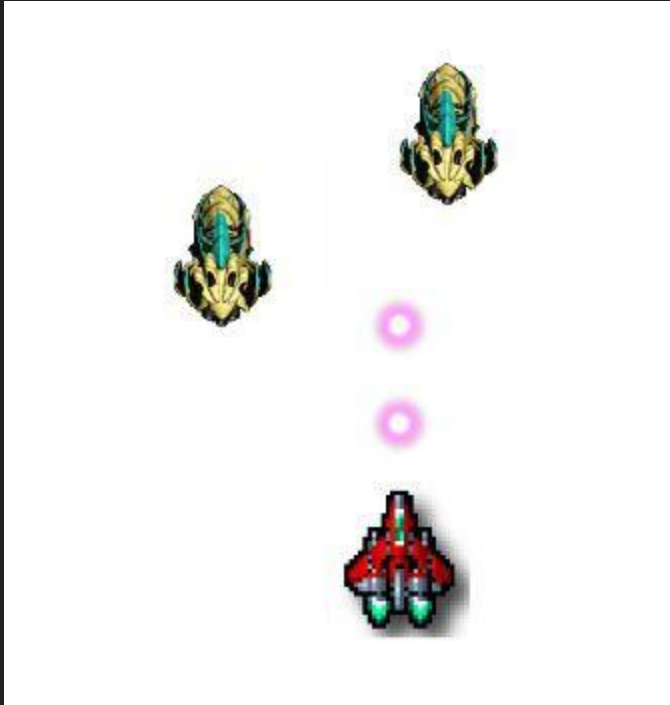


VGP230

Lesson 2 (HelloSpaceShooter)

Instructor: Darren Waine

Lesson Overview



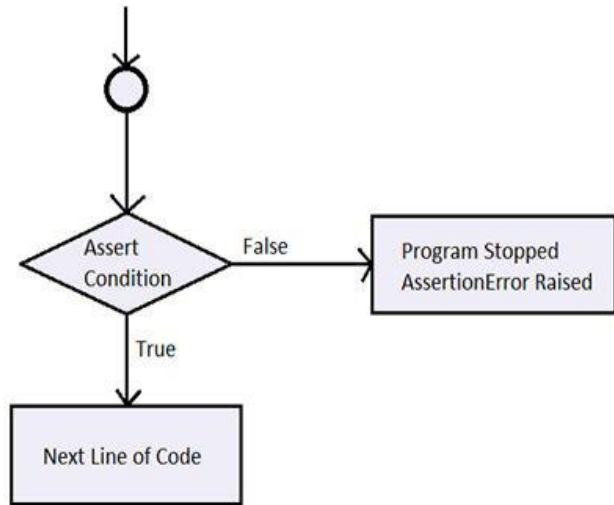
- Class structure in X
- Asserts
- Virtual/Abstract refresh
- Object Pooling
- In-class Project:
HelloSpaceShooter

Class Structure in X

- In general, we will follow this structure in the classes we create within X:
 - Load() - texture initialization, settings startup values
 - Unload() - cleaning up used memory if necessary
 - Update(float deltaTime) - physics, input, game logic
 - Render() - rendering sprites, textures, lines, etc
- This is a very similar concept to the Start() -> Run() -> Stop() structure that X uses internally!


Asserts

- An assertion is a statement in various programming languages that enables you to test assumptions about the program you're writing.
- Asserts usually contain a bool to be verified as true and a message to be logged/printed out if that bool is not true. This helps detect errors/bugs quickly with clear reasoning as to why.



Asserts - Continued

- How is this useful within X Engine?
 - Asserts can verify code setup is done properly (ie, textures being loaded)
 - They can also be used to verify certain expectations at run time, such as the proper values being passed in as an index for accessing an array, or that the pointers used are not null.
 - Normally, asserts should not be used for art assets like the example below, but it might help in your debugging.

```
 void Enemy::Load()  
{  
    mTextureId = X::LoadTexture("enemy.png");  
    XASSERT(mTextureId != 0, "Texture not loaded!");  
}
```

Virtual Function / Abstract Class refresh

- Member functions in a class that have the potential to be overridden by any derived (child) classes.
 - Each derived (child) class can implement their own version of that function that can be called at runtime by the compiler through a pointer to the base class.
- Abstract classes:
 - Classes can be marked as abstract by including one pure virtual function.
 - Syntax in the header: `virtual Load() = 0;`
 - Why?
 - Abstract classes cannot be instantiated
 - In order for any derived (child) class to not also be abstract, a body for any pure virtual method must be implemented.

Object Pooling - Introduction

- Imagine a common game scenario of shooting bullets. How is it done?
 - In C++, we would call “Bullet bullet = new Bullet();” each time we want to make a new bullet
 - As time goes on, depending on how many players and enemies are shooting bullets, this would be a very high amount of allocations.
 - ... Is that the best way?
- A better solution is to allocate all the memory needed for a “pool” of bullets ahead of time!
- The main goal:
 - To improve performance and memory use by reusing objects from a fixed pool instead of allocating and freeing each object individually.

Object Pooling - What is it?

- Instead of actually creating new game objects each time we need a new one (calling new to construct the object and allocate it in memory), we can utilize an “active” state on a set of pre-existing objects.
- When we want to “create” a new object, we can iterate through the pre-allocated list of objects until an inactive one is found. This can then be used as the “new” object, therefore making it active.

Object Pooling - Why do it?

- It comes with many benefits related to speed and memory.
- It helps prevent memory leaks and memory fragmentation.
- Memory leak:
 - Occurs when you do not deallocate the memory you've previously allocated.
 - It's called a "leak" because the available memory left over for the rest of the project has decreased! Imagine memory as a bucket of water; a memory leak happens when you spill some of that water.
- Memory fragmentation:
 - Occurs when you allocate bits of memory in many different locations, leaving gaps of available memory that may be too small to allocate anything in.
 - Object pooling helps prevent this, as we set a reusable block of memory ahead of time instead of making many small allocations/deallocations.

Sources (and useful links!)

- <https://www.codeproject.com/Articles/5166096/Robust-Cplusplus-Object-Pools>
- <https://gameprogrammingpatterns.com/object-pool.html>