

## Engine Description

# Main

# Graphics

# Physics

# Artificial Intelligence

# User Interface

# Audio

Professor Christopher Howard

August Session 2015

Table of Contents

Contents

[Engine Description 1](#_Toc433544126)

[Main 1](#_Toc433544127)

[Graphics 1](#_Toc433544128)

[Physics 1](#_Toc433544129)

[Artificial Intelligence 1](#_Toc433544130)

[User Interface 1](#_Toc433544131)

[Audio 1](#_Toc433544132)

[Overview 4](#_Toc433544133)

[Main 4](#_Toc433544134)

[Graphics 4](#_Toc433544135)

[Physics 4](#_Toc433544136)

[Artificial Intelligence 5](#_Toc433544137)

[User Interface 5](#_Toc433544138)

[Audio 5](#_Toc433544139)

[Engine Features 6](#_Toc433544140)

[Main 6](#_Toc433544141)

[Graphics 6](#_Toc433544142)

[Physics 6](#_Toc433544143)

[Artificial Intelligence 7](#_Toc433544144)

[User Interface 7](#_Toc433544145)

[Audio 7](#_Toc433544146)

[Core Descriptions 7](#_Toc433544147)

[Main 7](#_Toc433544148)

[Graphics 8](#_Toc433544149)

[Physics 8](#_Toc433544150)

[Artificial Intelligence 9](#_Toc433544151)

[User Interface 10](#_Toc433544152)

[Audio 10](#_Toc433544153)

[Feature Descriptions 10](#_Toc433544154)

[Main 10](#_Toc433544155)

[Graphics 11](#_Toc433544156)

[Physics 12](#_Toc433544157)

[Artificial Intelligence 13](#_Toc433544158)

[User Interface 14](#_Toc433544159)

[Audio 14](#_Toc433544160)

[Future Extensions 14](#_Toc433544161)

[Main 14](#_Toc433544162)

[Graphics 15](#_Toc433544163)

[Physics 15](#_Toc433544164)

[Artificial Intelligence 15](#_Toc433544165)

[User Interface 15](#_Toc433544166)

[Audio 15](#_Toc433544167)

[Issues and Solutions 16](#_Toc433544168)

[Main 16](#_Toc433544169)

[Graphics 17](#_Toc433544170)

[Physics 17](#_Toc433544171)

[Artificial Intelligence 18](#_Toc433544172)

[User Interface 18](#_Toc433544173)

[Audio 18](#_Toc433544174)

# Overview

## Main

The Main Core is the backbone and framework of the engine. It is designed to allow all other cores interface seamlessly with one another, and provide optimal support for the developer who chooses to build a game using it. It houses the window creation method, and general game loop that allows the game to function. The Main Core offers easy manipulation of the other cores through its functions.

It also contains the definition for the base game object, the ABC class, from which is derived other game objects such as the Player, Enemies, Pickups, and Projectiles. It contains managers that take care of each of these.

## Graphics

The Graphics core is responsible for all the rendering done in the demo. This includes the models, sprites, and textures used by any other core that needs to draw something to the screen. At its core is the usage of DirectX 9.0c to achieve these goals. It provides a simple interface to be able to use it through the GFX.h file.

## Physics

This demonstration shows off the features of our class designed, coded and integrated during this course. The following teams …

Physics Core which consisted of:

Darrel Hidle Team – Team Lead

Bennett Lynn and Vardges Gharibyan

## Artificial Intelligence

The AI core is a class that will control the behaviors of the non-player controlled enemies and missiles in the game. It will do this by directly accessing each enemy and missile that is currently active in the game world each frame and then updating their location and actions based on the programmed behavior. This will include the enemy’s movement patterns, fire patterns, and missile pathfinding.

## User Interface

User Interface handles all the things that the player uses to interact with the game, as well as the things the game uses to interact with the player.

## Audio

Throughout this class, our team has worked on an audio core that was integrated in to a game engine in the last two weeks. Although we had some issues along the way, we were able to achieve our goal of integration and left the audio core in an upgradable and reusa

ble working state within the engine. This paper will briefly explain the game engine, audio core, future extensions, and some of the issues and solutions we experienced along the way.

# Engine Features

## Main

The features of the Main Core are

* Window creation through Windows protocol
* Seamless integration of Graphics, Input, Physics, and Artificial Intelligence cores
* Main game loop
* Managers for game objects such as Player, Enemies, Asteroids, Pickups, and Projectiles.

## Graphics

The features for graphics include the following:

* Able to create and destroy a window.
* Can render X file format 3D models with a texture applied to the models.
* Can create fonts and draw them on screen where needed.
* Ability to load sprites and textures and drawing them on screen.

## Physics

Physics Core Features:

Independent of other cores

Provides a wrapper for box2d library and simplifies it further

Native access to Box2d in addition to wrapper functions.

Customizable stepped update design (inherited from box2d)

2d Physics collision support.

Continuous collision detection

Custom collision shapes as well as a box primitive

Collision callback function to intercept collision events

Physics object creation and management by way a base physics object and physics object manager.

Configurable physics parameters such as gravity, bounce and friction

## Artificial Intelligence

* Unique behaviors for each enemy type and the boss.
* Unique behaviors for each missile type both player and enemy.
* Direct manipulation of enemy and missile data, no need for accessor functions.

## User Interface

## UI contributes to the engine by providing the menu system and the change in game states.

## Audio

Within the engine, the audio core is triggered by events. As the player presses buttons, enemy fires, asteroids explode, and ships explode, these events will trigger specific sound effects that are called from the audio core. Another feature within the engine are that each core was created independently, which allows for code reusability.

# Core Descriptions

## Main

The Main Core was designed around Windows API and DirectX API. The Windows API handles window creation and system messages through the uses of WINAPI and PeekMessage, respectively. The DrirectX API handles graphics, meshes, input, and calculations through the use of the various functions that support those features. The Main Core primarily uses Windows API as it is not responsible for the calculations or functions that DirectX is used for.

## Graphics

The graphics core has been designed to take a 3D model and place it within a 2D area. Creating a 2.5D graphics engine that allows us to make our game. It is also able to take a texture and place it on both the model and the background area to give the game its look. The graphics core is able to render not just the models and textures but also is designed to render several other useful things. This graphics core is also what allows us to render fonts. Which is what allows us to put in text for our menus and in game text if needed. It also allows us to load in sprites as well as the 3d models for more in game variety. And to help our game look the best it can the graphics core is also able to render and use shaders.

## Physics

Physics core is designed to give provide physics support by creating an instance of Physics Core and creating physics objects and passing them into the physics world by the physics object manager interface. The physics object manager interface tracks the physics objects and provides a way to query their positions, velocity and other information that is simulated in the physics world. In addition the physics object manager can track user custom objects by including them in the physics object creation and it will pass them to the collision handler to simplify collision handling.

## Artificial Intelligence

The AI core will feature a simplistic but efficient behavior. The AI core will include a unique behavior for each enemy type in the game as well as a unique behavior for the boss of the game. The core will, on each frame, read a list that will describe the current game state by giving each enemy type that is active in the game at that particular moment. The core will then pass each enemy into one of its functions and determine its movement. The data of each enemy will be directly manipulated by the AI core as each enemy will be a child of a friend class of the AI core class. This update function will have distinct behaviors for watch enemy type and when the enemy object is passed into this function the type will be read and the appropriate behavior will be used.

The AI core will also control the movements of the missiles in the game as well. These missiles will be of 2 different types, an enemy and a player type. Each one of these missiles will have their own type of behavior ad will be modified in a similar way that the enemies are. They will be read from the list of current game objects and passed to an update function where their data will be modified directly.

## User Interface

## UI contributes to the engine by providing the menu system and the change in game states.

## Audio

The audio core utilizes FMOD as a sound bank and file wrapper. The core utilizes DirectX to create the FMOD wrapper as “FR”. The sound is triggered during game play and is played through the end users audio system through a class function.

# Feature Descriptions

## Main

Windows protocol is done through the use of Windows API variables and functions like HINSTANCE for the window handle, and WINAPI for the window creation.

The cores are seamlessly integrated through the use of interfaces. With these interfaces, the cores are interchangeable, so long as the new functions are properly defined in the interface. The interface functions are called within Main Core at the right times, so the developer that uses it will not need to worry about updating those references, only the reference the interface uses to run the specific core.

The main game loop is the only way a game runs, and it is the Main Core’s job to account for it. By using the interfaces, the main game loop is capable of accessing each core and running the proper functions.

The managers allow access to the various types of game objects including functions to interact with the objects and to make them interact with one another.

## Graphics

The graphics core has several features that we can use to make a great looking game. The graphics core is able to load DirectX X file format models using the model class which can be made in programs such as 3DSmax. It also is able to optimize the model if that is at all possible. The model class also allows the user to set the position, scale and rotation of each of the models. Both the model, sprite and texture classes load the textures and textures placed on the model internally. The texture class is what allows the graphics core to load textures by filename. It also is able to set the initial size of the storage data structures used to store a texture. It also allows for the user to access the dimensions of the texture. The sprite class is what allows the user to use sprites in game. This class uses the texture class to load the sprites. It also allows the user access to the sprite data by using the captured IDs. The camera class is what can be used to set the player camera. It creates the orthographic projection by giving the width, height and +/- Z clipping values. It also creates the perspective projection using the FOV and aspect ratios. This also uses the +/- Z clippling values. The shader class is what allows the graphics core to load and use shaders. Using the shader class the user can change parameters to change what a shader does. The user is able to load several shaders with different variables such as diffuse, ambient and specular. The graphics core also has the font class allowing in game text. Using the font class the user may set the size of the font as well as the type of font they want. It is also able to render the text in Italics or in bold.

## Physics

**Independent:**

Physics Core is independent of the other cores. It does not care what graphics rending method is in use. It can handle custom data objects by way of void pointers.

**Collision callback:**

Physics core allows the calling program to create a custom call back for collisions. When objects collide in the physics world the offending objects are passed by reference into the method registered by calling program. This allows the collision method to make decisions by accessing the methods that the calling program provided such as a string that indicates the name or in the case of our demo the objects are decedents of physics object so the game can check parameters such as name or velocity to decide what to do with the object. If the object needs to be destroyed it could call the markForDeath method of the object for example.

## Artificial Intelligence

* Each enemy type, including the boss, will each have a unique movement pattern. This will both help the player to determine what type of enemy it is. This will also serve as a way to increase the difficulty of the game and reward the player for meeting these challenges. The tougher the pattern of the enemy the more points they will reward to the player. The boss will also feature a combination of these movements to test what the player has learned. This will also reward the player for learning the patterns of each enemy and attack based on what they think the pattern is.
* Both the player and the enemy missiles must be separated as far as behavior is concerned due to the fact that it should be easier for the player to destroy enemies but not the other way around. Therefore the player missile will have “better” behavior that the enemy missile. All this means is that the player missiles behavior will always track the enemy while the enemy missile will give the player a chance to avoid it.
* The AI core will be a class that will be a friend of the base class for both the enemies and the missiles of the game. This means that there will be no need for any accessor functions, the data of the enemies and missiles can be directly edited by the AI class. This eliminates the need for excess functions that pass data back and forth. The less amount of instances where data is transferred the less chance of there being any type of bug or error.

## User Interface

UI relies heavily on the graphics core to do the actual drawing, accomplishing this by loading the sprites to be drawn into a list in GFX.h and then telling graphics which sprites should currently be drawn.

## Audio

The Audio Engine allows all sound development and database to be modular to the rest of the system by dividing it into two classes. It used as the interface for gathering the .wav files and using the audio class, which handle all the heavy processing to make a simple pointer function. In addition, all failovers will be taken care off before the sound is produced, to reduce pauses or latency during the application running for load balance reasons. The interface is controlled by a namespace, in order to centralize all the capabilities and keep dependencies from other classes.

# Future Extensions

## Main

Possible extensions of main include

* A debugger to help the developer debug engine-specific issues.
* A more user-friendly interface, similar to Unreal Engine and Unity.
* Extending the ABC class to create more types of game objects.

## Graphics

Some extensions for graphics would be:

* Addition of more shaders.
* Using DirectX vertex and index buffers directly.
* Ability to change the resolution and size of the window.
* Able to set the framerate that is desired by the user.
* Capability to change DirectX settings like multi-sampling and anisotropy.

## Physics

The creation of a visual editor for creating collision shapes and placing objects.

Create more wrapper methods.

## Artificial Intelligence

## User Interface

## We’d like to be able implement a health bar that blinks when the health drops below a certain point. More work could also be done to make the menus prettier.

## Audio

Future extensions will envelop the input from the main class, in order to collaborate an easy volume control interface. Other things will include video streaming that should handle well with its own failover in order for no invitation to acknowledgement failure when the pointer looks for the file. Also, with the simple integration of .wav files, adding more sounds to the database would make the code more reusable.

# Issues and Solutions

## Main

The biggest issue for Main was coordinating with Graphics to develop the window creation. WINAPP requires a graphics device to allow the window to be updated, but since Graphics would handle that, there was a problem with getting it so Graphics could access and use it. In the end this was solved by Main calling Graphics functions for things like window creation and device resetting.

Another issue for Main was creating the game demo files so that the other cores could access them and update them. The Engine Lead, Sam, suggested we use lists as they enable quick adding and removing of elements. Once Main got through that hurdle, we coordinated with other cores so they could access the lists of enemies and asteroids and properly update them.

There was also some confusion with physics, in that they had to use a specific type of object because of the third party physics code they used, so in the end we had to move some of the functionality that was in ABC, specifically position and velocity, to the physics core. Then we instead added a physics object to ABC and used that to keep track of its position and velocity.

## Graphics

There were various issues that the graphics core encountered. The first one was due to communication, or lack of it. The other was members of the core would produce code that was incompatible with the rest of the code. As for the code, an issue was found in the rendering list in that the background image would be drawn over the objects in the scene. This would be fixed by rearranging the way the background image is drawn.

## Physics

Box2d objects must live for the life of the game. This presents a challenge to object management. In this game for example there is a need to delete missiles and bullets when they are done. The solution as recommend by the box2d library was to create a Boolean variable to mark the object for deletion and call the remove method in the object manager instead of deleting it on the fly where it could be in the middle of a collision event.

Another interesting design issue is was limiting the player to the screen during the creation of the code before physics was integrated the player’s position was directly manipulated but with the addition of physics it is best practice to do all movement by applying forces to the player. When the player was manipulated by physics and directly the player would get stuck in the corners by applying forces to the ship opposite of the screen boundaries the problem turned into a very nice game play feature.

## Artificial Intelligence

There were a lot of communication issues due to the circumstances and medium we are using but the solution was just keeping in touch as much as we can and trying to stay on top of everything.

## User Interface

## The main issues we ran into were communication based. Aaron and Bill had a hard time understanding Chris’s instructions and, as a result, would submit code that didn’t take advantage of the GFX interface or wouldn’t work with the rest of the code at all.

## Audio

Issues included a memory leak from the audio.cpp to the audio\_interface.cpp, which was resolved from a constant loop that was created by improper library placements. Other issues included video error capture and input interface functions. The direct error check for the video was not implemented, which allowed the video to play but sometimes caused overlapping with other application functions. The input interface issue was never solved, since the error in the debug window never appeared. The input keys would get locked without a break. My guess is that the limiters for the incrementing volume would (by default) be set to a volume level that is never hit unless specified.