Singletons

GAME 311 – Network Programming

Objectives

- Attendance
- Discuss truck simulator Project
- Learn About Singleton Design Pattern
- Walkthrough 1

Singleton

- A Singleton is a design pattern for constructing a class which limits construction to a single instance.
  - This single instance is contained within the class itself
  - Typically this class is then exposed globally.
- Advantages:
  - Globally accessible Instance
  - Ensures only a single instance is ever created.
- Disadvantages:
  - Globally accessible Instance
  - Only initializes on first access.
  - Not thread safe in all situations (safe using VS2015+)

Singleton Example

```
class SingletonExample
{
public:
    static SingletonExample* GetInstance() // this is the main way to access the instance, note it's static
    {
        if (!instance)
        {
            instance = new SingletonExample; // creates a new instance of itself only if it doesn't exist
        }
        return instance; // returns either the established instance or brand new one. only one ever exists
    }
    ~SingletonExample() {}

    void IncrementHealth()  { health++; }
    int GetHealth() { return health; }

private:
    SingletonExample() // note the constructor is private, meaning it controls it's own instantiation
    {
        health = 0;
    }

    int health;
    static SingletonExample* instance;  // note the instance of itself stored privately
};
```

- Top of .cpp file:

```
SingletonExample* SingletonExample::instance = nullptr;
```

Singleton rules

- There are a few rules to designing a singleton class:

1. The constructor be made hidden (private)
2. Static Instance accessor function
   - Doesn't need to be named GetInstance() but that's the standard way
3. A private static instance within the class
4. A global pointer to access the static instance

```
1  class SingletonExample
2  {
3  public:
4      static SingletonExample* GetInstance()
5      {
6          if (!instance)
7          {
8              instance = new SingletonExampl
9          }
10         return instance; // returns either
11     }
12     ~SingletonExample() {}
13
14     void IncrementHealth()  { health++; }
15     int GetHealth() { return health; }
16
17 private:
18     SingletonExample() // note the constru
19     {
20         health = 0;
21     }
22
23     int health;
24     static SingletonExample* instance;  //
25 };
```

- Top of .cpp file:

```
SingletonExample* SingletonExample::instance = nullptr;
```

Singleton Example

- Accessing in other cpp files you simply need to use the classname followed by ::GetInstance().
- This will return the single instance of the class all areas of code have access to, the SINGLETON.

```
1   #include <iostream>
2   // Singelton Use
3   #include "SingletonExample.h"
4
5   using namespace std;
6
7   int main()
8   {
9       cout << "health:" <<  SingletonExample::GetInstance()->GetHealth() << endl;
10      SingletonExample::GetInstance()->IncrementHealth();
11      cout << "health:" << SingletonExample::GetInstance()->GetHealth() << endl;
12
13      system("Pause");
14  }
15
```

Singletons in Games

- Singletons are sometimes used to manage systems of games and game engines which need to be frequently used throughout many other sections of code.
- Often used in:
  - AudioManagers
  - InputManagers
  - Debuggers/ loggers
- Some architectures restrict to only a single Main or Game singleton which contains a single static instance of all other classes that a game may need accessible.
- Other Architectures prefer as restrictive of a system as possible to eliminate accidental use and abuse of systems.