Game dev with C++: Pointers and References

Pointers and References
Pointers and references both are mechanisms used to deal with memory, memory address, and data in a program.
Pointers are used to store the memory address of another variable whereas references are used to create an alias for already existing variable.

Pointers
C++ pointers are a symbolic representation of addresses. They enable programs to simulate call-by=reference, create and manipulate dynamic data structures. Pointers store the address of variables or a memory location.

References
When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting '&' in the declaration.

In game development, understanding pointers and references in C++ is crucial for managing memory efficiently and working with game objects. Here's a breakdown of both concepts and how they apply to games:
Pointers:
Imagine a pointer as a variable that stores a memory address. It's like a note telling you where to find something specific in your game's memory.
Uses in Game Dev:
Dynamic Memory Allocation: You can create objects (like enemies or items) at runtime using new and store their addresses in pointers. This allows you to have a flexible number of objects in your game.

Component-Based Systems: Many game engines use pointers to link components (like physics or rendering) to game objects. This allows for modularity and re-usability.

Data Structures: Pointers are often used internally in data structures like linked lists or trees, which can be useful for managing complex in-game relationships between objects.

Things to Consider with Pointers:

Memory Management: You are responsible for manually deleting objects created with new using delete. Memory leaks can occur if you forget to clean up, impacting performance.

Null Pointers: A pointer can point to nowhere, called a null pointer. It's essential to check for null pointers before dereferencing (accessing the data at the pointed address) to avoid crashes.

Complexity: Using raw pointers can make code harder to understand and debug.

References:
Think of a reference as an alias for an existing variable. It provides another name to access the same data directly.
Uses in Game Dev:
Function Arguments: When you pass a large object (like a player character) to a function, using a reference avoids unnecessary copying, improving performance.

Returning Data: References can be used to return data from functions without creating a copy, again for efficiency.

Benefits of References:
Safer: References automatically refer to existing data, avoiding memory management issues.

Clearer Code: They often make code more readable by showing a direct connection to the original data.

When to Choose Pointers vs. References:
In general, prefer references when you have a clear existing object you want to work with and don't need to modify the pointer itself.

Use pointers for dynamic memory allocation or situations where you need more control over the memory address.

Modern C++ and Smart Pointers:
In modern game development, techniques like smart pointers (like std::unique_ptr or std::shared_ptr) are often used to manage memory automatically. These handle pointer deletion and avoid memory leaks, making code safer and easier to maintain.
By understanding pointers, references, and smart pointers, you'll have a solid foundation for working with memory in your C++ game projects.

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
   ```

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

3. Running Executable:

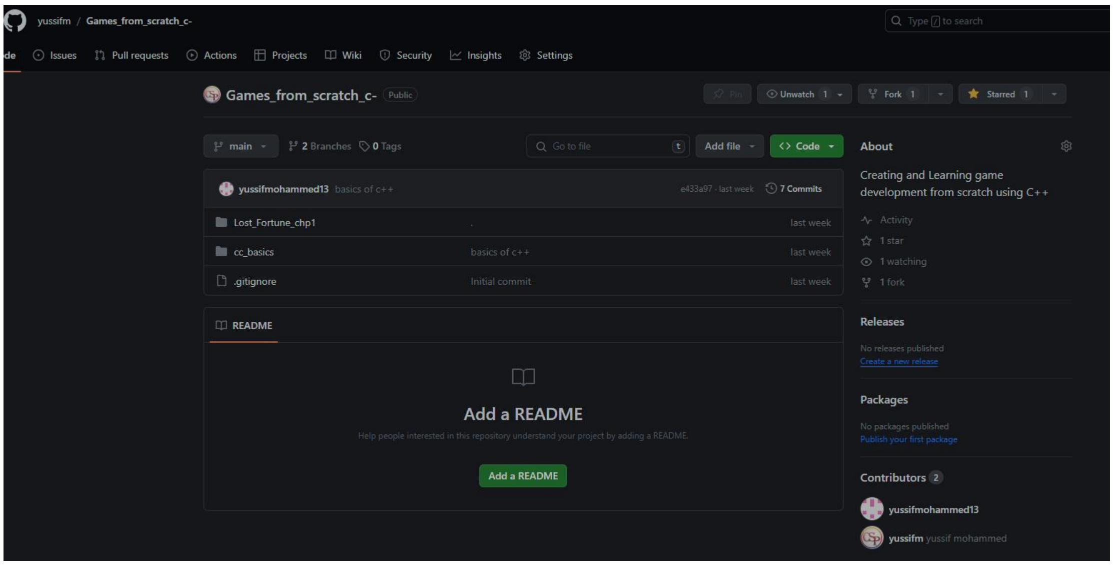
   To run the compiled executable:
   ```
   ./example
   ```

   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.



Games_from_scratch_c- Public

yussifmohammed13 basics of c++ e433a97 · last week 7 Commits

Lost_Fortune_chp1	.	last week
cc_basics	basics of c++	last week
.gitignore	Initial commit	last week

Add a README

Help people interested in this repository understand your project by adding a README.

About

Creating and Learning game development from scratch using C++

Activity
1 star
1 watching
1 fork

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2

yussifmohammed13
yussifm yussif mohammed