-151 | นำเวกเตอร์ของจุดของแต่ละโพลีกอนมาครอสกัน และนำมาหาค่าเฉลี่ย จะได้ เวกเตอร์ปกติของโพลีกอนของแต่ละโพลีกอน รูปที่ 5.35 |



**รูปที่ 4.47** ตัวอย่างการเขียนโปรแกรมคำนวณ Normal Vector ของจุด



**รูปที่ 4.48** ตัวอย่างการคำนวณ Normal Vector ของจุด (รูปประกอบ)

จากโค๊ดตัวอย่างรูปที่ 4.47 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 127-151 | นำเวกเตอร์ปกติของแต่ละโพลีกอนมาบวกกัน และนำมาหาค่าเฉลี่ย จะได้ เวกเตอร์ปกติของแต่ละจุด รูปที่ 5.37 |

* + - * **การเขียนโปรแกรมคำนวณแสงแบบ Directional Light บน Shader**

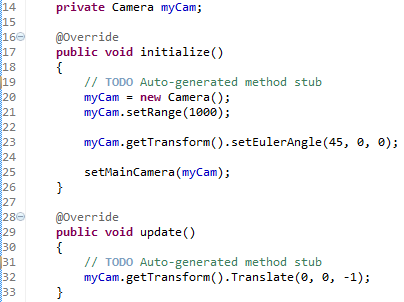
**E:\98. Project\99. Temporary\Document\01. Word\code\directionalShader.png**

**รูปที่ 4.49** ตัวอย่างการคำนวณแสงใน Shader

จากโค๊ดตัวอย่างรูปที่ 4.49 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 1 | หาเวกเตอร์ทิศทางของแสง |
| 2 | นำเวกเตอร์ปกติของจุดและเวกเตอร์ปกติของทิศทางแสงมาดอทกันเพื่อหาความเข้มของแสง |

* + 1. **Manage Camera**
       - **การเขียนโปรแกรมกำหนดมุมมองของกล้อง**

****

**รูปที่ 4.50** ตัวอย่างการควบคุมกล้องโดยผู้ใช้เฟรมวิร์ค

จากโค๊ดตัวอย่างรูปที่ 4.50 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 14 และ 20 | สร้างออบเจ็กต์ของกล้อง |
| 21 | กำหนดระยะการมองเห็นของกล้อง |
| 23 | กำหนดทิศทางของกล้อง |
| 25 | กำหนดกล้องที่สร้างขึ้นมาให้เป็นกล้องหลักของโปรแกรม |
| 32 | กำหนดให้กล้องเคลื่อนที่ถอยหลังทุกๆเฟรม เฟรมละ 1 หน่วย |

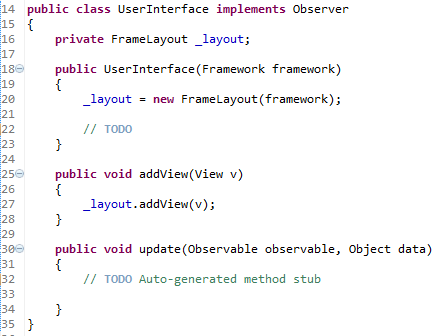
**E:\98. Project\99. Temporary\Document\01. Word\code\cam001.png**

**รูปที่ 4.51** ตัวอย่างการกำหนดมุมมองของกล้อง

จากโค๊ดตัวอย่างรูปที่ 4.51 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 1 | กำหนดระยะการมองเห็นของกล้อง |
| 2 | กำหนดพิกัดของกล้อง และกำหนดทิศทางของกล้อง |

* + 1. **Manage Graphics User Interface**
       - **คลาสต้นแบบสำหรับ User Interface**

****

**รูปที่ 4.52** คลาส User Interface

จากโค๊ดตัวอย่างรูปที่ 4.52 สามารถอธิบายคำสั่งได้ดังนี้

|  |  |
| --- | --- |
| บรรทัดที่ | คำอธิบาย |
| 16, 20 | ประกาศเลย์เอาต์หลักของ UI |
| 25 | ฟังก์ชันสำหรับเพิ่มวิวย่อยเช่น ปุ่ม, ข้อความ |
| 30 | ฟังก์ชันสำหรับรับข้อมูลจากคอนโทรลเลอร์เพื่ออัพเดทเลย์เอาต์ |

**บรรณานุกรม**

[1] OpenGL ES Super Bible  
[2] http://en.wikipedia.org/wiki/Artificial\_intelligence  
[3] http://en.wikipedia.org/wiki/Collision\_detection  
[4] เอกสารประกอบการสอนวิชาปัญญาประดิษฐ์สำหรับเกมคอมพิวเตอร์ โดย   
อาจารย์อาจหาญ วงศ์ศรีชา  
[5] OpenGL ES 2.0 Programming Guide

[6] เอกสารประกอบการสอนวิชาคอมพิวเตอร์กราฟิก 1 โดย อาจารย์วีระวัฒน์ ตันติศิริวัฒน์

[7] http://code.google.com/p/min3d/

[8] http://www.rozengain.com/blog/2011/08/23/announcing-rajawali-an-opengl-es-2-0-based-3d-framework-for-android/

[9] http://code.google.com/p/dwarf-fw/

[10] http://code.google.com/p/catcake/

[11] http://libgdx.badlogicgames.com/

[12] http://content.gpwiki.org/index.php/OpenGL:Tutorials:Basic\_Bones\_System

[13] http://www.oakki.com/cgm276/ShaderPrograming.pdf