Games Development with C++ Basics Part 4

C++ Arrays

- **Definition:** A fixed-size collection of elements of the same data type, accessed using an index that starts from 0.
- **Advantages:**
  - Efficient for random access: You can directly jump to any element in the array using its index.
  - Cache-friendly: Elements are stored contiguously in memory, leading to faster access for sequential operations.
- **Disadvantages:**
  - Fixed-size: Once declared, the size cannot be changed during program execution.
  - Out-of-bounds access errors: If you try to access an element outside the array's bounds (index < 0 or index >= size), it can lead to crashes or unexpected behaviour.

Examples in Game Development (C++ Arrays):

1. **Level Grid:** An array of characters or integers can represent a 2D grid for a platformer or strategy game. The values could indicate different types of terrain (e.g., 'W' for wall, 'E' for empty space).
   C++
   ```
   const int levelWidth = 10;
   const int levelHeight = 5;
   char levelGrid[levelWidth][levelHeight] = {
       {'W', 'E', 'W', 'E', 'W', 'E', 'W', 'E', 'W', 'E'},
       {'E', 'P', 'E', 'X', 'E', ' ', 'E', ' ', 'E', ' '} // 'P' for player, 'X' for enemy
       // ... (other rows of the grid)
   };
   ```

2. **Sprite Animation Frames:** An array of textures or image paths can store different animation frames for a character or object.
   C++
   ```
   const int numFrames = 8;
   std::string animationFrames[numFrames] = {
       "sprite_frame_1.png", "sprite_frame_2.png",
       "sprite_frame_3.png", // ... (other frames)
   };
   ```

3. **Sound Effects:** An array of audio clips or file paths can hold different sound effects for various game events.
   C++
   ```
   #include <SFML/Audio.hpp> // Assuming an audio library

   const int numSounds = 3;
   ```

```
sf::SoundBuffer soundBuffers[numSounds];
// ... (load sound buffers from files)
```

4. **High Scores:** An array of integers can store top high scores for a game.
   C++
   ```
   const int maxScores = 10;
   int highScores[maxScores];
   // ... (load or initialize high scores)
   ```

5. **Predefined Data Tables:** Arrays can hold pre-configured game data, like weapon stats or enemy properties.
   C++
   ```
   struct WeaponData {
       std::string name;
       int damage;
       int range;
   };

   const int numWeapons = 5;
   WeaponData weaponData[numWeapons] = {
       {"Sword", 20, 1},
       {"Bow", 15, 5},
       // ... (other weapons)
   };
   ```

C++ Structures

- **Definition:** A user-defined data type that groups related variables under a single name. Structures help create composite data objects, promoting code organization and readability.
- **Advantages:**
  - Bundles related data: Encapsulates different data types relevant to a single game entity or concept.
  - Improves code clarity: Makes code more readable by giving meaningful names to groups of variables.
- **Disadvantages:**
  - No built-in random access like arrays: You need to access members by name rather than an index.

Examples in Game Development (C++ Structures):

1. **Game Object:** A structure can represent a generic game object with common properties like position, velocity, and health.
   C++
   ```
   struct GameObject {
       float x, y; // Position
       float vx, vy; // Velocity
       int health;
   };
   ```

2. **Character Stats:** A structure can hold detailed character attributes like strength, agility, and

magic power.

```
C++
struct CharacterStats {
    int strength;
    int agility;
    int magicPower;
    int defense;
};
```

3. **Inventory Items:** A structure can describe items in a player's inventory, including name, type, and quantity.

```
C++
struct InventoryItem {
    std::string name;
    std::string type;}; // e.
```

Common g++ commands for compiling, linking, and running C++ code:

1. **Compiling C++ Code**:

To compile a C++ source file (e.g., `example.cpp`) into an object file (e.g., `example.o`):

```
g++ -c example.cpp -o example.o
```

This command compiles the source file `example.cpp` into an object file named `example.o`.

2. **Linking Object Files**:

To link one or more object files (e.g., `example.o`) into an executable (e.g., `example`):

```
g++ example.o -o example

 Executable
 EG: g++ .\basics_parthThree.cpp -o basicThree
```

This command links the object file(s) into an executable named `example`.

3. **Running Executable**:

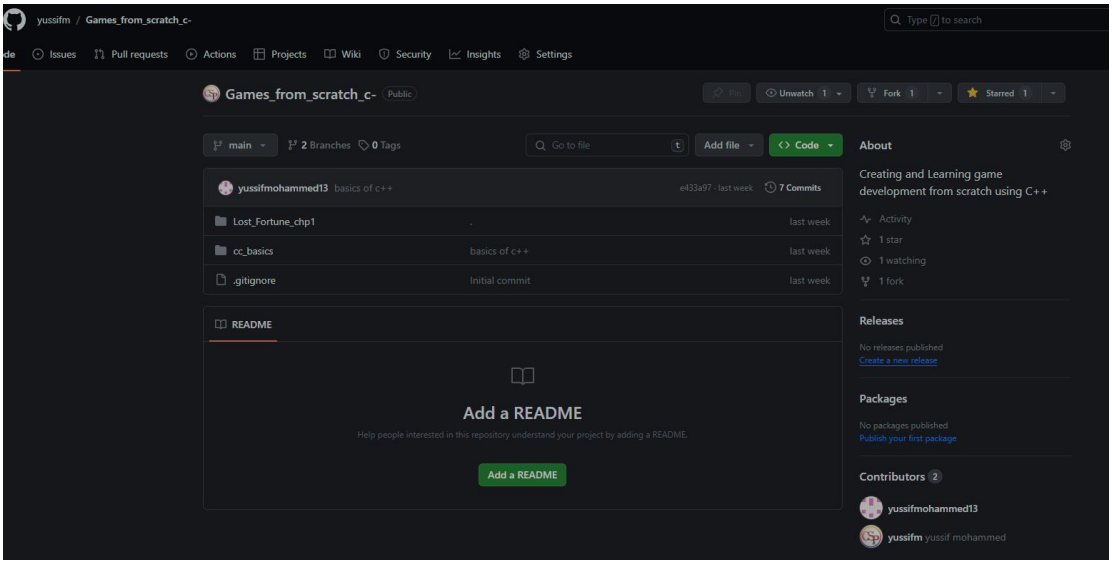
To run the compiled executable:
```
./example
```

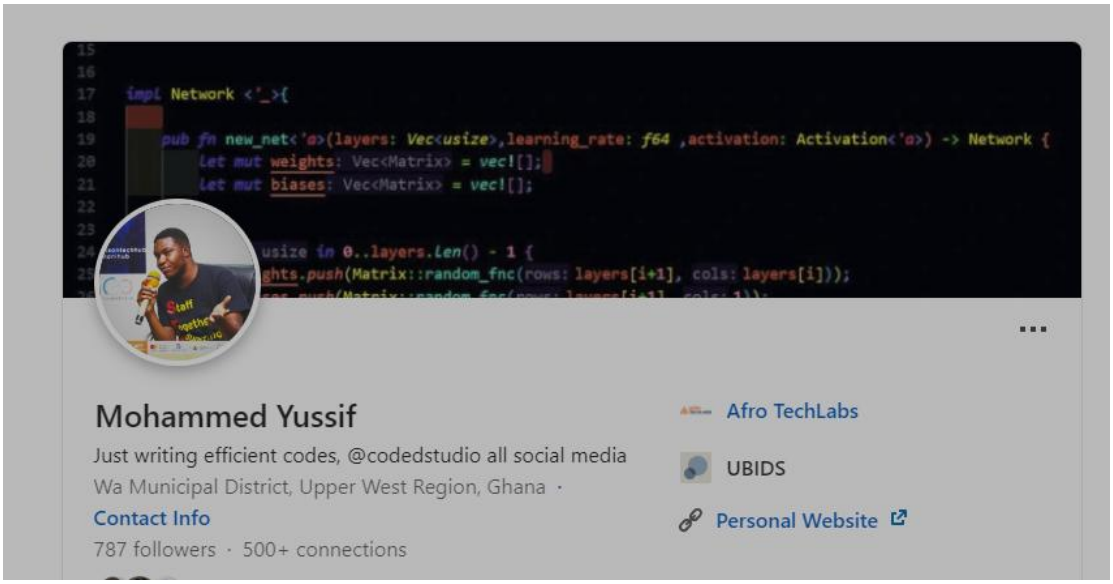
This command executes the compiled executable named `example`.

Here's a summary:

- `***g++***`: This is the GNU Compiler Collection for compiling C++ programs.
- `***-c***`: This option tells `g++` to compile the source file(s) into object file(s) without linking.
- `***-o***`: This option specifies the output file name.
- `***./***`: This notation is used to execute the compiled executable from the current directory.

You can adjust the filenames (`example.cpp`, `example.o`, `example`) according to your actual source code file(s) and desired executable name.




impl Network <'_>{
pub fn new_net<'a>(layers: Vec<usize>,learning_rate: f64 ,activation: Activation<'a>) -> Network {
let mut weights: Vec<Matrix> = vec![];
let mut biases: Vec<Matrix> = vec![];
usize in 0..layers.len() - 1 {
ghts.push(Matrix::random_fnc(rows: layers[i+1], cols: layers[i]));
Mohammed Yussif
Just writing efficient codes, @codedstudio all social media
Wa Municipal District, Upper West Region, Ghana ·
Contact Info
787 followers · 500+ connections
Afro TechLabs
UBIDS
Personal Website