Assignment 04: การเรียนรู้ Data Structures สำหรับ Game Development

🎯 จุดประสงค์การเรียนรู้

- เรียนรู้การใช้งาน LinkedList ใน C#
- เข้าใจการทำงานของ Hashtable และ Dictionary
- นำ Data Structures มาใช้ในการแก้ปัญหาในเกม
- จัดการข้อมูลแบบไดนามิกและมีประสิทธิภาพ
- เขียน code ที่ปลอดภัยและมีประสิทธิภาพในการจัดการข้อมูล

📚 โครงสร้างของ Assignment

- **Lecture Methods (3 methods)** - การ implement ฝึกหัดด้วย Data Structures พื้นฐาน พร้อมกันในห้องเรียน
- **Assignment Methods (11 methods)** - การประยุกต์ใช้ Data Structures ในสถานการณ์เกม

Lecture Methods

Methods เหล่านี้แสดงแนวคิด Data Structures พื้นฐาน Implement เพื่อฝึกหัดแต่จะไม่มีการให้คะแนน

1. LCT01_SyntaxLinkedList

วัตถุประสงค์: แสดงการประกาศและใช้งาน LinkedList พื้นฐาน รวมถึงการเพิ่ม ลบ และเข้าถึงข้อมูล

Method Signature:

```
void LCT01_SyntaxLinkedList()
```

Logic ที่ต้อง implement:

- สร้าง LinkedList ของประเภท string
- เพิ่มข้อมูลที่ท้ายของ LinkedList ด้วย `AddLast("Node 1")` และ `AddLast("Node 2")`
- เพิ่มข้อมูลที่ต้นของ LinkedList ด้วย `AddFirst("Node 0")`
- แสดงเนื้อหาใน LinkedList โดยใช้ foreach loop
- เข้าถึงข้อมูลใน LinkedList โดยใช้ First, Last, และ Find
- เพิ่ม node ก่อนและหลัง node ที่กำหนดด้วย `AddBefore` และ `AddAfter`
- ลบ Node แรกด้วย `RemoveFirst()` และลบ Node ตามค่าที่กำหนดด้วย `Remove()`
- แสดงผลตามรูปแบบที่กำหนด

Test Case:

- **Input:** ไม่มี parameters
- **Expected Output:** ประกอบด้วย "LinkedList ...", รายชื่อ nodes, "first", "last", "firstNode.Previous is null", "lastNode.Next is null", และการแสดงผลหลังการเพิ่มและลบ

2. LCT02_SyntaxHashTable

วัตถุประสงค์: แสดงการใช้งาน Hashtable รวมถึงการเพิ่ม เข้าถึง และลบข้อมูล

Method Signature:

```
void LCT02_SyntaxHashTable()
```

Logic ที่ต้อง implement:

- สร้าง Hashtable
- เพิ่มข้อมูลลงใน Hashtable ด้วย keys ต่างชนิด (int และ string)
- เข้าถึงข้อมูลใน Hashtable ด้วย Key และ cast เป็น string
- แสดงข้อมูลใน Hashtable โดยใช้ foreach DictionaryEntry
- ตรวจสอบการมีอยู่ของ Key ด้วย `ContainsKey()`
- ลบข้อมูลออกจาก Hashtable ด้วย `Remove()`
- แสดงผลตามรูปแบบที่กำหนด

Test Case:

- **Input:** ไม่มี parameters
- **Expected Output:** ประกอบด้วย "fruit1: Apple", "fruit2: Banana", "badFruit: Rotten Tomato", "found 2", "table ..."

3. LCT03_SyntaxDictionary

วัตถุประสงค์: แสดงการใช้งาน Dictionary รวมถึงการเพิ่ม เข้าถึง และลบข้อมูล

Method Signature:

```
void LCT03_SyntaxDictionary()
```

Logic ที่ต้อง implement:

- สร้าง Dictionary โดยระบุชนิดของ Key เป็น int และ Value เป็น string
- เพิ่มข้อมูลลงใน Dictionary ด้วย `Add()` และ indexer
- แสดงข้อมูลใน Dictionary โดยใช้ foreach KeyValuePair
- ตรวจสอบการมีอยู่ของคีย์และดึงค่าออกมาด้วย `ContainsKey()`
- ดึง key ออกมาทั้งหมดด้วย `Keys` property
- ลบข้อมูลออกจาก Dictionary ด้วย `Remove()`
- Clear ข้อมูลใน Dictionary ด้วย `Clear()`
- แสดงผลตามรูปแบบที่กำหนด

Test Case:

- **Input:** ไม่มี parameters

- **Expected Output:** ประกอบด้วย "Dictionary has 3 keys", "has key 1 : True", "value of key 1 : Apple", "All keys in dictionary:", "Dictionary has 2 keys"

Assignment Methods

Methods เหล่านี้เป็นการประยุกต์ใช้ Data Structures ในสถานการณ์เกม และจะมีการให้คะแนน

AS01_CountWords

วัตถุประสงค์: นับจำนวนคำใน array ของ strings โดยใช้ Dictionary

Method Signature:

```
void AS01_CountWords(string[] words)
```

Logic ที่ต้อง implement:

- ใช้ Dictionary<string, int> เพื่อนับความถี่ของแต่ละคำ
- นับแต่ละคำโดยใช้ for loop วนลูปผ่าน array
- ถ้าคำมีอยู่แล้วให้เพิ่มค่า count ถ้าไม่มีให้เพิ่มคำใหม่
- แสดงผลลัพธ์โดยแปลง Keys และ Values เป็น array และใช้ for loop
- แสดงผลในรูปแบบ "word: '{word}' count: {count}"

Test Cases:

1. **Input:** `["apple", "banana", "apple", "cherry", "banana", "apple"]` **Expected Output:**

   ```
   word: 'apple' count: 3
   word: 'banana' count: 2
   word: 'cherry' count: 1
   ```

2. **Input:** `["hello", "world", "hello"]` **Expected Output:**

   ```
   word: 'hello' count: 2
   word: 'world' count: 1
   ```

3. **Input:** `["test"]` **Expected Output:**

   ```
   word: 'test' count: 1
   ```

AS02_CountNumber

วัตถุประสงค์: นับจำนวนตัวเลขใน array ของ integers โดยใช้ Dictionary

Method Signature:

```
void AS02_CountNumber(int[] numbers)
```

Logic ที่ต้อง implement:

- ใช้ Dictionary<int, int> เพื่อนับความถี่ของแต่ละตัวเลข
- นับแต่ละตัวเลขโดยใช้ for loop วนลูปผ่าน array
- ถ้าตัวเลขมีอยู่แล้วให้เพิ่มค่า count ถ้าไม่มีให้เพิ่มตัวเลขใหม่
- แสดงผลลัพธ์โดยแปลง Keys และ Values เป็น array และใช้ for loop
- แสดงผลในรูปแบบ "number: {number} count: {count}"

Test Cases:

1. **Input:** `[1, 2, 3, 2, 1, 3, 1]` **Expected Output:**

```
number: 1 count: 3
number: 2 count: 2
number: 3 count: 2
```

2. **Input:** `[5, 5, 5]` **Expected Output:**

```
number: 5 count: 3
```

3. **Input:** `[0, -1, 2, 0, -1]` **Expected Output:**

```
number: 0 count: 2
number: -1 count: 2
number: 2 count: 1
```

AS03_CheckValidBrackets

วัตถุประสงค์: ตรวจสอบความถูกต้องของวงเล็บใน string โดยใช้ Dictionary และ LinkedList

Method Signature:

```
void AS03_CheckValidBrackets(string input)
```

Logic ที่ต้อง implement:

- ใช้ Dictionary<char, char> เพื่อ map วงเล็บเปิดกับปิด `{('(', ')'), ('[', ']'), ('{', '}')}`
- ใช้ LinkedList เป็น stack เพื่อติดตามวงเล็บเปิด

- วนลูปผ่าน string เพื่อตรวจสอบแต่ละตัวอักษร
- ถ้าเจอวงเล็บเปิดให้เพิ่มเข้า stack (AddLast)
- ถ้าเจอวงเล็บปิดให้ตรวจสอบว่า stack ว่างหรือไม่ และวงเล็บเปิดล่าสุดตรงกันหรือไม่
- แสดงผล "Valid" หรือ "Invalid"

Test Cases:

1. **Input:** `"()"` **Expected Output:** `Valid`
2. **Input:** `"([{}])"` **Expected Output:** `Valid`
3. **Input:** `"(]"` **Expected Output:** `Invalid`
4. **Input:** `"abc(def)ghi"` **Expected Output:** `Valid`
5. **Input:** `""` **Expected Output:** `Valid`

AS04_PrintReverseLinkedList

วัตถุประสงค์: พิมพ์ LinkedList ในลำดับย้อนกลับโดยไม่แก้ไข list เดิม

Method Signature:

```
void AS04_PrintReverseLinkedList(LinkedList<int> list)
```

Logic ที่ต้อง implement:

- ตรวจสอบว่า list เป็น null หรือมีจำนวน 0 elements
- เริ่มจาก Last node และใช้ while loop วนไป Previous
- พิมพ์ค่าของแต่ละ node จนถึง node แรก

Test Cases:

1. **Input:** LinkedList with [1, 2, 3, 4, 5] **Expected Output:**

```
5
4
3
2
1
```

2. **Input:** LinkedList with [42] **Expected Output:** `42`
3. **Input:** Empty LinkedList **Expected Output:** `List is empty`

AS05_FindMiddleElement

วัตถุประสงค์: หา element กลางของ LinkedList โดยใช้ two-pointer technique

Method Signature:

```
void AS05_FindMiddleElement(LinkedList<string> list)
```

Logic ที่ต้อง implement:

- ตรวจสอบว่า list เป็น null หรือมีจำนวน 0 elements
- ใช้ slow และ fast pointers เริ่มจาก First node
- ใน while loop: slow เดิน 1 ก้าว, fast เดิน 2 ก้าว
- เมื่อ fast ถึงจุดสิ้นสุด slow จะอยู่ที่ middle
- พิมพ์ค่าของ middle element

Test Cases:

1. **Input:** `["A", "B", "C"]` **Expected Output:** `B`
2. **Input:** `["A", "B", "C", "D"]` **Expected Output:** `C`
3. **Input:** `["A"]` **Expected Output:** `A`
4. **Input:** Empty LinkedList **Expected Output:** `List is empty`

AS06_MergeDictionaries

วัตถุประสงค์: รวมสอง Dictionary โดยรวมค่าของ keys ที่ซ้ำกัน

Method Signature:

```
void AS06_MergeDictionaries(Dictionary<string, int> dict1, Dictionary<string, int>
dict2)
```

Logic ที่ต้อง implement:

- สร้าง Dictionary ใหม่จาก dict1
- วนลูปผ่าน dict2 และรวมเข้า merged dictionary
- ถ้า key มีอยู่แล้วให้บวกค่า ถ้าไม่มีให้เพิ่ม key ใหม่
- แสดงผล merged dictionary ในรูปแบบ "key: {key}, value: {value}"

Test Cases:

1. **Input:** dict1 = {"apple":3, "banana":2}, dict2 = {"apple":1, "cherry":4} **Expected Output:** ประกอบด้วย "key: apple, value: 4", "key: banana, value: 2", "key: cherry, value: 4"
2. **Input:** dict1 = {"a":1, "b":2}, dict2 = {"c":3, "d":4} **Expected Output:** ประกอบด้วย "key: a, value: 1", "key: b, value: 2", "key: c, value: 3", "key: d, value: 4"

AS07_RemoveDuplicatesFromLinkedList

วัตถุประสงค์: ลบ duplicates ออกจาก LinkedList โดยเก็บเฉพาะตัวแรก

Method Signature:

```
void AS07_RemoveDuplicatesFromLinkedList(LinkedList<int> list)
```

Logic ที่ต้อง implement:

- ตรวจสอบว่า list เป็น null หรือมีจำนวน <= 1 elements
- ใช้ Dictionary<int, bool> เพื่อติดตามค่าที่เห็นแล้ว
- วนลูปผ่าน list และลบ node ที่ซ้ำ
- แสดงผล list หลังการลบ duplicates

Test Cases:

1. **Input:** [1, 2, 2, 3, 3, 3, 4] **Expected Output:**

```
1
2
3
4
```

2. **Input:** [5, 5, 5, 5] **Expected Output:** `5`

3. **Input:** [1, 2, 3, 4] **Expected Output:**

```
1
2
3
4
```

AS08_TopFrequentNumber

วัตถุประสงค์: หาตัวเลขที่ปรากฏบ่อยที่สุดใน array

Method Signature:

```
void AS08_TopFrequentNumber(int[] numbers)
```

Logic ที่ต้อง implement:

- ตรวจสอบว่า array เป็น null หรือ empty
- ใช้ Dictionary เพื่อนับความถี่ของแต่ละตัวเลข
- หาตัวเลขที่มีความถี่สูงสุด
- แสดงผลในรูปแบบ "{number} count: {count}"

Test Cases:

1. **Input:** `[1, 2, 3, 2, 2, 1]` **Expected Output:** `2 count: 3`

2. **Input:** `[5, 5, 5, 5]` **Expected Output:** `5 count: 4`
3. **Input:** `[1, 2, 3, 4]` **Expected Output:** `1 count: 1`
4. **Input:** `[]` **Expected Output:** `Input array is empty`

AS09_PlayerInventory

วัตถุประสงค์: อัปเดต inventory ของผู้เล่นโดยเพิ่มไอเท็ม

Method Signature:

```
void AS09_PlayerInventory(Dictionary<string, int> inventory, string itemName, int quantity)
```

Logic ที่ต้อง implement:

- ตรวจสอบว่า inventory เป็น null หรือไม่
- ถ้า itemName มีอยู่แล้วให้เพิ่มจำนวน ถ้าไม่มีให้เพิ่มไอเท็มใหม่
- แสดงผล inventory ทั้งหมดหลังการอัปเดต

Test Cases:

1. **Input:** inventory = `{"sword":1, "potion":5}`, itemName = `"shield"`, quantity = `2` **Expected Output:** จะต้องมี `sword: 1`, `potion: 5`, `shield: 2`
2. **Input:** inventory = `{"sword":1, "potion":5}`, itemName = `"potion"`, quantity = `3` **Expected Output:** จะต้องมี `sword: 1`, `potion: 8`
3. **Input:** inventory = null **Expected Output:** `Inventory is null`

EX01_GameEventQueue

วัตถุประสงค์: ประมวลผล event ใน queue ของเกม

Method Signature:

```
void EX01_GameEventQueue(LinkedList<GameEvent> eventQueue)
```

Logic ที่ต้อง implement:

- ตรวจสอบ queue ว่างหรือ null
- วนลูปประมวลผลทุก event ใน queue โดยดึงออกจากต้น (RemoveFirst)
- แสดงผล "Processing event: {event.Name}"
- แสดงผล "Remaining events in queue: {count}"
- ประมวลผลตามประเภท event และแสดงผลที่เหมาะสม

Test Case:

- **Input:** Queue with events `"enemy":"Goblin appeared"`, `"powerup":"Health boost found"`, `"level":"Reached level 2"`
- **Expected Output:**

```
Processing event: Goblin appeared
Remaining events in queue: 2
Enemy event processed - Goblin appeared
Processing event: Health boost found
Remaining events in queue: 1
Power-up event processed - Health boost found
Processing event: Reached level 2
Remaining events in queue: 0
Level event processed - Reached level 2
```

EX02_PlayerStatsTracker

วัตถุประสงค์: อัปเดตสถิติของผู้เล่น

Method Signature:

```
void EX02_PlayerStatsTracker(Dictionary<string, int> playerStats, string statName, int value)
```

Logic ที่ต้อง implement:

- ตรวจสอบ playerStats เป็น null หรือไม่
- ถ้า statName มีอยู่แล้วให้เพิ่มค่า ถ้าไม่มีให้เพิ่ม stat ใหม่
- แสดงผล "Updated {statName}: {newValue}"
- แสดงผลสถิติทั้งหมดหลังการอัปเดต

Test Cases:

1. **Input:** playerStats = `{"kills":10, "deaths":2}`, statName = `"assists"`, value = `5` **Expected Output:** จะต้องมี Updated assists: 5, "Current player statistics:", "kills: 10", "deaths: 2", "assists: 5"

   ```
   Updated assists: 5
   Current player statistics:
   kills: 10
   deaths: 2
   assists: 5
   ```

2. **Input:** playerStats = `{"kills":10, "deaths":2}`, statName = `"kills"`, value = `3` **Expected Output:** จะต้องมี "Updated kills: 13", "Current player statistics:", "kills: 13", "deaths: 2"

```
Updated kills: 13
Current player statistics:
kills: 13
deaths: 2
```