JustFormed

Daniel Anglin – Project Officer
danaidh@gmail.com
James Leonis – Technology Officer
virus2566@gmail.com
Alex Rodriguez – Interface Officer
phantomhawk88@yahoo.com
Timothy Turcich – Gameplay Officer
poppyspy@hotmail.com

Production Contract

We, the members of JustFormed as named herein, do agree to deliver a product that meets the standards set forth by our Producers, and following the technological specifications and outline found in this document.

We reserve the right to change those features this document describes at the discretion of the group after having consulted and meeting the approval of the project's Executive Producers.

Daniel Anglin

James Leonis

Alex Rodriguez

Timothy Turcich

Executive Producer

Table of Contents

Introduction

The Game

Technical Aspects

Synopsis

ZFighter is a 3D strategic combat flight simulator with several twists. The main objective for the player is to search and destroy enemy hunters. The number of enemies that you fight will depend on the difficulty setting and the stage. Each stage is cleared by defeating all of the enemies in the stage.

The game is played from a first person cockpit view with a third person minimap detailing the entire playable area. The player can toggle between either a split-screen view of the cock pit and map in seperate viewports and a 3rd person view in the other or a fullscreen view of the cockpit and the map as a small minimap. The player is able to use lasers and mines to destroy the enemy.

The combat arena is significantly different from other combat flight simulators because it takes place inside a viscous material that hampers the detection of anything but large objects. The player and enemy hunters must burrow through this material in order to find and destroy each other. By tunneling, they leave behind a tunnel trail. These tunnels will be instrumental to locating the enemy.

Due to the viscous world hampering detection, the tunnels are the key in discovering the location of the enemy. Following the tunnels both allows the player and enemies to find their opponents and gives them a significant speed boost. To counter this, the player can drop mines, execute confusing tunneling maneuvers, and utilize power-ups to gain a temporary edge over the computer player.

Power-ups allow the player to gain a temporary edge. Each is activated at the time of the player's choosing. These power-ups include the Sonar Ping, which can temporarily reveal the location of all objects in the world, and the homing torpedo, which will follow a tunnel and blow up when it encounters another object or the end of the tunnel.

The world is be populated by large detectable static objects. These obstacles require the player to adjust their strategy to take them into account otherwise they will take damage or be destroyed. ZFighter features three static maps and a randomly generated map option. Careful planning of tunnel and mine placement will be the cornerstone of a winning strategy.

Story

ZFighter is the premier combat simulation system for the Space Fleet as well as many non-military commercial groups desiring realistic tactical space flight training in an environment more suitable for learning. Its current configurations allow for skills testing with a variety of input mechanics to allow pilots to become accustomed to flying nimble, high speed craft that typically have the classic control stick and throttle systems as well as the slower, less maneuverable juggernauts that are used for hauling or defense systems that generally utilize more comfortable keyed inputs due to their availability of space.

ZFighter pits the user against a variety of opponents, environments and battle situations to push them to expand their limits and skills further and further. Do not take this system lightly, believing it to be merely a simple flying simulation, this is as real as it gets soldier! You will be faced with enemies that are tougher and faster than you. You will be riding the razor edge every minute you live within that chair. And if you want to survive to see another day, you better hope you have what it takes to bring those buggers down!

What? Are you still here? Get to work soldier! You have enemies to burn!

Gameplay Narrative

The player starts off in a training exercise and must scan the area and destroy all hostiles. By firing off sensors the player's radar can detect helpful power ups and see hostiles that cross the sensors path while it is active. Once an enemy is located and the players ship approaches near a light trail appears that the enemy ships leaves behind. The player can only see this light trail for a short distance and must follow it to get to the enemy. The enemy however is also fighting against you and will be dropping mine traps along this trail to destroy you. The player also has this light trail and if the enemy finds it, will pursue along it to destroy you with lasers and seeking missiles. The player and enemy have similar weapons. The Seeking missiles do not seek out the ship but seek out the trails behind the ship and follow along them until they reach the trails creator. Once the missile gets tot the trail it splits and goes in both directions of the trail. The missile and mines are quite powerful and it doesn't take many of them to destroy your target or be destroyed by them. They however are a limited supply depending on what type of ship the player and the enemy are using and in the case of running out the player and enemy will only have their laser weapons left to destroy each other with unless they pick up an ammo power up which would restore some of ships payload. The other power ups available are armor increase which repairs damaged armor and a Energy sensor pickup which will illuminate and allow you to see where all enemies are for a short period of time.

After the training exercise which is the first level of the game the same scenario will be played out in the 2nd and 3rd levels except now obstacles such as asteroids appears around the level area which is a large cube in shape. The first 3 levels represent the 3 different looks that levels will take on, but as you go beyond those levels the look will recycle while more obstacles and enemies are placed in for challenge.

Entity Stat Descriptions

Vehicle Stats

Mass	Vehicle mass (in kg)
Max Velocity	Maximum velocity threshold (to prevent vehicle damage)
Max Engine Force	Maximum thruster output (in N)
Max Armor	Total damage allowable before the ship is destroyed
Sensor Cool down	Minimum time between sensor use (in seconds)
Laser Cool down	Minimum time between laser shots (in seconds)
Missile and Mine Cool down	Minimum time between special weapons use (in seconds)
Laser Damage	Damage (to armor) caused by laser shots
Maximum Mines	Maximum cargo for space mines
Maximum Missiles	Maximum cargo for homing missiles

Equipment Stats

Mass	Equipment mass (in kg)
Max Velocity	Maximum velocity threshold (to prevent equipment destruction)
Engine Force	Maximum thruster output. (in N)
Damage	Damage (to armor) caused by weapon
Sensing Radius	Radius of sensor detection capabilities (in meters)

Player Ships

Interceptor
The Interceptor is the smallest and lightest weight spacecraft that the player gets to use in the game. This small ship has a limited weapons payload and its weapons are less damaging. The ship's speed and maneuverability by far makes up for its lack of weaponry.

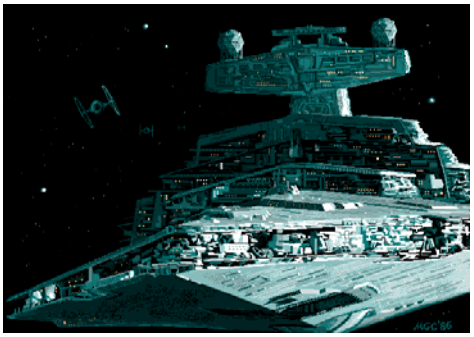
Mass	1000kg
Max Velocity	300m/s
Max Engine Force	50N
Max Armor	50 Armor
Sensor Cool down	30 seconds
Laser Cool down	0.3 seconds
Missile and Mine Cool down	2 seconds
Laser Damage	-17 Armor
Maximum Mines	5
Maximum Missiles	2

Cruiser
The Cruiser is average in respect to speed and weight in the player arsenal. It has a respectable payload of weapons and it has a more powerful laser system then smaller ships like the interceptor.

Mass	3000kg
Max Velocity	200m/s
Max Engine Force	100N
Max Armor	100 Armor
Sensor Cool down	30 seconds
Laser Cool down	0.3 seconds
Missile and Mine Cool down	2 seconds
Laser Damage	-35 Armor
Maximum Mines	10
Maximum Missiles	5

Hulk
The Hulk is a battleship and while lacking speed and maneuverability has are large arsenal and heavy armor.

Mass	10000kg
Max Velocity	100m/s
Max Engine Force	200N
Max Armor	300 Armor
Sensor Cool down	30 seconds
Laser Cool down	0.3 seconds
Missile and Mine Cool down	2 seconds
Laser Damage	-50 Armor
Maximum Mines	20
Maximum Missiles	20



Enemy Ships

Fish

The Fish is the smallest and lightest weight spacecraft that the Enemy gets to use in the game. This ship swims and jukes through the darkness. It has a very limited payload of weapons, but in turn is the fastest ship in the game.

Mass	800kg
Max Velocity	350m/s
Max Engine Force	60N
Max Armor	50 Armor
Sensor Cool down	30 seconds
Laser Cool down	0.2 seconds
Missile and Mine Cool down	2 seconds
Laser Damage	-13 Armor
Maximum Mines	3
Maximum Missiles	1

Monkey

The monkey is the enemy's mid-average type ship. It's very similar to the player's Cruiser, except it specializes in mine laying so it has a larger payload of them.

Mass	3000kg
Max Velocity	200m/s
Max Engine Force	100N
Max Armor	100 Armor
Sensor Cool down	30 seconds
Laser Cool down	0.3 seconds
Missile and Mine Cool down	2 seconds
Laser Damage	-35 Armor
Maximum Mines	20
Maximum Missiles	0

Elephant

The Elephant is a floating fortress and has the largest payload of weapons.

Mass	9000kg
Max Velocity	100m/s
Max Engine Force	125N
Max Armor	350 Armor
Sensor Cool down	30 seconds
Laser Cool down	0.3 seconds
Missile and Mine Cool down	2 seconds
Laser Damage	-50 Armor
Maximum Mines	30
Maximum Missiles	30

Enemy AI Behavior

Enemies in ZFighter will follow a simple pattern of behavior. There are three possible behavior modes that any enemy can be in. Each enemy ship has a preferred state based on its vehicle type.

Wandering State: The basic behavior mode of all enemies. This mode is in effect when the enemy has not yet found a player's trail and the player has not tripped a sensor of the enemy's.

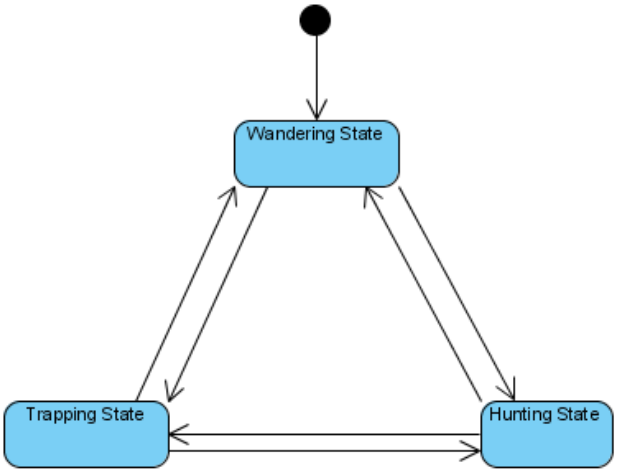
Enemies will begin in this mode.
Enemies enter Wandering state if enemy loses player trail while in Hunting state.
Enemies attempt enter Wandering state if player has not tripped an enemy sensor over one minute after enemy enters Trapping state.

Hunting State: The aggressive mode of enemy ships. This mode is in effect when an enemy has found a player ship trail and begins following it in an attempt to destroy the player.

Enemies enter Hunting state when player trail is found.

Trapping State: The defensive mode of enemy ships. This mode is in effect when an enemy's sensor registers contact with a player or the player attacks the enemy. The enemy will attempt to trap the player by laying space mines in its path.

Enemies enter Trapping state when a sensor registers contact with player.



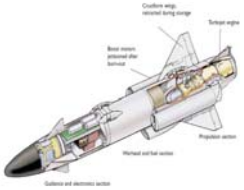
AI Behavior Preferences
(% chance to remain in state)

	Wandering	Hunting	Trapping
Fish	20%	70%	10%
Monkey	20%	40%	40%
Elephant	20%	10%	70%

Preference % will affect attempts to change states based on time.

Weapons

Laser
The primary weapons of all ships are high powered laser cannons mounted to the front of the vessel. These weapons vary in power based on the type of ship in use and the stats can be seen on the vehicle statistic pages.



Tracking Missile
Tracking missiles if in range of an enemy ship's light trail will follow the trail and collide with anything along that path. Traps left along these trails can deter the ship itself from being the target of the explosion.

Mass	50kg
Max Velocity	400m/s
Engine Force	25N
Damage	-75% of Total Armor

Mine Trap
Ships can drop mines along their trails that act as traps for other ships traveling along that trail. Lasers and missiles can destroy mines.

Mass	50kg
Damage	-100% of Total Armor

Sensor
Sensors can be destroyed to counter the enemy placing them.

Sensing Radius	30000m

Power Ups

Homing Missile
Bonus: +1 Homing Missile
Description: Give the player an auto targeting missile weapon that seeks the enemy by following their light trail.

Laser Damage Increase
Bonus: + 25% to Ship laser damage.
Description: Depending on what Ship gets this pickup the laser the ship can fire will increase its damage dealt to other ships by 25%
T*his is not an actual powerup, the cake is a lie!*

Care Package
Bonus: +25 to Ship Armor (the ships armor factor cannot go above its maximum value)
Description: The care package is a repair kit for damaged ships in battle.

Mine Ammo
Bonus: +2 Mines (the mine payload of a ship cannot go above its maximum carrying capacity)
Description: Mine ammo pickups giving more mines to a ship allow the ship to defend it self longer.

Energy Detector
Bonus: This allows all ship trails in the area to become visible on radar. (Lasts 10 seconds)
Description: The energy detector detects all energy signatures in the level for 10 seconds. Displaying them to the ships radar and showing ship trails.

Levels

Level 1: Planetary Orbit

This battle takes place in a near a small planet. This is a training exercise where the player must find a hostile in the area and destroy them.

Enemies	1
Obstacles (space debris)	No
Heavenly Bodies (planets / space stations)	No

Level 2: Asteroid Field

This battle takes place near a small asteroid field. Several hostiles are in the region and must be destroyed.

Enemies	3
Obstacles (space debris)	Yes
Heavenly Bodies (planets / space stations)	No

Level 3: Space Station Alpha

Following a distress call to its source near an old space station you are set upon by several hostiles. Space station and nearby planet affect ship movements and manueverability.

Enemies	6
Obstacles (space debris)	Yes
Heavenly Bodies (planets / space stations)	Yes

Level 4: Randomly Generated Level

All levels beyond Space Station Alpha are randomly generated and will have increasing difficulty levels by raising the number of enemies, enemy strength and speed, and collideable objects. At this point the game becomes an unending test of the player's ability to outperform the enemy ships using special tactics and manuevers until the enemies become too quick to defeat.

Acceleration = Engine Force / Ship Mass
Velocity = Velocity + Acceleration * Delta Time
Position Change = Velocity * Delta Time

Functions for movment in 3D space
These Functions manipulate ship movement and orientation in the world

```
// gets the orientation and position of the opbject in the
world
D3DXMATRIX GetOrientation(void) {

   D3DXMATRIX Rotation(m_Orientation), Transform;
   Rotation._41 = Rotation._42 = Rotation._43 = 0;
   D3DXMatrixTranspose(&Rotation, &Rotation);
   D3DXMatrixIdentity(&Transform);
   Transform._41 = -(m_Orientation._41);
   Transform._42 = -(m_Orientation._42);
   Transform._43 = -(m_Orientation._43);
   D3DXMatrixMultiply(&Transform, &Transform,
&Rotation);
   return Transform;
}

// gets the orientation and position of the opbject in the
world
void SetOrientation(D3DXMATRIX *Orientation) {
   memcpy(&m_Orientation, Orientation,
sizeof(D3DXMATRIX));
}

//translates the object by its right vector
void TransX(float distance) {
        m_Orientation._41 += distance;
}

//translates the object by its up vector
void TransY(float distance) {
        m_Orientation._42 += distance;
}

//translates the object by its look vector
void TransZ(float distance) {
        m_Orientation._43 += distance;
}
```

```
//translates the object by a vector in its world space
void Trans(D3DXVECTOR3 distance)
{
        m_Orientation._41 += distance.x;
        m_Orientation._42 += distance.y;
        m_Orientation._43 += distance.z;
}

//rotates the object around its right vector
void RotateX(float angle)
{
        D3DXMATRIX Rotation;
   D3DXMatrixRotationX(&Rotation,
D3DXToRadian(angle));
   D3DXMatrixMultiply(&m_Orientation,
&m_Orientation, &Rotation);
}

//rotates the object around its up vector
void RotateY(float angle)
{
        D3DXMATRIX Rotation;
   D3DXMatrixRotationY(&Rotation,
D3DXToRadian(angle));
   D3DXMatrixMultiply(&m_Orientation,
&m_Orientation, &Rotation);
}

//rotates the object around its look vector
void RotateZ(float angle)
{
        D3DXMATRIX Rotation;
   D3DXMatrixRotationZ(&Rotation,
D3DXToRadian(angle));
   D3DXMatrixMultiply(&m_Orientation,
&m_Orientation, &Rotation);
}
```

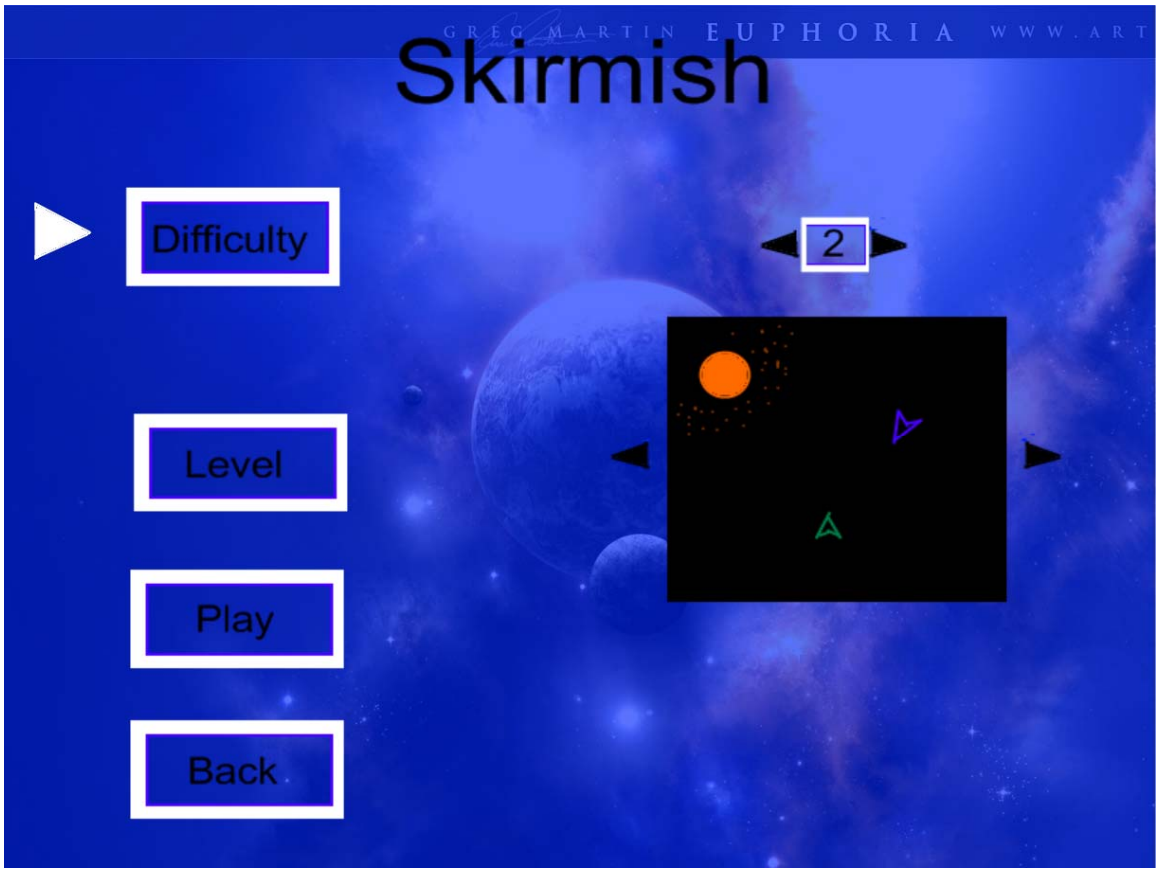
Interface

Main Menu

In this menu the user is able to choose if they would want to play campaign or skirmish mode, also the user is able to choose between changing the options of the game, viewing the high scores or exiting the game. Every time the user enters the main menu the word ZFighter is flashed on the screen.

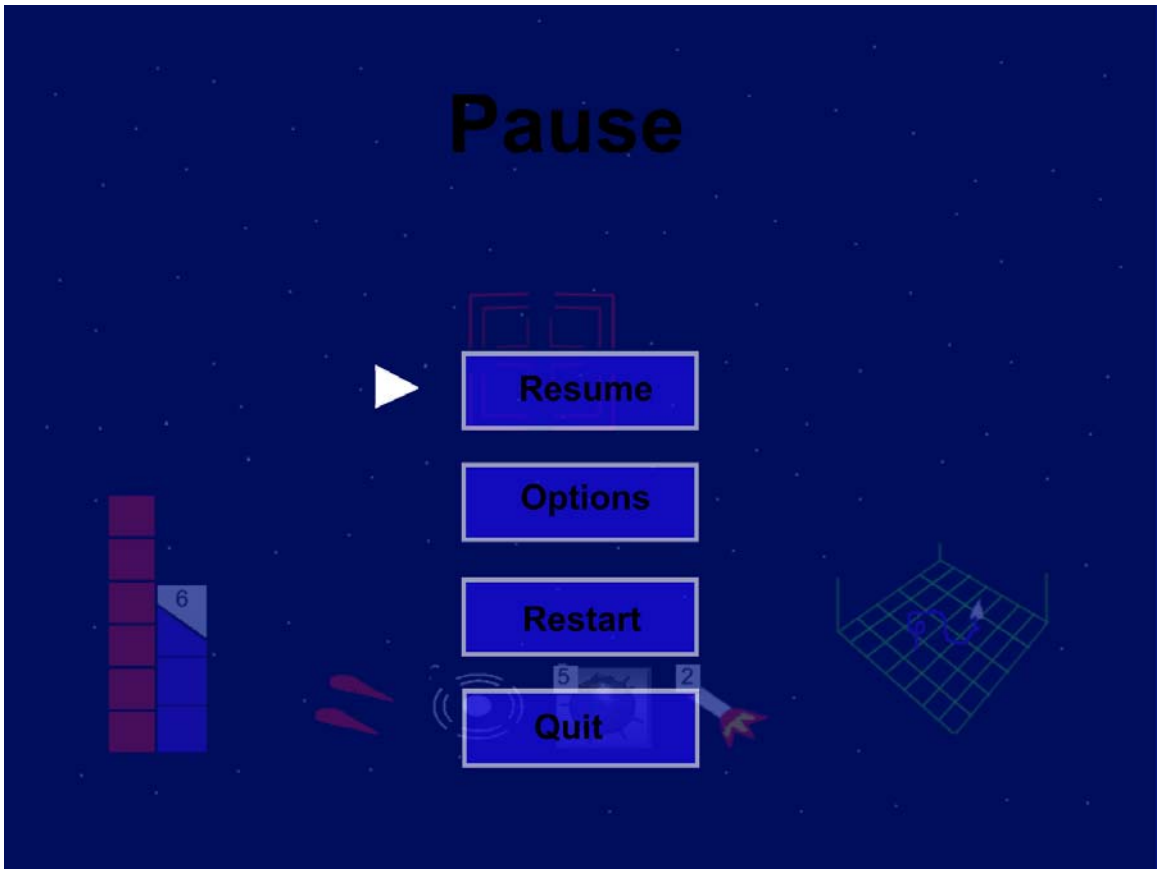
Skirmish Menu

In this menu the user is able to choose the difficulty level and the level that the user would wish to play “Skirmish” mode in. The user is able to move the difficulty left and right for more or less, the user is also able to do this to choose the level they wish to play in. While selecting the level the user is able to see a small image of the level that they are going to choose.



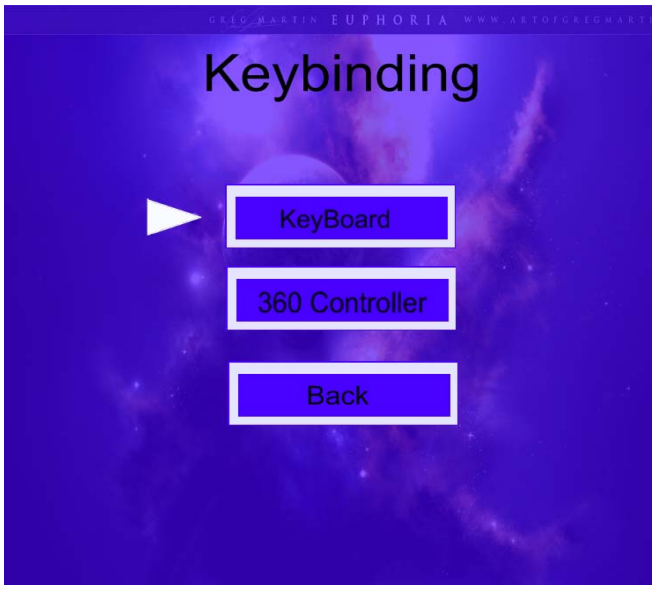
Pause Menu

The user is able to use this menu while the user is playing the game and for whatever reason the user must stop the game. In this menu the user can choose to resume the game, change the game options, restart the fight that he or she is currently involved in or quit the user's current game.

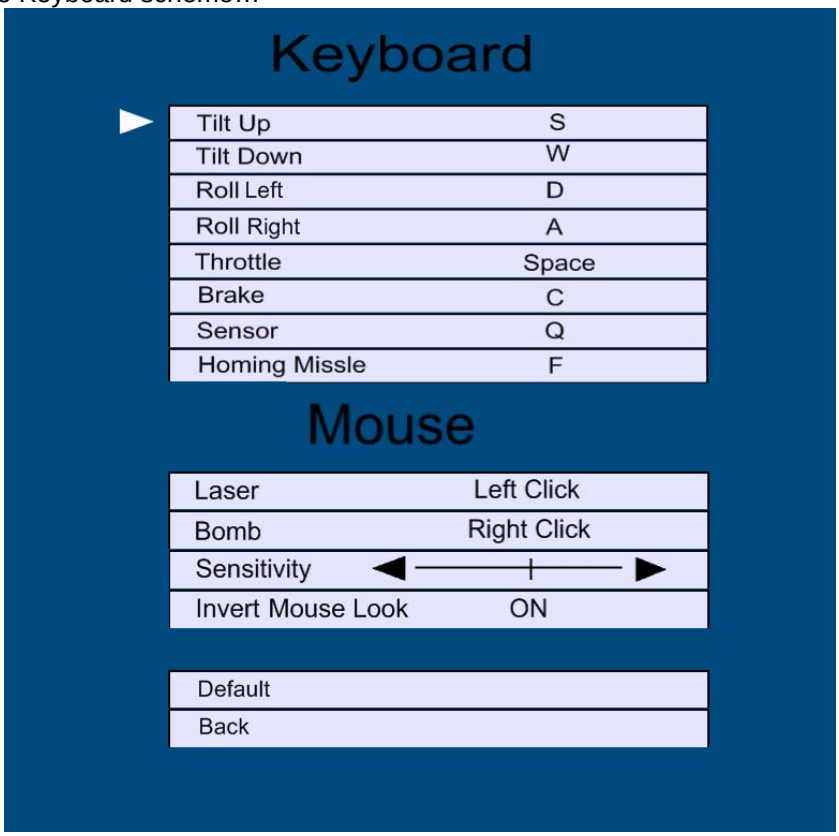


Key-Binding Menu

In this menu the user is able to determine what controller scheme he or she would prefer, the user is able to choose between the keyboard and mouse or the XBOX 360 controller.



Looking at the Keyboard scheme…



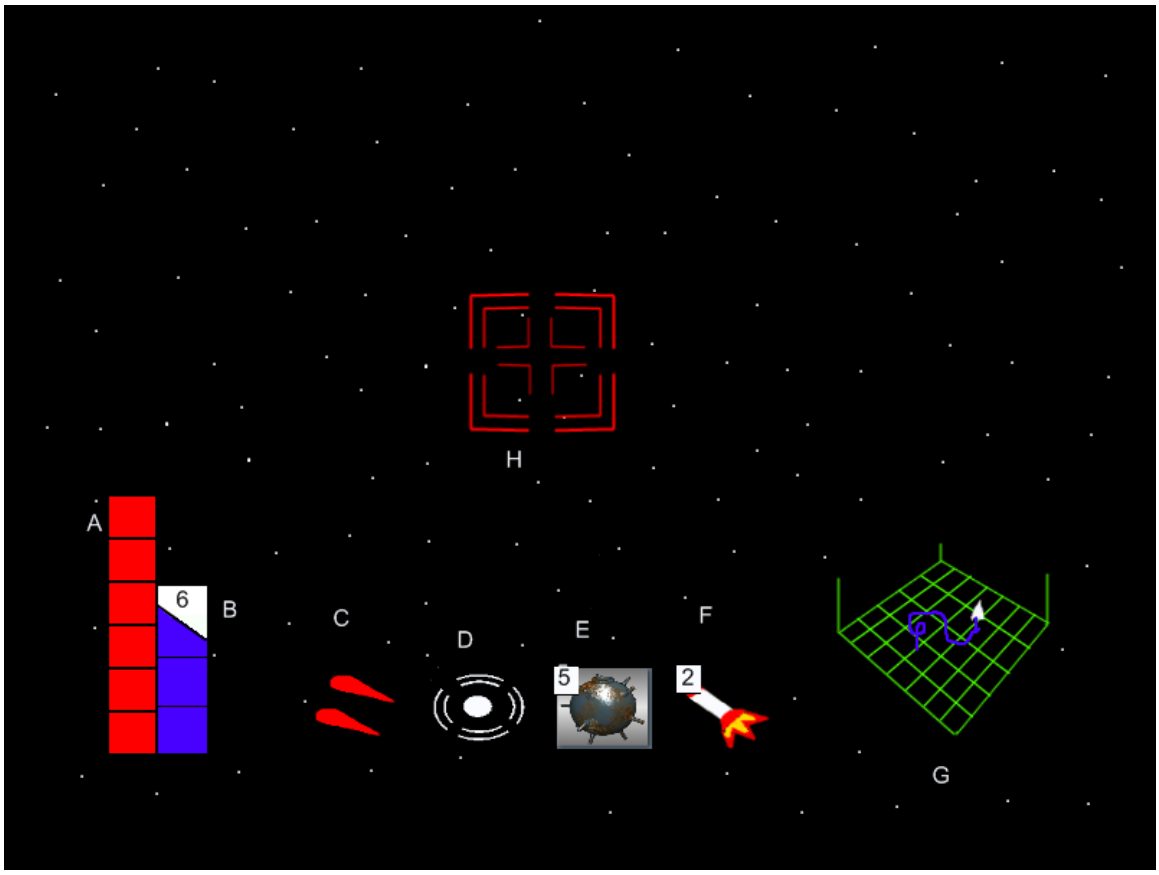
Looking at the XBOX 360 controller scheme…

The controls in the XBOX 360 controller scheme the player is able to change how they would move their ship, for whatever reason imaginable, for example is they wanted to switch Tilt Up to Left Thumb-stick down to Left Thumb-stick up, and switch Tilt Down to Left Thumb-stick up to Left Thumb-stick down.

Heads Up Displays

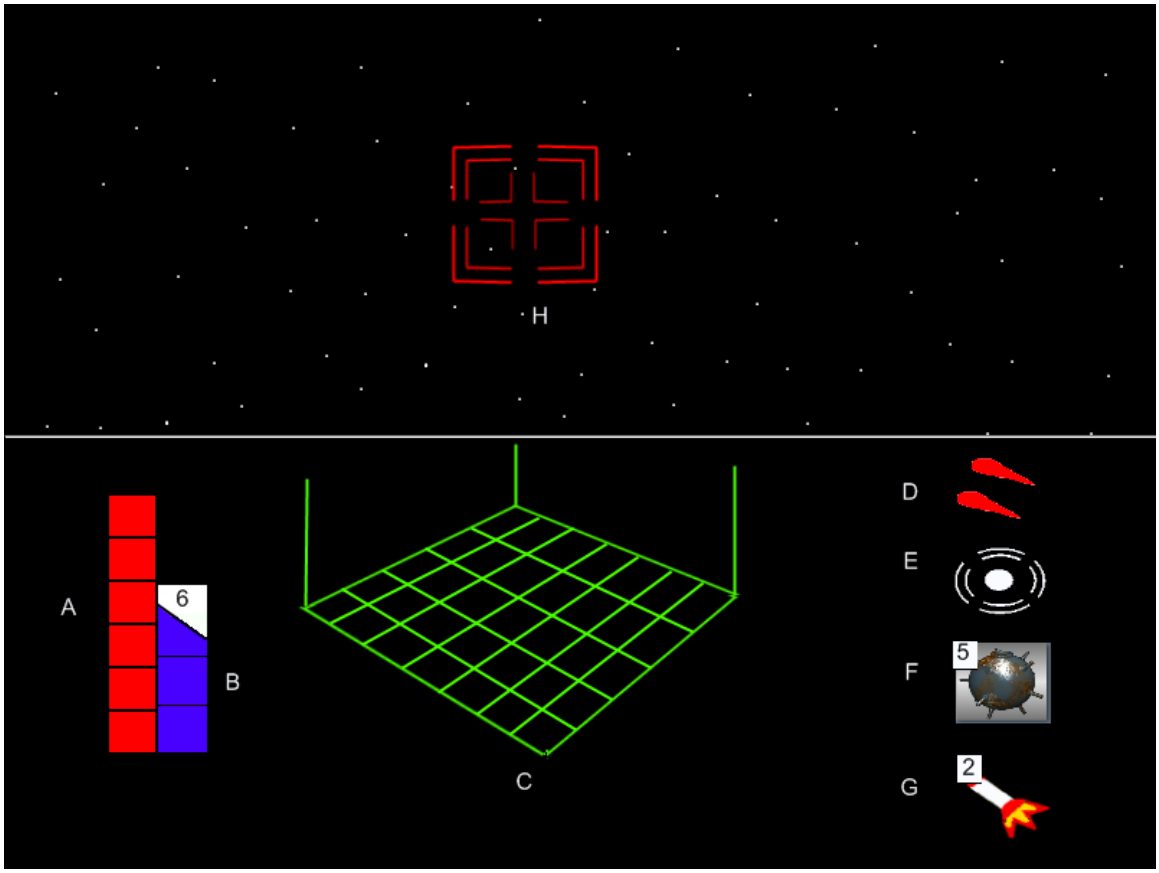
. H.U.D. # 1 (Heads Up Display)

The player can switch between views at any time with the change view input.



A. Armor – once this bar reaches zero (or it empties) the user's ship is killed.
B. Lives – once the bar reaches zero (or it empties) the user looses. If the user obtains more than three lives the number of how many lives the user has is displayed on the top of the life bar.
C. Laser – the user's main weapon.
D. Sensor – an item that goes off when an enemy is too close to it, letting the user know the enemy's position.
E. Mine – a trap that the user and enemy leaves behind to explode whenever someone is too close to it.
F. Homing missile – a weapon that will follow an enemy's trail when fired (in the enemy's case this missile will follow a user's turn).
G. Radar – a tiny radar that will let the user know where his or her trail is, it will also inform the user when a sensor activates.
H. Reticle – a grid that informs the user where his or her laser and homing missile is aimed towards.

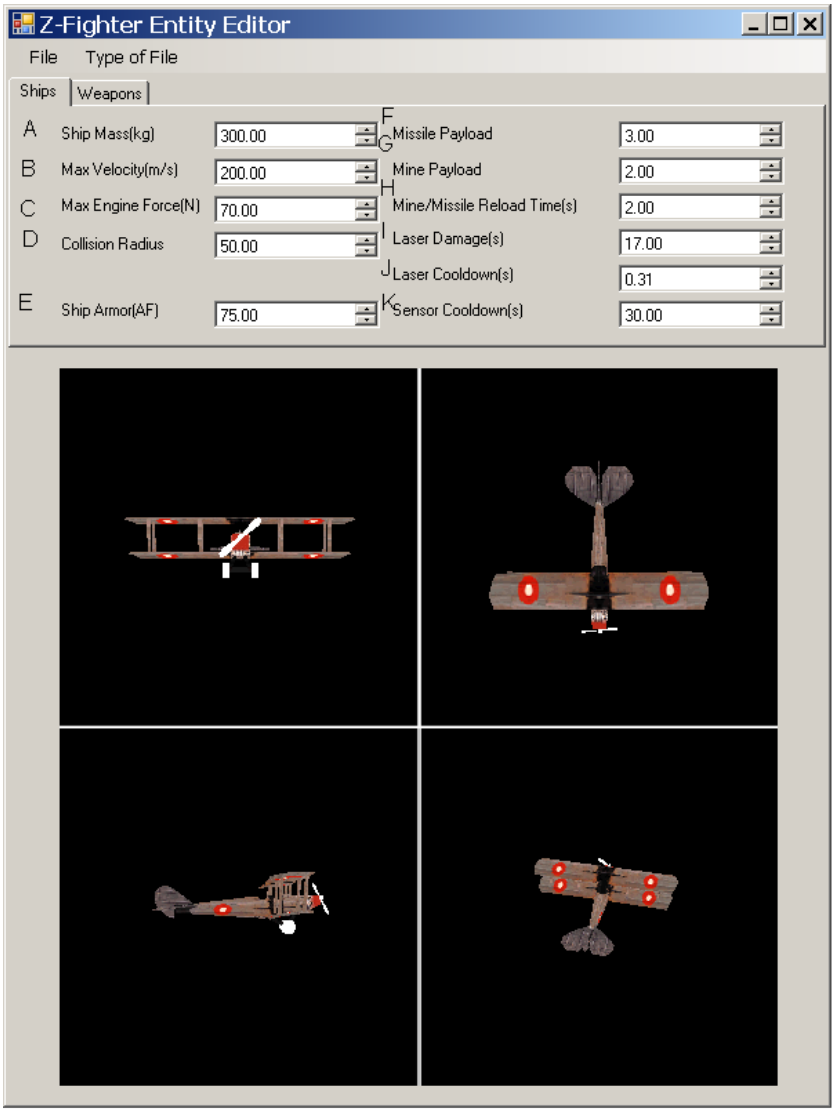
H.U.D. # 2 (Heads Up Display)



A. Armor – once this bar reaches zero (or it empties) the user's ship is killed.
B. Lives – once the bar reaches zero (or it empties) the user looses. If the user obtains more than three lives the number of how many lives the user has is displayed on the top of the life bar.
C. Radar – a tiny radar that will let the user know where his or her trail is, it will also inform the user when a sensor activates.
D. Laser – the user's main weapon.
E. Sensor – an item that goes off when an enemy is too close to it, letting the user know the enemy's position.
F. Mine – a trap that the user and enemy leaves behind to explode whenever someone is too close to it.
G. Homing missile – a weapon that will follow an enemy's trail when fired (in the enemy's case this missile will follow a user's turn).
H. Reticle – a grid that informs the user where his or her laser and homing missile is aimed

Tools

Object Editor



A) Ships Mass in Kilograms.
B) The number representing the maximum velocity the ship is able to travel at.
C) Changes the value for the maximum force the ships engine exerts on the ship.
D) The radius from the center point of the mesh the represents the bounding circle for collision detection.
E) The value representing the Armor Value of the ship. The higher the Armor the more hits the ship can take in battle.
F) This represents the maximum number of missiles a ship entity can have.
G) This represents the maximum number of mines a ship entity can have.
H) Represents the time required before a mine of missile weapon can be fired again after already deploying one.
I) The amount of armor damage the ship entity's laser weapon will do to enemy ships.
J) The time before another laser can be fire.
K) The time before another sensor can be deployed.

The Object Editor is going to Save and Read in data in this order.
Binary:

ShipType - short (2 bytes) this is going to be read in first so we know the proper ship template to fill the rest of the data with.
Mass – int (4 bytes)
Max Veclocity – D3DXVECTOR3(sizeof(D3DXVECTOR3))
Max Engine Force - D3DXVECTOR3(sizeof(D3DXVECTOR3))
Health/Armor – int (4 bytes)
Max Missiles – int (4 bytes)
Max Mines- int (4 bytes)
Missile Mine Cool down – float(4 bytes)
Sensor Cool Down – float(4 bytes)
Laser Damage – int (4 Bytes)
Laser cool down – float (4 bytes)

10 different things altogether.

This is a sample XML file format for players and enemies the Type will be the difference

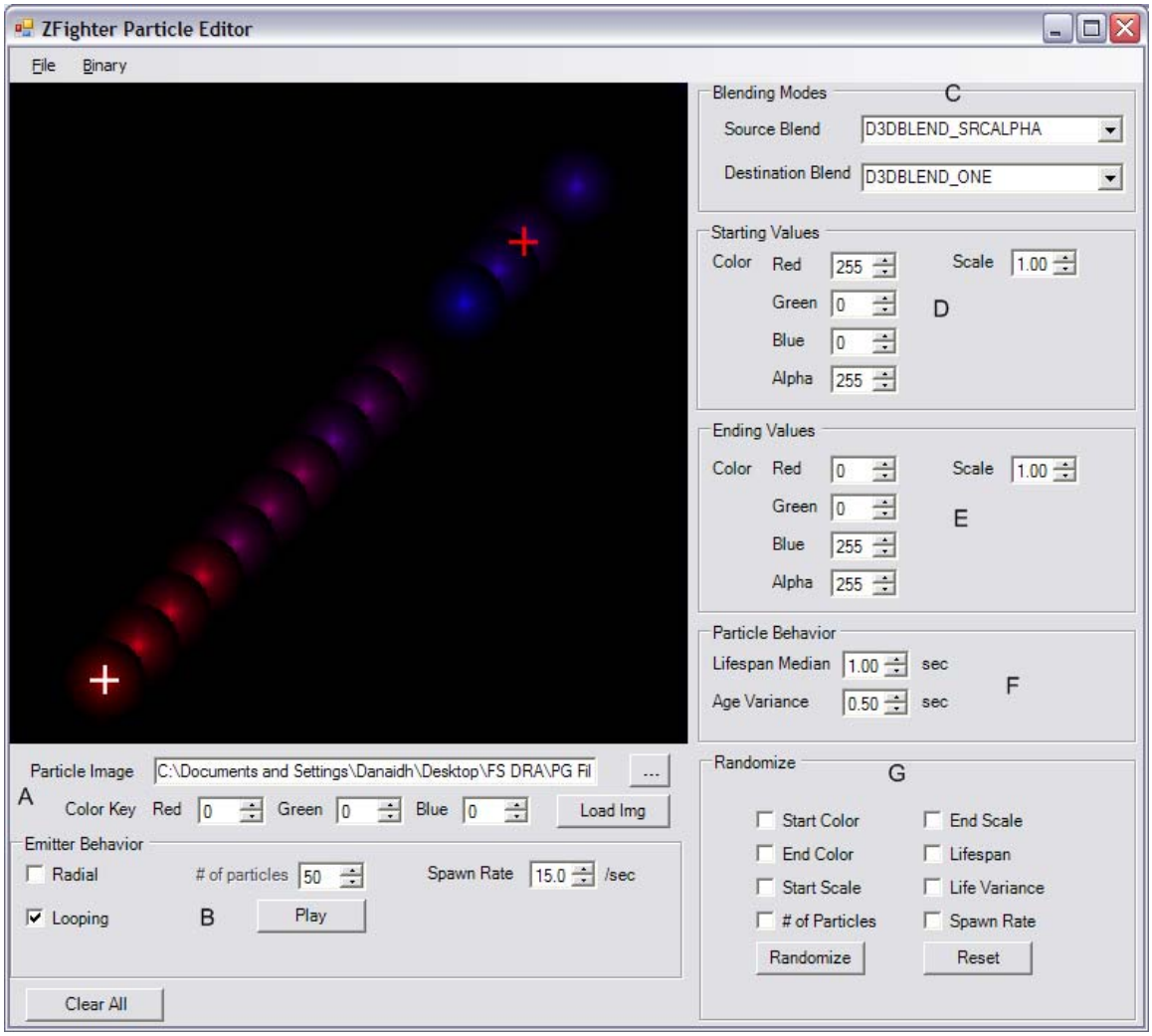
<Ship Mass= “0” MaxVelX=”0” MaxVelY=”0” MaxVelZ=”0” MaxForceX=”0” MaxForceY=”0” MaxForceZ=”0”>
    <Type Health=”0” MaxMissiles=”0” MaxMines=”0” MineMissileCoolDown=”0” SensorCoolDown=”0” LaserDamage=”0” LaserCoolDown=”0”>
    </Type>
</Ship>

Weapon

<Weapon Mass= “0” MaxVelX=”0” MaxVelY=”0” MaxVelZ=”0” MaxForceX=”0” MaxForceY=”0” MaxForceZ=”0”>
    <Type Damage=”0” >
    </Type>
</Weapon>

<Sensor Range= “0”>
</Sensor>

Particle Editor

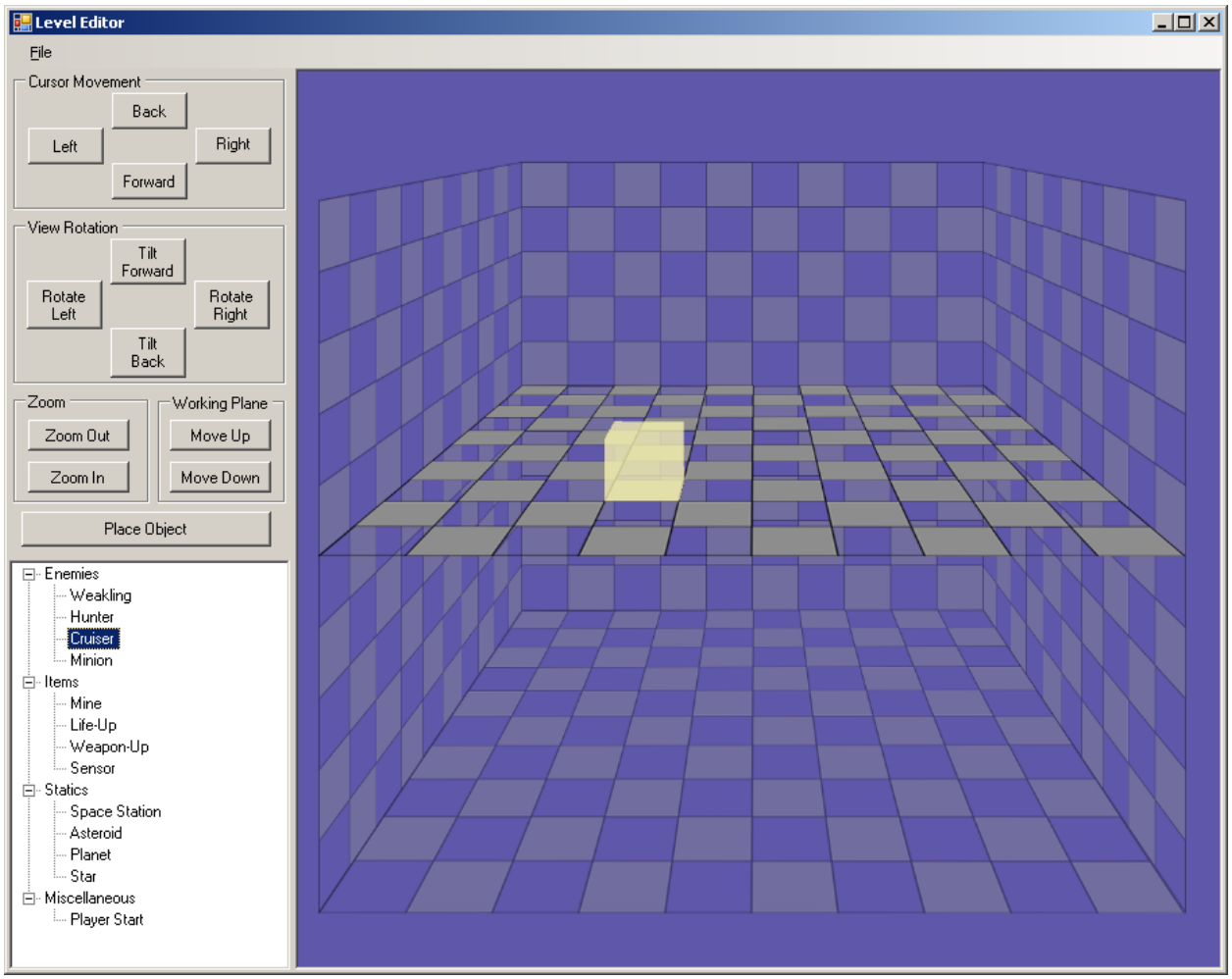


A. The Image used for the particle.
B. The specific behaviors displayed by the particle's emitter. Also a the play button allows the user to see the particle emitter in action.
C. The mode that is going to be used by the particle.
D. The starting values that the particles are going to have (for color and size).
E. The ending values that the particles are going to have (for color and size).
F. The lifespan of each particle.
G. Allows the user to randomize any of the particle's values.

The Particle Editor will output in both binary and xml from the menu toolbar. The xml format is as follows:

```
<Particle>
    <Img>
            <Filename>burst.jpg</Filename>
            <KeyColor R="0" G="0" B="0">
    </Img>
    <Looping>false</Looping>
    <Radial>false</Radial>
    <BlendModes SrcBlend=”0” DestBlend=”0” />
    <SpawnRate>20.00</SpawnRate>
    <Lifespan>3.40</Lifespan>
    <Variance>2.00</Variance>
    <Start>
            <Color A="255" R="255" G="0" B="0">
            <Scale>1.00</Scale>
    </Start>
    <End>
            <Color A="255" R="255" G="0" B="0">
            <Scale>1.00</Scale>
    </End>
</Particle>
```

World Editor



The main level editing will be done in one window. The user builds the world by selecting them from the object list, maneuvering the cursor to the desired spot, and laying down the object.

GUI Components

Editor Window

- Editing Plane Grid
  This plane grid, shown more prominently than the others, serves to visually orient the user to where, in the context of the other planes, the object locations and the cursor location. This plane will move vertically with the cursor, and will always be on the bottom of the cursor.
- The Editing Cursor
  This is the main editing cursor. It shows a visual representation of where objects will be placed. It will always be atop the Editing Plane, so the user will not get disoriented.

Object Selector

This tree-view serves as the object selector. This is enumerated from the object editor files.

Navigation Panel

- Cursor Movement
  This moves the cursor along the Editing Plane grid in the selected direction
- View Rotation
  This rotates the view of the level, so the user can get a different perspective.
  Rotate Left / Rotate Right
  Rotates the view 90^{o} to the left or right
  Tilt Forward / Tilt Back
  This rotates the view slightly up or down, giving the user various degrees of viewpoint.
- Zoom
  Zooms the view of the level in or out
- Working Plane
  Shifts the Editing Grid plane up or down
- Place Object
  Places an object selected within the Object Selector in the world

This is the World Editor's file output schema. This will be outputted into binary and XML.

Note: Empty world tiles are ignored by the game, and not exported by the editor. This reduces the size of the level file significantly and aids in readability and editability. File extensions are not final.

```
<Map>
	<Instances>
		<Statics>
			<Object Filename=".\Objects\Asteroid.sta" Index="0"></Object>
		</Statics>
		<Enemies>
			<Object Filename=".\Objects\Minion.nmy" Index="0"></Object>
		</Enemies>
		<Items>
			<Object Filename=".\Objects\Mine.itm" Index="0"></Object>
		</Items>
	</Instances>

	<ObjectLocation>
		<Index X="0" Y="2" Z="1">
			<Orientation X="-160" Y="30" Z="45"></Orientation>
			<Object Type="Static">0</Object>
		</Index>
		<Index X="5" Y="1" Z="0">
			<Orientation X="90" Y="-30" Z="-160"></Orientation>
			<Object Type="Enemy">0</Object>
		</Index>
		<Index X="2" Y="5" Z="3">
			<Orientation X="100" Y="-10" Z="30"></Orientation>
			<Object Type="Item">0</Object>
		</Index>
		<Index X="5" Y="5" Z="5">
			<Orientation X="90" Y="90" Z="-90"></Orientation>
			<Object Type="Misc">0</Object>
		</Index>
	</ObjectLocation>
</Map>
```

Technology Summary

Graphics Engine

- ZFighter features a fully implemented 3D environment engine. Utilizing a unique camera system, ZFighter will provide you with multiple ways to see the action. DirectX is the main renderer of the system.

Tile System

- ZFighter includes a Map Editor that is also is fully 3D. It features an easy to understand interface and an intuitive design for navigating and placing objects in the world. It will be able to use the data provided inside the Object editor, and be able to export level data in XML and binary formats.

Sound Engine

- ZFighter features immersive 3D sound effects, mp3 based music playback, and ID based sound storage. This will support volume control and 3D panning of sounds. FMod provides the core functionality.

Particle Engine

- ZFighter features a particle manager which handles the creation, updating, and rendering of particle systems. Features include lifetime variance, color variance, spawn rate control and different blending modes.

Animated Textures

- ZFighter features a unique implementation of DirectShow to provide textures that can be rendered in a Direct3D environment. This is accomplished by rendering videos to textures. Utilizing this, the menu system will feature gameplay movies in the background.

Asset Caching

- ZFighter utilizes a cache system to store game assets. Game objects are then instanced from these cached items, greatly reducing load time and providing a centralized management system for all game assets.

Message and Event System

- ZFighter features a robust internal communication system that aids in the interoperability of its modules. This allows each module more independence, and thus eases the ability to find and eliminate costly bugs.

Physics System

- ZFighter utilizes the physical properties of each object to determine movement and control. Physical properties, such as mass, inertia, gravity, and force will all play a part in determining how each ship handles.

Game State Machine

- ZFighter utilizes game states to control the menu system and gameplay.

Code Architectural Overview

CBaseMessage

Description: This module provides the foundation for any given message, as well as holding an enumeration of all message types that have been created. Because it is the base of which all other messages are inherited from, the core message functionality is enumerated here.

Stipulations: The MESSAGEID type is set to unsigned int. An enum defines all of the message ID types.

Interface: The module only contains functions to set and retrieve the message type. Because this is an interface for other messages, thus will never be called by outside code.

MEMBERS

Name	Type	Description
m_Msg	MESSAGEID	ID of the message based upon the enumeration contained within the base message module

METHODS

Return	Name	Type/Parameters	Description
void	SetMessageID	MESSAGEID msg	Allows the child message to set its type upon creation.
MESSAGEID	GetMessageID	void	Allows for external modules to access a given messages type.

enum { MSG_CREATE_PLAYER = 0, MSG_DESTROY_PLAYER = 1, MSG_CREATE_ENEMY = 2, MSG_DESTROY_ENEMY = 3, MSG_MAX}

SUB-MODULES

CCreatePlayerMessage: public CBaseMessage

Description: This module allows the use of the messaging system to create player ships.

Interface: The module has only a constructor and destructor; no parameters are stored within this message type.

CDestroyPlayerMessage: public CBaseMessage

Description: This module allows the use of the messaging system to destroy player ships.

Interface: The module has only a constructor and destructor, no parameters are stored within this message type.

CCreateEnemyMessage: public CBaseMessage

Description: This module allows the use of the messaging system to create enemy ships.

Interface: This modules constructor receives the 3D vector position and orientation of the enemy ship to be created. An accessor is used to retrieve that information for use in the creation process.

MEMBERS

Name	Type	Description
m_vecCoords	D3DXVECTOR3 (don't make it a pointer)	This parameter stores the world location that the enemy should be created at.
m_vecOrient	D3DXVECTOR3	X, Y, Z Rotation values stored in a vector
m_nEnemyType	short	Type of enemy ship

METHODS

Return	Name	Type/Parameters	Description
void	CCreateEnemyMessage	D3DXVECTOR3 vecCoords	The constructor of the message receives the world coordinates of the enemy ship.
D3DXVECTOR3	GetParam	void	Accessor for the world coordinates of the ship

SUB-MODULES

CDestroyEnemyMessage: public CBaseMessage

Description: This module allows the use of the messaging system to destroy enemy ships.

Interface: This module's constructor receives a pointer to the enemy ship that is to be destroyed. An accessor is used to retrieve that pointer.

MEMBERS

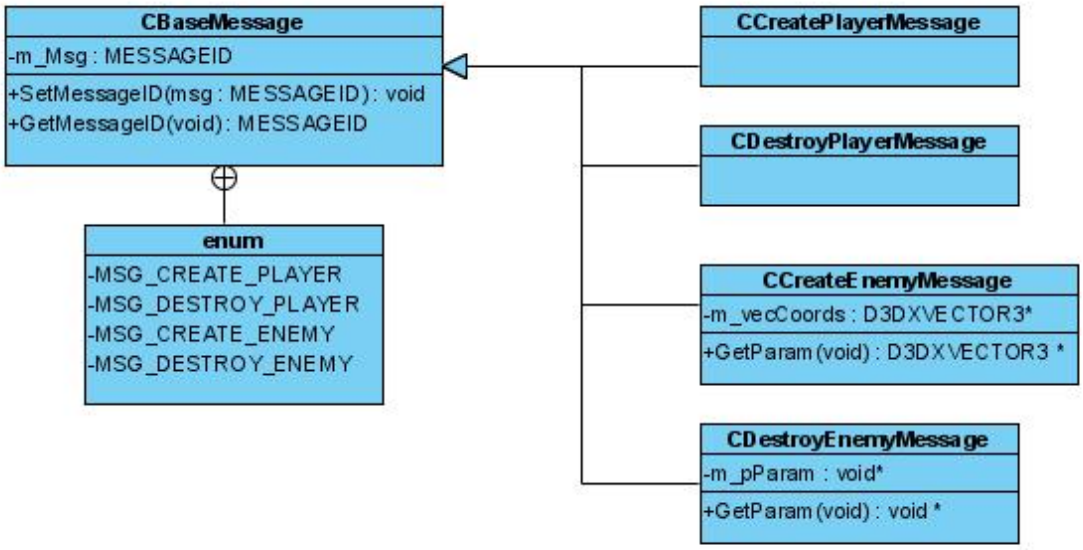
Name	Type	Description
m_pParam	CEnemyShip*	The pointer of the enemy ship to be destroyed.

METHODS

Return	Name	Type/Parameters	Description
void	CDestroyEnemyMessage	CEnemyShip* pParam	The constructor of the message recieves a pointer of the enemy to be destroyed.
CEnemyShip*	GetParam	void	Accessor for the enemy ship to be destroyed.

SUB-MODULES

Module Diagram for Message System



CGameTime

Description: This is the interface for all time related functions. It is responsible for loading and unloading the high resolution timer, computing the frame time delta, calculating the frame rate, and calculating the time based multiplier.

Stipulations: As a singleton, only one of these objects will ever exist in the lifetime of the program. Calling the GetInstance() function is necessary as it not only retrieves this object, but will also initialize the high resolution timing if it hasn't been initialized yet. The Release() function must likewise be called to restore the timing resolution.

Interface: This class will need to be accessed by any other class needing to calculate time based events and/or time based movement.

MEMBERS

Name	Type	Description
m_Instance	CGameTime	Stores the instance of the timer
m_bInit	bool	Stores whether the timer has been initialized
m_TimeCaps	TIMECAPS	Holds the windows TIMECAPS options. This is used to load and unload the high resolution timing for the game.
m_dwLastTime	DWORD	Stores the last time the timer was updated
m_dTimeStep	double	Stores the time step between the last update and the current update
m_dwFrameRate	DWORD	Stores the current framerate
m_dTimeMultiplier	double	Stores the time based fraction used to calculate time based functions, such as movement

METHODS

Return	Name	Type/Parameters	Description
CGameTime*	GetInstance	void	Gets the instance of the CGameTimer. If not previously instanced, sets up the high resolution timing
void	Release	void	Releases any memory used by the timer and unloads the high resolution timing
void	Update	void	Updates the timer by one frame

SUB-MODULES

CFMod

Description: This wrapper holds FMod sound functionality. All 2D and 3D sounds are handled within this module. All of the sounds are loaded into an array via the CreateSound function. This function returns an integer ID that will be used to access a particular sound for various functions.

Stipulations: This module is a singleton, and must be initialized by the game class prior to use. It must also be shutdown prior to exit of the game.

Interface: This module contains functions for modifying and playing audio files. When something calls for a sound to be played, they will access the singleton interface to play the sound.

MEMBERS

Name	Type	Description
m_pInstance	CFMod*	Pointer to the of the CFMod class
m_vSounds	Vector<FMOD::Sound*>	Storage of all sounds used by the module.
m_pSys	FMOD::System*	Pointer to FMod system object
m_pMusicChannel	FMOD::Channel*	Pointer to FMod channel object
m_Result	FMOD_RESULT	FMod result for error handling

METHODS

Return	Name	Type/Parameters	Description
CFMod*	GetInstance	void	Gets the instance of the CFMod module. If it does not exist, it is created.
Void	DeleteInstance	void	Deletes the instance
Bool	Init	int nMaxChannels, FMOD_INITFLAGS flags	Initialize the fmod system with max number of channels and special flags.
Int	CreateSound	char* szFilename	Returns the vector index of the newly created sound
bool	PlaySound3D	int nSoundID, D3DXVECTOR3 vSoundPos	Plays the sound at the parameter index
bool	PlayLoop	Int nSoundID	Plays a looping track
Bool	PlaySound2D	Int nSoundID	Plays a normal sound
Bool	Stop	Int nSoundID	Stops a specific sound given by it's ID

SUB-MODULES

CDirect3D

Description: This is the main wrapper for all Direct3D functionality. Anything and everything that is required to interface with the Device will utilize this wrapper.

Stipulations: As a singleton, this class will exist only once throughout the lifetime of the program. Calling the GetInstance() function will retrieve a pointer to this object. However, the Initialize() function must be called before any Direct3D functionality can be used. The Shutdown() function must likewise be called at the end of the program to properly release the modules used by Direct3D.

Interface: This class will be accessed by any module requiring the core of Direct3D. Most of the functionality will be used by other DirectX subsystems, and other modules will primarily use the render functions.

MEMBERS

Name	Type	Description
m_Instance	CDirect3D*	Instance of the CDirect3D class
m_lpDirect3D	LPDIRECT3D9	Pointer to the Direct3D object
m_lpD3DDevice	LPDIRECT3DDEVICE	Pointer to the Direct3D device
m_lpSprite	LPD3DXSPRITE	Direct3D sprite interface
m_lpFont	LPD3DXFONT	Direct3D font interface
m_lpLine	LPD3DXLINE	Direct3D line interface
m_PresentParams	D3DPRESENT_PARAMS	The Device's presentation parameters

METHODS

Return	Name	Type/Parameters	Description
CDirect3D*	GetInstance	void	Returns the instance of the CDirect3D object
bool	Initialize	HWND hwnd, int nWidth, int nHight, bool bWindowed bool bVsync	Initializes Direct3D
void	Shutdown	void	Shuts down Direct3D
void	Clear	unsigned char cRed, unsigned char cGreen, unsigned char cBlue	Clears the screen to the given color
bool	DeviceBegin	void	Signals the device to begin rendering
bool	SpriteBegin	void	Readies the sprite for rendering
bool	LineBegin	void	Readies the line for rendering
bool	DeviceEnd	void	Signals the device to stop rendering

CDirect3D Methods Continued

bool	SpriteEnd	void	Stops the sprite renderer
bool	LineEnd	void	Stops the line renderer
void	ChangeDisplayParam	int nWidth, int nHeight, bool bWindowed	Changes the display parameters
void	DrawRect	RECT rRect, unsigned char cRed, unsigned char cGreen, unsigned char cBlue	Draws a rectangle with the given colors
void	DrawLine	int nX1, int nY1, int nX2, int nY2 unsigned char cRed, unsigned char cGreen, unsigned char cBlue	Draws a line from (x1, y1) to (x2, y2) in the given color
void	DrawText	char *lpzText, int nX, int nY unsigned char cRed, unsigned char cGreen, unsigned char cBlue	Draws text starting at the point (x, y) in the given color
void	CreatePrimitive	PRIMITIVE_TYPE Type, LPD3DXMESH* ppMesh	Creates a basic DirectX primitive mesh of Type, and sets it to ppMesh
void	SetRenderState	D3DRENDERSTATETYPE Type, DWORD dwValue	Changes the render state of the device
void	GetRenderState	D3DRENDERSTATETYPE Type, DWORD* pdwValue	Get the render state of type Type, and set it to pdwValue
void	Present	RECT* rDestRect	Presents the completed render. rDestRect describes where in the client area to render to, or the whole client area if it is NULL

SUB-MODULES

enum PRIMITIVE_TYPE {PRIM_BOX, PRIM_SPHERE, PRIM_CYLINDER,PRIM_TEAPOT, PRIM_POLYGON, PRIM_TORUS, PRIM_MAX };

CDirectShowFilter

Description: This is the custom video filter to be used by DirectShow. Much like defining the vertex struct for Direct3D, DirectShow requires this custom filter definition to play back video.

Stipulations: This will never be called inside the program. It is used by DirectShow to validate, parse, and render the video. DirectShow will call all of these functions itself internally. Because DirectShow initializes and handles all filter memory internally, and they last for the life of the program, the filter must never be cleaned up manually.

Interface: This module will exclusively be used by DirectShow to store and play videos. The texture will be referenced within CAnimatedTexture.

MEMBERS

Name	Type	Description
m_pTexture	IDirect3DTexture9*	Texture where the movie playback will occur on
m_Format	D3DFORMAT	Stores texture properties
m_lVideoWidth	LONG	Pixel width of the video
m_lVideoHeight	LONG	Pixel height of the video
m_lVideoPitch	LONG	Video surface pitch

METHODS

Return	Name	Type/Parameters	Description
HRESULT	CheckMediaType	const CMediaType * MediaType	Checks the contents of the media's interface for video
HRESULT	SetMediaType	const CMediaType * MediaType	Parses MediaType for video information and assigns it to the texture
HRESULT	DoRenderSample	IMediaSample *pMediaSample	Copies the video data onto the texture for rendering

SUB-MODULES

CAnimatedTexture

Description: This handles all DirectShow methods for video playback. Because the movie will be rendered to a texture, this will allow both Direct3D and DirectShow to cooperate at the same time without conflicts. This is advantageous because we can then leverage DirectShow to create animated textures in Direct3D.

Stipulations: Because of how DirectShow handles memory management, these textures must not be manually released. Load must be called with a valid file name to the movie file to properly set up the animated texture. Unload must be called to clean up the Media subsystems and the graph system.

Interface: This will be used by the CAttractState menu class for the background movie, but its functionality can be extended to any object that uses texture data and wants it to be animated.

MEMBERS

Name	Type	Description
m_pGraph	IGraphBuilder*	Used to register the filter and to load media
m_pMediaControl	IMediaControl*	Interface used to control media playback
m_pMediaPosition	IMediaPosition*	Used to determine the position in the media
m_pMediaEvent	IMediaEvent*	Interface to handle media events, such as when the playback is complete
m_pTexture	IDirect3DTexture9*	Texture that will be rendered to

METHODS

Return	Name	Type/Parameters	Description
bool	Load	char * szFilename	Loads a file for playback
bool	Unload	void	Frees all necessary data
bool	EndOfAnimation	void	Returns true when the animation has finished playback
void	Play	void	Plays the animation
void	Stop	void	Stops the animation
void	Restart	void	Restarts the animation
void	GoToTime	REFTIME Time	Go to a specified time within the animation
IDirect3DTexture9*	GetTexture	void	Returns the texture that has been rendered

SUB-MODULES

CCamera

Description: This is the basic camera class. All rendering will be done inside the Render function. Because all rendering is handled here, this allows the possibility to have multiple cameras throughout the world, each being fully functional. The m_prViewportRect member points to what part of the client window that the camera should render to.

Stipulations: CCamera is passive. It requires itself to either be bound to another object or moved by the actions of another update. Before the camera can be used, the BuildPerspective function must be called to set up the Projection matrix. The view matrix must also be populated for the camera to work properly.

Interface: This is used whenever some rendering is to take place.

MEMBERS

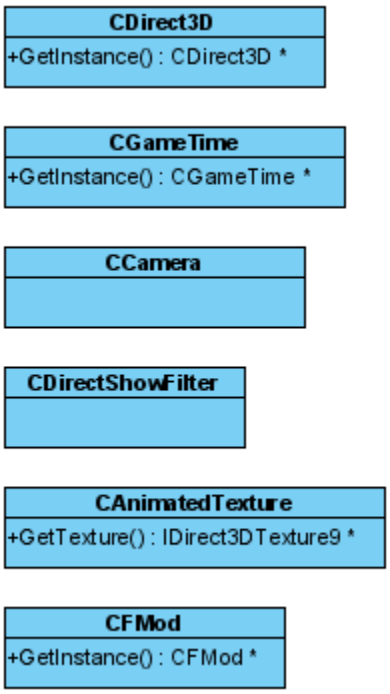
Name	Type	Description
m_matProjection	D3DXMATRIX	The camera's projection matrix
m_matView	D3DXMATRIX	The camera's view matrix
m_prViewportRect	RECT*	Where in the window client to render. Default is set to NULL to render to the entire screen

CCamera METHODS

Return	Name	Type/Parameters	Description
void	BuildPerspective	float fFOV, float fAspect, float fZNear, float fZFar	Builds the perspective matrix based on the parameters
void	RotateLocalX	float fX	Rotates the camera's local pitch
void	RotateLocalY	float fY	Rotates the camera's local yaw
void	RotateLocalZ	float fZ	Rotates the camera's local roll
void	TranslateLocal	D3DXVECTOR3 vAxis	Translate the camera's local position by vAxis
void	TranslateLocalX	float fX	Translate the camera's local X position by fX
void	TranslateLocalY	float fY	Translate the camera's local Y position by fY
void	TranslateLocalZ	float fZ	Translate the camera's local Z position by fZ
void	RotateGlobalX	float fX	Rotates the camera's global pitch
void	RotateGlobalY	float fY	Rotates the camera's global yaw
void	RotateGlobalZ	float fZ	Rotates the camera's global roll
void	TranslateGlobal	D3DXVECTOR3 vAxis	Translate the camera's global position by vAxis
void	TranslateGlobalX	float fX	Translate the camera's global X position by fX
void	TranslateGlobalY	float fY	Translate the camera's global Y position by fY
void	TranslateGlobalZ	float fZ	Translate the camera's global Z position by fZ
void	Render	void	Renders the scene into the window client area

SUB-MODULES

Module Diagram for Wrapper Singletons



CXInput

Description: This module contains all functionality of handling DirectX XInput. This module is used by the player, menus and anything else that needs input. This module contains specifics pertaining to the XBOX360 controller input.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will not need to be accessed by any other modules though it will be frequently called upon for its functionality. Most of the time this module's functionality will be accessed by the player class.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state.
m_wPrevControllerState	WORD	Stores the previous controller state.
m_pInstance	CXInput *	Pointer to the main instance for XInput.
m_vibControllerVibration	XINPUT_VIBRATION	Holds the value of the controller's vibration (When vibration right and left is set to zero, the vibration is stopped)

CXInput METHODS

Return	Name	Type/Parameters	Description
static CXInput*	GetInstance	void	Get the instance of the CXInput Class.
void	DeleteInstance	void	Deletes the instance.
XINPUT_STATE	GetXInputState	void	This is the function is called when ever the input from the controller needs to be known. This function also checks if the controller is still connected.
void	SetControlVibration	Int nLowFreq – the frequency given to the left vibration motor. Int nHighFreq – the frequency given to the right vibration motor. (left, right)	This function initializes the game with information pertinent to help initialize other key modules. This function handles most initialization in the game.
bool	CheckButton	WORD wButton – the button be checked if pressed.	This function checks to see if the button sent in is being pressed.
bool	CheckBufferedButton	WORD wButton – the button be checked if pressed.	This function checks to see if the button sent in is being pressed. This function will only detect the button once if the button is held down.
unsigned char	CheckTrigger	WORD wButton – the button be checked if pressed. Int nTrigger – Determines which trigger is being pressed.	This function checks to see if a trigger has been pressed and returns the sensitivity.
short	CheckThumbMoving	Int nThumbStick – Determines which thumbstick is being moving. (what if BOTH are??)	This function checks to see which thumb-stick is being used and its sensitivity then returns the current value of the thumb-stick that it is checking for.

SUB-MODULES

CGameState

Description: This is the game state base class. It contains no information. Its purpose is to provide a basic template for all state modules that will inherit from this.

Stipulations: This module is an abstract base of the game state system. Each game state will inherit its functionality from this module.

Interface: The functions in this class are virtual functions that must be redefined by child classes.

METHODS

Return	Name	Type/Parameters	Description
bool	Init	void	This function initializes the state upon switching to the state.
void	Update	void	This function handles time based update mechanics.
void	Exit	void	This function handles the shutdown of the state before a new state is activated.
Void	Input	void	Because only the top most state should receive input
Void	Render	void	

CMainMenu: public CGameState

Description: This module contains the main menu and the functionally that is going to be provided by it.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will have to be accessed by the other menus that will branch off this menu.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state.
m_pInstance	CMainMenu *	Pointer to the main instance of the Main Menu.
m_nSelectedOption	int	Determines which option in the main menu is selected.

METHODS

Return	Name	Type/Parameters	Description
static CMainMenu*	GetInstance	void	Get the instance of the CMainMenu Class.
void	DeleteInstanceNOOO!!!	void	Deletes the instance.
bool	Init	void	Initalizes the state.
void	Update	void	This function updates the CMainMenu Class.
void	Exit	void	Exits the state.

SUB-MODULES

COptionsMenu: public CGameState

Description: This module contains the options menu and the functionally that is going to be provided by it.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will have to be accessed by the other menus that will branch off this menu.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state. Why isn't this just a pointer to the CXInput singleton??
m_pInstance	COptionsMenu *	Pointer to the main instance of the Options Menu.
m_nSelectedOption	int	Determines which option in the main menu is selected.

METHODS

Return	Name	Type/Parameters	Description
static COptionsMenu*	GetInstance	void	Get the instance of the COptionsMenu Class.
Void	DeleteInstance	void	Deletes the instance.
Void	Update	void	This function updates the COptionsMenu Class.
Private Void	SetSFXVolume	int nVolume	Sets the sound effects volume.
Private Void	SetVolume	int nVolume	Sets the volume for the game.

SUB-MODULES

CSkirmishMenu: public CGameState

Description: This module contains the skirmish menu and the functionally that is going to be provided by it.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will have to be accessed by the other menus that will branch off this menu.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state.Why not CXInput*??
m_pInstance	CSkirmishMenu *	Pointer to the main instance of the Skirmish Menu.
m_nSelectedOption	int	Determines which option in the main menu is selected.
m_nLevel	int	Holds level data until the user decides to create it.
m_sDificulty	Short	Holds the difficulty for the new level.

METHODS

Return	Name	Type/Parameters	Description
static CSkirmishMenu*	GetInstance	void	Get the instance of the CSkirmishMenu Class.
Void	DeleteInstance	void	Deletes the instance.
Void	Update	void	This function updates the CSkirmishMenu Class.
Private Void	SetLevelDificulty	short sLevelDificulty – the difficulty of the new level.	This function sets the difficulty of the level that is about to be created.
Private Void	SetLevelSelected	Int nLevelSelected – the level that was selected for skirmish.	This function sets the level that is going to be used for skirmish.
Private Void	CreateLevel	Void	This function creates the level that the user just described.

SUB-MODULES

CCredits: public CGameState

Description: This module contains the credits that are going to be displayed on the screen.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will have to be accessed by the other menus that will branch off this menu.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state. Why not CXInput*??
m_pInstance	CCredits *	Pointer to the main instance of the Credits.

METHODS

Return	Name	Type/Parameters	Description
static CCredits*	GetInstance	void	Get the instance of the CCredits Class.
void	DeleteInstance	void	Deletes the instance.
void	Update	void	This function updates the CCredits Class.

SUB-MODULES

CHighScores: public CGameState

Description: This module contains the highscores that are going to be displayed on the screen.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will have to be accessed by the other menus that will branch off this menu.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state. . Why not CXInput*??
pInstance	CHighScores *	Pointer to the main instance of the High Scores.
m_plyPlayerList	vector<PlayerData>	Vector of PlayerData that holds all the player inforation for the high-scores list.

METHODS

Return	Name	Type/Parameters	Description
static CHighScores*	GetInstance	void	Get the instance of the CHighScores Class.
void	DeleteInstance	void	Deletes the instance.
void	Update	void	This function updates the CHighScores Class.

SUB-MODULES

PlayerData – a structure that holds the data of each player for the high-score list.

char* szName – Name of the player.

int nScore – Score of the player.

CPauseMenu: public CGameState

Description: This module contains the pause menu and the functionally that is going to be provided by it.

Stipulations: As a singleton, this object must be created and can exist only once within the entire system. Invoking the GetInstance() method will either create or retrieve the object and its functionality can be accessed through the result. At the end of its use, this object's DeleteInstance() method should be called, and all pointers to the object should be set to NULL and not used otherwise risking access of bad memory.

Interface: This module will have to be accessed by the other menus that will branch off this menu.

MEMBERS

Name	Type	Description
m_xisControllerState	XINPUT_STATE	Stores the current controller state. . Why not CXInput*??
m_pInstance	CPauseMenu*	Pointer to the main instance of the Pause Menu.
m_nSelectedOption	int	Determines which option in the pause menu is selected.

METHODS

Return	Name	Type/Parameters	Description
static CPauseMenu*	GetInstance	void	Get the instance of the CPauseMenu Class.
void	DeleteInstance	void	Deletes the instance.
void	Update	Double dElapsedTime – Time elapsed from the previous call to this function.	This function updates the CPauseMenu Class.

SUB-MODULES

CPlayState: public CGameState

Description: This is the game play state class. It contains information related only to game play. It will initialize each level prior to the beginning of play and will process gameplay inputs.

Stipulations: This module is a game state system inheriting from CGameClass. It is a singleton.

Interface:

MEMBERS

Name	Type	Description
m_pInstance	CPlayState*	Static pointer to Play State singleton object.
m_xisControllerState	XINPUT_STATE	Stores the current controller state.
m_nCurrentLevel	int	Current level number.
m_nDifficulty	int	Current difficulty rating.

METHODS

Return	Name	Type/Parameters	Description
CPlayState*	GetInstance	void	Creates the first and only instance of this class and returns its memory address.
void	DeleteInstance	void	Deletes the instance of this class.
bool	Init	void	This function initializes the state upon switching to the state.
void	Update	void	This function handles time based update mechanics.
void	Exit	void	This function handles the shutdown of the state before a new state is activated.

SUB-MODULES

CAttractState: public CGameState

Description: This module plays a movie in the menu background after a period of user inactivity. The idea is to get the player intrigued by the demo of gameplay, and thus play the game.

Stipulations: This menu state will be called after a period of inactivity in the menu. The Init() function must be called to properly set up the background mesh and the animated texture. Exit() must be called to free the mesh and to call the animated texture's Unload() function.

Interface: This is a state within the menu system state machine. All necessary interaction will be handled internally by that system.

MEMBERS

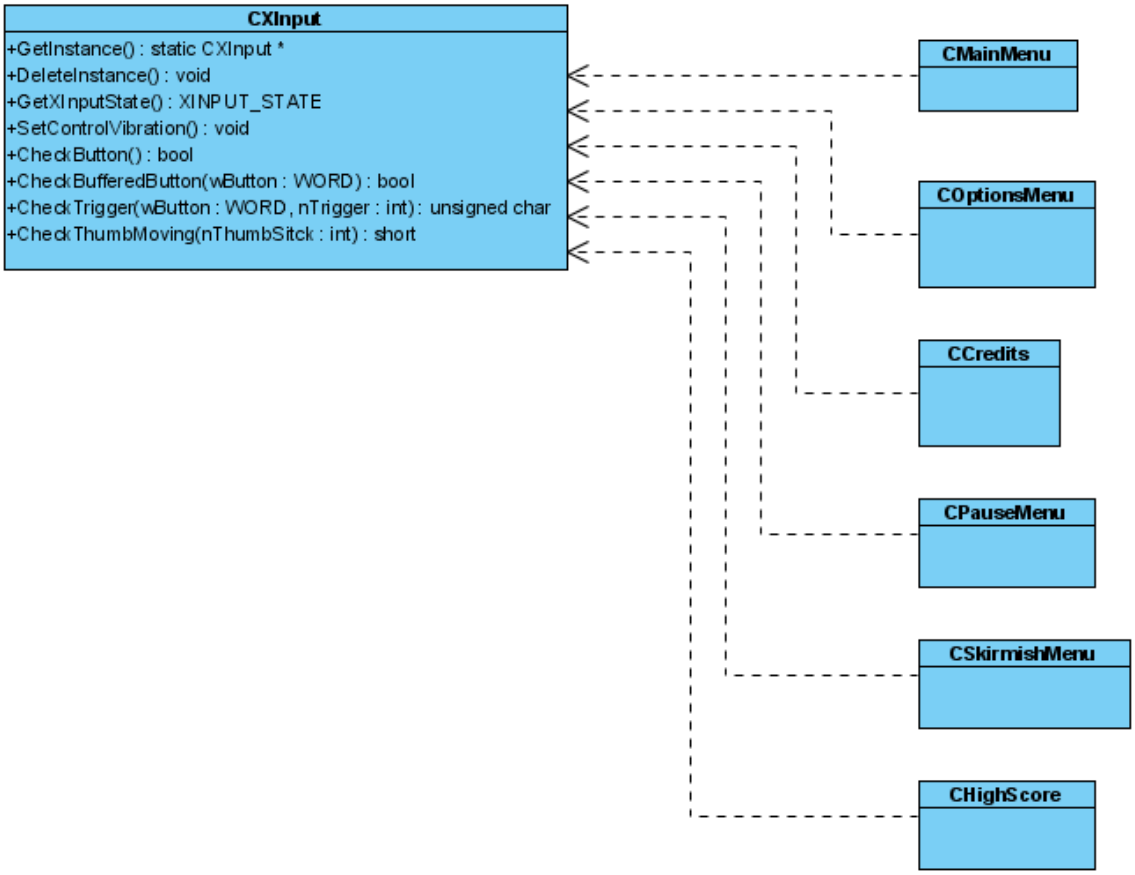
Name	Type	Description
m_pAnimTexture	CAnimatedTexture*	The animated texture on which the movie is played
m_pBackgroundMesh	LPD3DXMESH	The mesh on which the texture will be rendered to
m_xisControllerState	XINPUT_STATE	Stores the current controller state.

METHODS

Return	Name	Type/Parameters	Description
bool	Init	void	Initializes the animation
void	Update	void	Updates the animated texture
void	Exit	void	Deallocates any memory and unloads the animated texture
void	Render	void	Renders the movie

SUB-MODULES

Game State System and Input Wrapper



CBase3D: public IBaseInterface

Description: This is this abstract class interface for all the main objects in the game. It handles collisions and has the code to manipulate the objects position in 3D space. Objects that derive from this are CStatic, CParticle, CMovingEntity.

Stipulations: The CBase3D is an abstract base class and can not be instantiated by itself. You can only create objects that derive from this class.

Interface: This module provides the basic interface for all the objects. Other then providing a basic interface for the inherited objects, this class will be used as the generic parent type for function calls that need to act on it's children. It invokes the math functionality of DirectX to do the translations and rotations.

MEMBERS

Name	Type	Description
m_matOrientation	D3DXMATRIX	4x4 matrix that holds the position and orientation of the object in the world.
m_ObType	enum	This enumeration holds values of the different types of objects that derive from CBase3D. Used when checking the objects against each other with detections.
m_nType	Short	The type of the object
m_fMass	float	The mass of the object used for computing "semi-realistic" physics.
m_vecVelocity	D3DVECTOR3	The velocity of the object
m_vecAcceleration	D3DVECTOR3	The acceleration on the object
m_vecForce	D3DVECTOR3	The net force acting on the object.
m_fRadius	Float[]	collision sphere radius

CBase3D METHODS

Return	Name	Parameters	Description
D3DXMATRIX	GetOrientation	void	Gets the object's orientation matrix
void	SetOrientation	D3DXMATRIX	Sets the object in the world with set orientation and position
void	TransX	Float distance	Translates the object in the direction of its Right vector.
void	TransY	Float distance	Translates the object in the direction of its Up vector.
void	TransZ	Float distance	Translates the object in the direction of its Look vector.
void	Trans	D3DVECTOR3	Translates the object in the direction of a vector in the objects local world.(needed after doing physics based on force and mass)
void	RotateX	Float angle	Rotates the object around its Right vector
void	RotateY	Float angle	Rotates the object around its Up vector
void	RotateZ	Float angle	Rotates the object around its Look vector
void	Update	void	Updates the object according to the elapsed time
void	Render	void	overwritten to render whatever the given object is(mesh, particle, billboard)
bool	Check Collision	CBase3D*	Checks the collision of this object and the passed in one.

SUB-MODULES

CStatic: public CBase3D, public IListener

Description: These objects typically are not moving in the game world although they might. They are mainly obstacles the player will have to maneuver around. They are visually represented by either a billboard or mesh.

Interface: CStatics react to collisions from other objects in the world. They can handle events using the Events system.

MEMBERS

Name	Type	Description
m_nMesh	Unsigned int	Index based on DXwrapper

METHODS

Return	Name	Parameters	Description
void	Update	void	Updates the object according to the elapsed time
void	Render	void	overwritten to render whatever the given object is(mesh, particle, billboard)
bool	Check Collision	CBase3D*	Checks the collision of this object and the passed in one.
Void	HandleEvent	CEvent* pEvent	Handles events that involve this object
Void	InitMesh	Void	Initializing the mesh

CParticle

Description: Contain the information about an individual partices lifetime such as color, scale, position changes.

Interface: Initializes with information read in with files that are created with the particle editor.

MEMBERS

Name	Type	Description
m_vecPos	D3DXVECTOR3	The position the particle starts at relative to the Emmision point contained by the Orientaion matrix it gets by deriving from CBase3D
m_vecVel	D3DXVECTOR3	The velocity of this particle
m_fAge	Float	How old the particle is in its lifetime
m_Emitter	CEmitter*	The emitter that corresponds to this particle

CEmitterTemplate

Description: One of these exists for each different type of particle effect in the game. It holds the data that emitters will use throughout the game.

Interface: read in a file that can be edited by the Particle Editor.

Name	Type	Description
m_nImageID	Int	Number corresponding to the image used for the particle in the texture manager.
m_fMedianAge	Float	The average age a particle can be.
m_fAgeVariance	Float	The range in age a particle can be.
m_nSourceBlend	Int	interpolation
m_nDestinationBlend	Int	interpolation
m_chStartColor	UChar[4]	The color a the particles of this type of emitter will start at.
m_chEndColor	UChar[4]	The color a the particles of this type of emitter will end at.
m_fStartScale	Float	The scale the particles of this type of emitter will start at.
m_fEndScale	Float	The scale the particles of this type of emitter will end at.
M_bisRadial	Bool	Is the emitter radial or not?
M_bisLooping	Bool	Is the smitter looping or not?
M_fspeed	Float	Speed of the particles in this type of emitter.
M_vecDir	D3DXVECTOR3	The direction of the particles in this type of emitter.
m_fSpawnRate	Float	The rate atwhich the particles are created.

METHODS

Return	Name	Paramters	Description
Void	LoadEmitterTemplate	Void	Fills in this type of emitters data by reading in a file that can be edited with the particle editor.

CEmitter: public CBase3D, public IListener

Description: Object controls the emission of given particles contains all the same information as a particle so it can fill out each individual particle with that information.

Interface: Creates the particles specific to this emission

MEMBERS

Name	Type	Description
m_vParticles	Vector<CParticle>	List of all the living particles.
m_vecPos	D3DXVECTOR3	The position of the emitter.
m_EmitterTemplate	CEmitterTemplate*	The Emitter Template that corresponds to this emitter

METHODS

Return	Name	Paramters	Description
Bool	CheckCollision	CBase3D*	Checks the collision based on the particle emissions position and the passed in CBases collision sphere. This is mainly for the laser, because the position of the emitter will be where it collides
void	CreateParticle	void	Creates a CParticle and pushes it into the vector of all particles for this emission
Void	Update	Void	Updates the particles in the emission during their lifetimes
Void	Render	Void	Renders the particle emission to the screen.
Void	HandleEvent	CEvent* pEvent	Handles events that involve this object

SUB-MODULES

CMovingEntity: public CBase3D

Description: These object are the primary moving objects in the game, including the Player, Enemies, and weapons. This class serves primarily as a template for the aformentioned objects.

Interface: This class defines more features required by the moving objects of the game. This class will not be used directly by outside code.

MEMBERS

Name	Type	Description
m_nMesh	Unsigned int	Index based on DXwrapper
m_nMaxHealth	Short	The maximum health of the object
m_nCurHealth	Short	The Current health of the object
m_bState	bool	Current state of the object(dead or alive)
m_dwLastShot	double	The timer governing the rate this object can fire a weapon
m_dwLastSensor	double	The timer governing the rate this object can fire a sensor
m_fShotRate	float	The rate the object fires at
m_fSensorRate	Float	The rate the object can drop sensors
m_fMaxSpeed	float	The max speed this object can travel through the world at
m_pEmitter	**CEmitter** *	The emitter pertaining to this object(trails behind enemies and players and possibly emissions for the engines of the missiles)

METHODS

Void	Update	Void	Updates the objects based on the time pased
Void	Render	Void	Renders the objects model or partical to the screen
bool	Check Collision	CBase3D*	Checks the collision of this object and the passed in one.
Void	HandleEvent	CEvent* pEvent	Handles events that involve this object

SUB-MODULES

CWeapon: public CMovingEntity, public IListener

Description: The different types of weapons that the players and enemies use in the game. These objects have different properties based on the value of an enumeration and a file that can be edited.

Interface: The weapons are editable by the object editor. There are ship and enemy versions alike but are separated for collision reasons.

MEMBERS

Name	Type	Description
m_eWeaponTypes	Enum	There are 2 versions of weapon with 3 for the ship weapons and 3 for the enemies
m_nWeaponType	Short	Current type the weapon is
m_nMissileDmg	Short	The dmg done by missile weapons
m_nLaserDmg	Short	The dmg done by laser weapons
m_nMineDmg	Short	The dmg done by mineweapons

METHODS

Return	Name	Parameters	Description
Void	Update	Void	Updates the objects based on the time pased
Void	Render	Void	Renders the objects model or partical to the screen
bool	Check Collisions	CBase3D*	Checks the collision of this object and the passed in one.
Void	HandleEvent	CEvent* pEvent	Handles events that involve this object
void	InitWeapon	Void	initializes the weapon based on the types and reads in the appropriate file that has the information about the weapon.

SUB-MODULES

CPlayer: public CMovingEntity, public IListener

Description: these objects control the players ship and get input in the update function. The player is able to choose 3 different ways to play (the differnet ships are heavy, medium, and light with different stats)

Interface: depending on what ship the player is going to play as, this object will initialize based on the information read in at the beginging of the game.

MEMBERS

Name	Type	Description
m_nShipType	Short	The ships type
m_eShipTypes	enum	Enumeration containing values for the different type of ships Player(Light,Medium,Heavy) and Enemy(Light, Medium, Heavy)
m_nHealth	Short	The amount of health the player has.
m_nLives	Short	How many lives are left.
m_nShipType	Short	The type of ship the player is using
m_eShipTypes	Enum	Enumeration holding values of the different types of ships
m_nMaxMissile	Short	The max number of missiles that the player can have based on the type of ship he has
m_nMaxMines	Short	The max number of mines the player can have based on the type of ship he has
m_nNumMissile	Short	The current number of missiles that the player has
m_nNumMines	Short	The current number of mines that the player has

METHODS

Return	Name	Paramerters	Description
Void	Update	Void	Updates the objects based on the time pased
Void	Render	Void	Renders the objects model to the screen
bool	Check Collisions	CBase3D*	Checks the collision of this object and the passed in one.
Void	HandleEvent	Cevent* pEvent	Handles events that involve this object
void	InitShip	Void	initializes the ship based on its type (heavy,medium,light) and reads in the apropriate file for that ship type.

CEntityBaseState

Description: This is the enemy AI state base class. It contains no information. Its purpose is to provide a basic template for all state modules.

Stipulations: This module is an abstract base of the entity state system. Each enemy AI state will inherit its functionality from this module. It is a templated module so that it may be used by the enemies as well as potentially for players if neccesary.

Interface: The functions in this class are virtual functions that must be redefined by child classes.

METHODS

Return	Name	Type/Parameters	Description
bool	Init	T*	This function initializes the state upon switching to the state.
Void	Update	T*	This function handles time based update mechanics.
Void	Exit	T*	This function handles the shutdown of the state before a new state is activated.

CAIHuntingState: public CEntityBaseState

Description: This is the game play state class. It contains information related only to game play. It will initialize each level prior to the beginning of play and will process gameplay inputs.

Stipulations: This module is a game state system inheriting from CGameClass. It is a singleton.

MEMBERS

Name	Type	Description
m_pInstance	CAIHuntingState*	Static pointer to Hunting State singleton object.

METHODS

Return	Name	Type/Parameters	Description
CAIHuntingState*	GetInstance	void	Creates the first and only instance of this class and returns its memory address.
Void	DeleteInstanceblaaaaaarghhh	void	Deletes the instance of this class.
Bool	Init	Cenemy*	This function initializes the state upon switching to the state.
Void	Update	Cenemy*	This function handles time based update mechanics.
Void	Exit	Cenemy*	This function handles the shutdown of the state before a new state is activated.

SUB-MODULES

CAITrappingState: public CEntityBaseState

Description: This is the game play state class. It contains information related only to game play. It will initialize each level prior to the beginning of play and will process gameplay inputs.

Stipulations: This module is a game state system inheriting from CGameClass. It is a singleton.

MEMBERS

Name	Type	Description
m_pInstance	CAITrappingState*	Static pointer to Trapping State singleton object.

METHODS

Return	Name	Type/Parameters	Description
CAITrappingState*	GetInstance	void	Creates the first and only instance of this class and returns its memory address.
Void	DeleteInstance	void	Deletes the instance of this class.
Bool	Init	void	This function initializes the state upon switching to the state.
Void	Update	void	This function handles time based update mechanics.
Void	Exit	void	This function handles the shutdown of the state before a new state is activated.

SUB-MODULES

CAIWanderingState: public CEntityBaseState

Description: This is the game play state class. It contains information related only to game play. It will initialize each level prior to the beginning of play and will process gameplay inputs.

Stipulations: This module is a game state system inheriting from CGameClass. It is a singleton.

MEMBERS

Name	Type	Description
m_pInstance	CAIWanderingState *	Static pointer to State singleton object.

METHODS

Return	Name	Type/Parameters	Description
CAIWanderingState *	GetInstance	void	Creates the first and only instance of this class and returns its memory address.
Void	DeleteInstance	void	Deletes the instance of this class.
Bool	Init	CEnemy*	This function initializes the state upon switching to the state.
Void	Update	Cenemy*	This function handles time based update mechanics.
Void	Exit	Cenemy*	This function handles the shutdown of the state before a new state is activated.

SUB-MODULES

CEnemy: public CmovingEntity, public Ilistener

Description: These objects are computer enemies with AI behavioral mechanics. The type of ship used is determined by the current level and difficulty settings.

Interface: This object will read in appropriate stats based on the type of enemy ship.

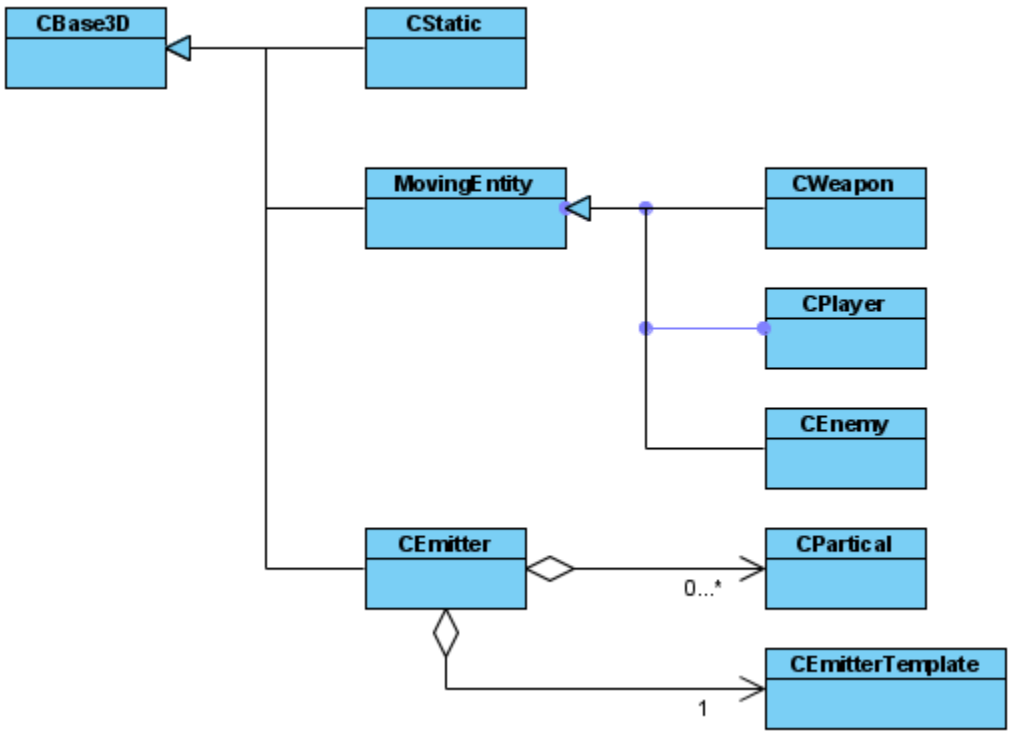
MEMBERS

Name	Type	Description
m_pAIState	CAIBaseState*	Pointer to current AI state
M_nShipType	short	The type of ship the enemy is using
m_eShipTypes	enum	Enumeration holding values of the different types of ships
m_nMaxMissile	short	The max number of missiles that the enemy can have based on the type of ship he has
m_nMaxMines	short	The max number of mines the enemy can have based on the type of ship he has
m_nNumMissile	short	The current number of missiles that the enemy has
m_nNumMines	short	The current number of mines that the enemy has

METHODS

Return	Name	Type/Parameters	Description
void	Update	void	Updates the objects based on the time passed
void	Render	void	Renders the objects model to the screen
bool	Check Collision	CBase3D* pBase3D	Checks the collision of this object and the passed in one.
void	HandleEvent	CEvent* pEvent	Handles events that involve this object
void	InitShip	void	initializes the ship based on its type (heavy,medium,light) and reads in the apropriate file for that ship type.

Object Module Hierarchy



CGameClass

Description: This module is the primary data containment and management component of this game. This class is responsible for initializing and maintaining all of the singleton wrapper classes.

Stipulation: This module is a singleton class, it is the primary communications point between the main windows framework and the game modules.

Interface: This object will handle loading of initial memory items as well as changing between menu and play states.

MEMBERS

Name	Type	Description
m_pCurrentState	CGameState*	Pointer to current Game State
m_pD3D	CDirect3D*	Pointer to Direct 3D object.
m_pDI	CSGD_DirectInput*	Pointer to DirectInput wrapper object
m_pDS	CSGD_DirectSound*	Pointer to DirectSound wrapper object.
m_pTM	CSGD_TextureManager*	Pointer to TextureManager wrapper object.
m_pWM	CSGD_WaveManager*	Pointer to WaveManager wrapper object.
m_pOF	CSGD_ObjectFactory<string, CBase3D>*	Pointer to ObjectFactory wrapper object.
m_pOM	CSGD_ObjectManager*	Pointer to ObjectManager wrapper object.
m_pDis	CSGD_Dispatcher*	Pointer to Event Dispatcher wrapper object.
m_pMS	CSGD_MessageSystem*	Pointer to Message System wrapper object.
m_nScreenHeight	int	Local storage for current screen height.
m_nScreenWidth	int	Local storage for current screen width.
m_bWindowed	bool	Local storage for current windowed mode.
m_pShip	CPlayer*	Pointer to current player ship.
m_pInstance	CGameClass*	Static pointer to GameClass singleton object.

CGameClass METHODS

Return	Name	Type/Parameters	Description
CGameClass*	GetInstance	void	Creates the first and only instance of this class and returns its memory address.
void	DeleteInstance	void	Deletes the instance of this class.
bool	GameInit	HINSTANCE hInstance, HWND hWnd, int nWidth, int nHeight, bool bWindowed, bool bVsync	Function to initialize the gameclass object and neccesary wrapper objects prior to main game loop.
bool	GameMain	void	Main game loop functionality for time based actions.
bool	GameShutdown	void	Clean up memory contained within the gameclass object and shutdown all wrapper instances.
private void	ChangeState	CGameState* newState	Change game states between menus and play state.
void	GameMessageProc	CBaseMessage* pMsg	Handle all messages internal to the game.

SUB-MODULES