

CHALMERS

# Sim-engines group 7

---

Group Report

## 1 Conclusion

As the result of the simulation engine project, as a team, we accomplished to built a game engine that each member contributed succesfully by adding their own extension and integrating them to work together. We built our own FPS game by making use of our game engine. In spite of the fact that there were not much time to first finishing the game engine completely and building a game using it, we tended to integrate our engine – framework and each extension towards the game we wanted to create as an end result. Finally we accomplished to built a small map to run our game, with the models, graphics, AI, physics and multimedia extensions working together.

## 2 Introduction

### 2.1 Purpose

The task of the project was to study an existing Open Source simulation engine, integrate necessary external components, extend it with meaningful modifications and to implement a simple tech demo/game to demonstrade the improved engine.

#### 2.1.1 OGRE

Ogre is the simulation engine that we extended. OGRE (Object-Oriented Graphics Rendering Engine) is a scene-oriented, flexible 3D rendering engine written in C++ designed to make it easier and intuitive for developers to produce applications utilising hardware-accelerated 3D graphics. As its name states, OGRE is "just" a rendering engine. As such, its main purpose is to provide a general solution for graphics rendering. It is not an all-in-one solution in terms of game development or simulation as it doesn't provide audio or physics support, for instance.

The choice of OGRE as a graphics engine allows developers the freedom to use whatever physics, input, audio and other libraries they want and allows the OGRE development team to focus on graphics rather than distribute their efforts amongst several systems. OGRE explicitly supports the OIS, SDL and CEGUI libraries, and includes the Cg toolkit.

#### 2.1.2 Extending OGRE

Taking advantage of OGRE providing freedom to use physics – audio – AI and multimedia extentions, we planned to make fully functional game engine by implementing these features in a limited amount of time. Also simultaneously adapting our engine into a game concept by adding several features on the run.

## 2.2 Goals

### Graphics

- The basic idea is to create a programmable GPU shader that is capable of blurring the reflections generated by an environment map.
- Use GLSL as the language of implementation.

### AI

- a 2D path-finding engine which was implemented by OpenSteer.
- The decision-making was implemented in the Fuzzy logic way.
- Goal

- multi-chase
- avoid obstacles
- hide
- path-finding

## Physics

- Create 2D destructible objects
- Generate cracking patterns in real time using a Voronoi Diagram.
- Be as realistic as possible

## Multimedia

- OpenAL, OgreAL 3D sound
- Interaction with camera (Face tracking)
  - Optimize face tracking algorithm
  - Create new thread for only this extension(Boost)
  - move user face to left/right to interact with game
  - Replace user's face with smiley

## 3 Pre-study

### 3.1 How does the system look today?

We use some basic interaction such as mouse and keyboard.

### 3.2 What did you want to make better?

We want to add more interaction to any game engine, so we create extension to detect object for control the game.

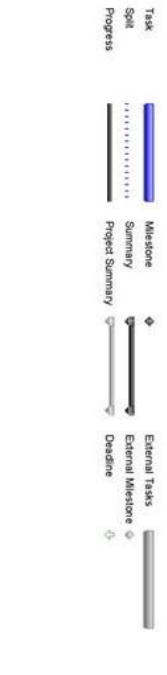
### 3.3 Project plan and division of themes

ID	Task Name	Duration	Start	Finish	Pred
1	<b>Software Engineering [Yigit]</b>	<b>30 days?</b>	<b>Mon 10/26/09</b>	<b>Fri 12/4/09</b>	
2	Create working convention	1 day?	Fri 10/30/09	Fri 10/30/09	
3	Functional and non functional requirement	5 days?	Mon 10/26/09	Fri 10/30/09	
4	Design Framework (Game States and should be MVC) output may be class diagram of framework	7 days?	Mon 10/26/09	Tue 11/2/09	
5	Design Class diagram of each extensions	6 days?	Wed 11/4/09	Wed 11/11/09	
6	Design Interaction between extensions	5 days?	Mon 10/26/09	Fri 10/30/09	
7	Make schedule for everyone	2 days?	Mon 11/2/09	Tue 11/3/09	
8	Implement Framework	13 days?	Thu 11/12/09	Mon 11/30/09	
9	Merge all extension	4 days?	Tue 12/1/09	Fri 12/4/09	
10	Test	2 days?	Tue 12/1/09	Wed 12/2/09	
11	<b>Multimedia Extension [Akekarat]</b>	<b>26 days?</b>	<b>Mon 10/26/09</b>	<b>Mon 11/30/09</b>	
12	face tracking on OpenCV	4 days?	Mon 10/26/09	Thu 10/29/09	
13	merge OpenCV with Ogre3D to test	4 days?	Mon 11/2/09	Thu 11/5/09 12	
14	create Thread for OpenCV	3 days?	Mon 11/9/09	Wed 11/11/09 13	
15	determine the output by using many frames	4 days?	Thu 11/12/09	Tue 11/17/09 14	
16	send each frame image to Ogre3D	2.5 days?	Wed 11/18/09	Fri 11/20/09 15	
17	render each frame on Ogre3D	2 days?	Fri 11/20/09	Tue 11/24/09 16	
18	document everything	4.5 days?	Tue 11/24/09	Mon 11/30/09 17	
19	<b>Graphics Extension [George]</b>	<b>26 days?</b>	<b>Mon 10/26/09</b>	<b>Mon 11/30/09</b>	
20	write a vertex shader, fragment shader of environment maps and blurry reflections	7.5 days?	Mon 10/26/09	Wed 11/4/09	
21	edit material script for environment maps and blurry reflections	5.5 days?	Wed 11/4/09	Wed 11/11/09 20	
22	apply either a mipmap biasing operation or gaussian blur algorithm to the fragment shader	7 days?	Thu 11/12/09	Fri 11/20/09 21	
23	document the work	5 days?	Mon 11/23/09	Mon 11/30/09 22	
24	<b>AI Extension [Sicheng]</b>	<b>26.5 days?</b>	<b>Mon 10/26/09</b>	<b>Tue 12/1/09</b>	
25	use the FFL to determine the patrol/hide states based on the HP of enemies;	6 days?	Mon 10/26/09	Mon 11/2/09	
26	adopt the FFL to decide whether to attack or detect in the patrol mode, based on the distance to the main character;	6 days?	Tue 11/3/09	Tue 11/10/09 25	
27	In the target detection state (detect), the realization will be done by the chase/tag plugin of OpenSteer;	5 days?	Wed 11/11/09	Tue 11/17/09 26	
28	In the hide state, change the code of chase/tag plugin of OpenSteer to find obstacles and hide;	4 days?	Wed 11/18/09	Mon 11/23/09 27	
29	document.	5.5 days?	Tue 11/24/09	Tue 12/1/09 28	
30	<b>Physics Extension [Henrique]</b>	<b>26 days?</b>	<b>Mon 10/26/09</b>	<b>Mon 11/30/09</b>	
31	implement basic cracking pattern algorithm	6 days?	Mon 10/26/09	Mon 11/2/09	
32	integrate the algorithm with Ogre3D	4.5 days?	Tue 11/3/09	Mon 11/9/09 31	
33	implement division of the polygons according to pattern	5.5 days?	Mon 11/9/09	Mon 11/16/09 32	
34	generate different patterns for different materials	4 days?	Tue 11/17/09	Fri 11/20/09 33	
35	apply forces to the fragments	4.5 days?	Mon 11/23/09	Fri 11/27/09 34	
36	document	1.5 days?	Fri 11/27/09	Mon 11/30/09 35	
37	<b>Miscellaneous [All members]</b>	<b>13 days?</b>	<b>Tue 12/1/09</b>	<b>Thu 12/17/09</b>	
38	create MAYA map model and import to Ogre [George]	4 days?	Tue 12/1/09	Fri 12/4/09	
39	create MAYA character model + Animation [George]	5 days?	Mon 12/7/09	Fri 12/11/09 38	
40	create MAYA object model [George]	4 days?	Mon 12/14/09	Thu 12/17/09 39	
41	convert MAYA map to OpenSteer map [Simon]	4 days?	Tue 12/1/09	Fri 12/4/09	
42	all user interface HP, Bullets and esse [Simon]	9 days?	Mon 12/7/09	Thu 12/17/09 41	
43	3D Sound part (OpenAL) [None]	1 day?	Mon 12/14/09	Mon 12/14/09	
44	<b>Game play part</b>	<b>13 days?</b>	<b>Tue 12/1/09</b>	<b>Thu 12/17/09</b>	
45	character walking [Yigit]	2 days?	Tue 12/1/09	Wed 12/2/09	
46	control character [Banana]	2 days?	Thu 12/3/09	Fri 12/4/09	
47	dodge left/right [Banana]	2 days?	Mon 12/7/09	Tue 12/8/09	
48	character shoot gun [Banana]	3 days?	Wed 12/9/09	Fri 12/11/09	
49	check contact of bullet and object and then object move or damaged [Henrique]	3 days?	Tue 12/15/09	Thu 12/17/09	
50	attach gun to camera and move it	6 days?	Tue 12/1/09	Tue 12/8/09	
51	win/lose [Yigit]	7 days?	Wed 12/9/09	Thu 12/17/09 50	
52	Presentation	3 days?	Tue 12/15/09	Thu 12/17/09	

ID	Task Name	Duration	Start	Finish	Prede
1	Software Engineering [Vigil]	30 days?	Mon 10/26/09	Fri 12/4/09	
2	Create working convention	1 day?	Fri 10/30/09	Fri 10/30/09	
3	Functional and non functional requirement	5 days?	Mon 10/26/09	Mon 10/26/09	
4	Design Framework (Game Shires and should be MVC) output may be class diagram of framework	7 days?	Mon 10/26/09	Tue 11/2/09	
5	Design Class Diagram of each extensions	6 days?	Wed 11/4/09	Wed 11/11/09	
6	Design Interaction between extensions	5 days?	Mon 10/26/09	Fri 10/30/09	
7	Make schedule for everyone	2 days?	Mon 11/2/09	Tue 11/2/09	
8	Implement Framework	13 days?	Thu 11/12/09	Mon 11/20/09	
9	Merge all extension	4 days?	Tue 12/1/09	Fri 12/4/09	
10	Test	2 days?	Wed 12/2/09	Thu 12/2/09	
11	Multimedia Extension [Akkurat]	26 days?	Mon 10/26/09	Mon 11/09/09	
12	face tracking on OpenCV	4 days?	Mon 10/26/09	Thu 10/29/09	
13	merge OpenCV with Open3D to test	4 days?	Mon 11/2/09	Thu 11/5/09 12	
14	create Thread for OpenCV	3 days?	Mon 11/9/09	Wed 11/11/09 13	
15	determine the output by using many frames	4 days?	Thu 11/12/09	Thu 11/17/09 14	
16	send each frame image to Open3D	2.5 days?	Wed 11/18/09	Fri 11/20/09 15	
17	render each frame on Open3D	2 days?	Fri 11/20/09	Tue 11/24/09 16	
18	document everything	4.5 days?	Tue 11/24/09	Mon 11/30/09 17	
19	Graphics Extension [George]	26 days?	Mon 10/26/09	Wed 11/4/09	
20	write a vertex shader, fragment shader of environment maps and blurry reflections	7.5 days?	Mon 10/26/09	Wed 11/11/09 20	
21	edit material script for environment maps and blurry reflections	5.5 days?	Wed 11/4/09	Fri 11/20/09 21	
22	apply either a mipmap biasing operation or gaussian blur algorithm to the fragment shader	7 days?	Thu 11/12/09	Mon 11/20/09 22	
23	document the work	6 days?	Mon 11/23/09	Mon 11/30/09 22	
24	AI Extension [Siching]	26.5 days?	Mon 10/26/09	Tue 12/1/09	
25	use the FLL to determine the patrol/hide states based on the hp of enemies.	6 days?	Mon 10/26/09	Mon 11/2/09	
26	adapt the FLL to decide whether to attack or detect in the patrol mode, based on the distance to the main character.	6 days?	Tue 11/2/09	Tue 11/10/09 25	
27	In the target detection state (detect), the realization will be done by the chasing plugin of OpenSteer.	5 days?	Wed 11/11/09	Tue 11/17/09 26	
28	In the hide state, change the code of chasing plugin of OpenSteer to find obstacles and hide.	4 days?	Wed 11/18/09	Mon 11/23/09 27	
29	document	5.5 days?	Tue 11/24/09	Tue 12/1/09 28	
30	Physics Extension [Henrique]	26 days?	Mon 10/26/09	Mon 11/09/09	
31	implement basic culling pattern algorithm	6 days?	Mon 10/26/09	Mon 11/2/09	
32	integrate the algorithm with Open3D	4.5 days?	Tue 11/2/09	Mon 11/9/09 31	
33	implement division of the polygons according to pattern	5.5 days?	Mon 11/9/09	Mon 11/16/09 32	
34	generate different patterns for different materials	4 days?	Tue 11/17/09	Fri 11/20/09 33	
35	apply forces to the fragments	4.5 days?	Mon 11/23/09	Fri 11/27/09 34	
36	document	1.5 days?	Fri 11/27/09	Mon 11/30/09 35	
37	Miscellaneous [All members]	13 days?	Tue 12/1/09	Thu 12/17/09	
38	create MAYA map model and import to Ogr [George]	4 days?	Tue 12/1/09	Fri 12/4/09	
39	create MAYA character model + Animation [George]	5 days?	Mon 12/7/09	Fri 12/11/09 38	
40	create MAYA object model [George]	4 days?	Mon 12/14/09	Thu 12/17/09 39	
41	convert MAYA map to OpenSteer map [Simon]	4 days?	Tue 12/1/09	Fri 12/4/09	
42	all user interface HP, Bullets and aim [Simon]	9 days?	Mon 12/7/09	Thu 12/17/09 41	
43	3D Sound part [OpenAU] [Xione]	1 day?	Mon 12/14/09	Mon 12/14/09	
44	Game data part	13 days?	Tue 12/1/09	Thu 12/17/09	
45	character walking [Vigil]	2 days?	Tue 12/1/09	Wed 12/2/09	
46	control character [Banana]	2 days?	Thu 12/3/09	Fri 12/4/09	
47	doge lelight [Banana]	2 days?	Mon 12/7/09	Tue 12/8/09	
48	character shoot gun [Banana]	3 days?	Wed 12/9/09	Fri 12/11/09	
49	check content of bullet and object and then object move or damaged [Henrique]	3 days?	Tue 12/15/09	Thu 12/17/09	
50	attach gun to camera and move it	6 days?	Tue 12/1/09	Tue 12/8/09	
51	whisper [Vigil]	7 days?	Wed 12/9/09	Thu 12/17/09 50	
52	Presentation	3 days?	Thu 12/15/09	Thu 12/17/09	



Project Schedule  
Date: Sun 11/01/10



## 4 Specification of demands (for each extension)

### 4.1 Functional demands (functionality)

#### 4.1.1 Graphics

- be able to transfer variables from Ogre rendering engine to the Vertex shader.
- be able to transfer variables between the Vertex shader to Fragment shader.
- Do all the Matrices calculations in on space preferably in the World space.
- create either real-time calculations of mipmaps or prebuild them.
- Real-time calculation of reflection ray vector.

#### 4.1.2 AI

- The extension plug-in of Opensteer should be able to synchronise with the game.
- The plug-in provides interfaces for the core of the game to control the player object and receive the movement parameters of enemy objects.
- The enemy objects driven by state-machine could run autonomously.
- Enemy objects could chase the player, attack, hide, and avoid obstacles.
- The path-finding of enemy objects is based on Dijkstra algorithm.

#### 4.1.3 Physics

The physics extension should deliver a solution for generation of simple destructible 2D objects. More specifically, the extension should not only tessellate the fragments, but also generate the crack patterns for it. On top of that, after the division of polygons is done, gravity should bring the broken pieces downwards. Also, the generated fragments should always be different every time an object is broken. Finally, the whole process should be done in real time and without a big impact in the performance of the game.

#### 4.1.4 Multimedia

- detect face of user
- show face of user in real-time
- detect left and right movement of user's face
- replace smileys on user's face
- change smileys face when depend on player's life point

### 4.2 Non-functional demands (properties)(quality requirement)

#### 4.2.1 Graphics

- Do approximation for Fresnel calculation for better efficiency (using Shlick's approximation).
- Allow the shader to work on several video cards (increase compatibility).
- Create a similar environment map that is close to the real 3D world environment.

#### 4.2.2 AI

- The computation and the response of the AI should be fast.
- The enemy will "cheat" , such as know the position of the player without seeing.
- reduce the object collision in the AI layer.

#### 4.2.3 Physics

The shape of fragments generated combined to the ability of setting the material properties of the object should add more realism to the results.

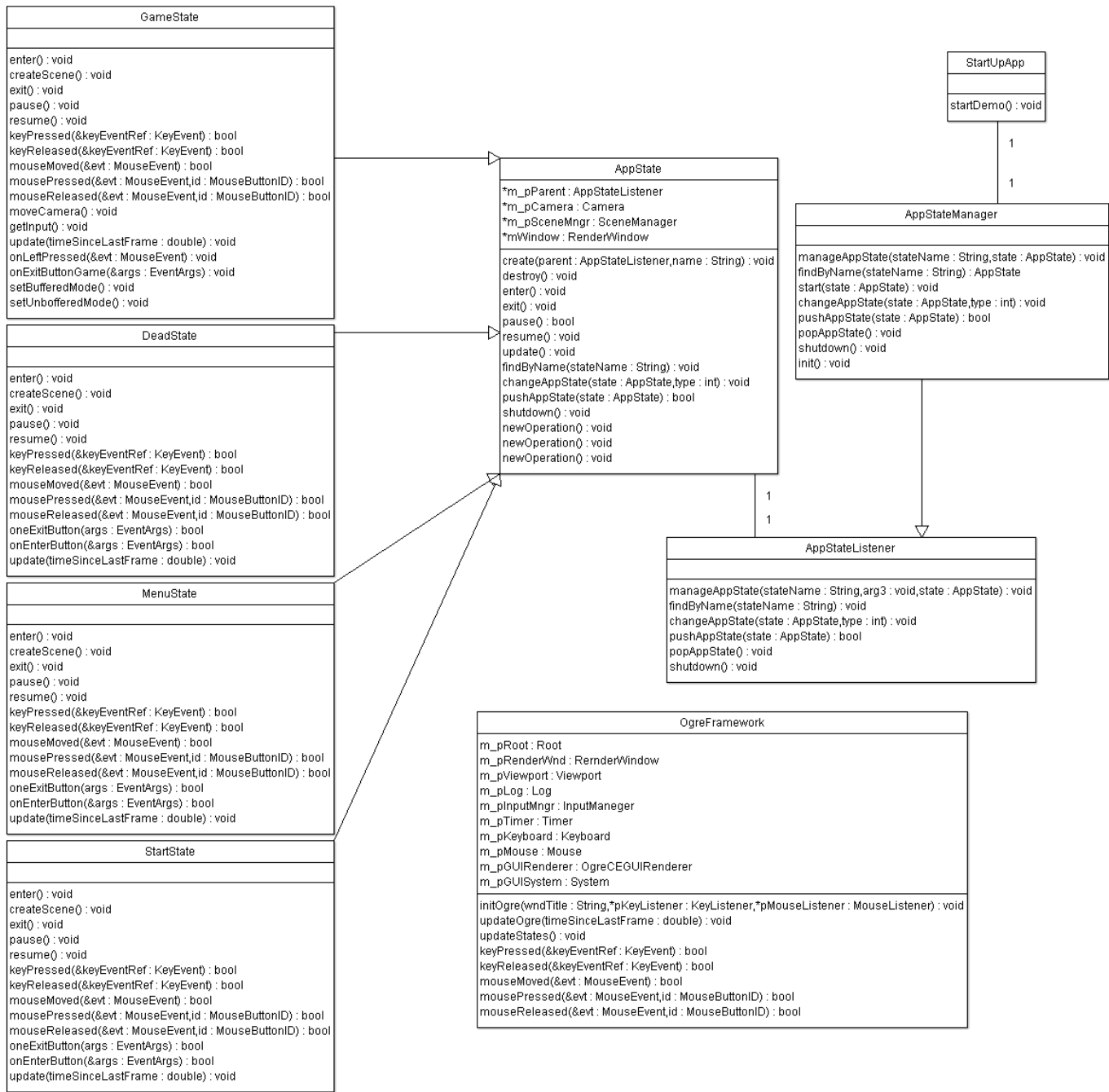
#### 4.2.4 Multimedia

- should be fast
- detect user's face in most cases
- track movement of user's face in most cases

### 5 Analysis

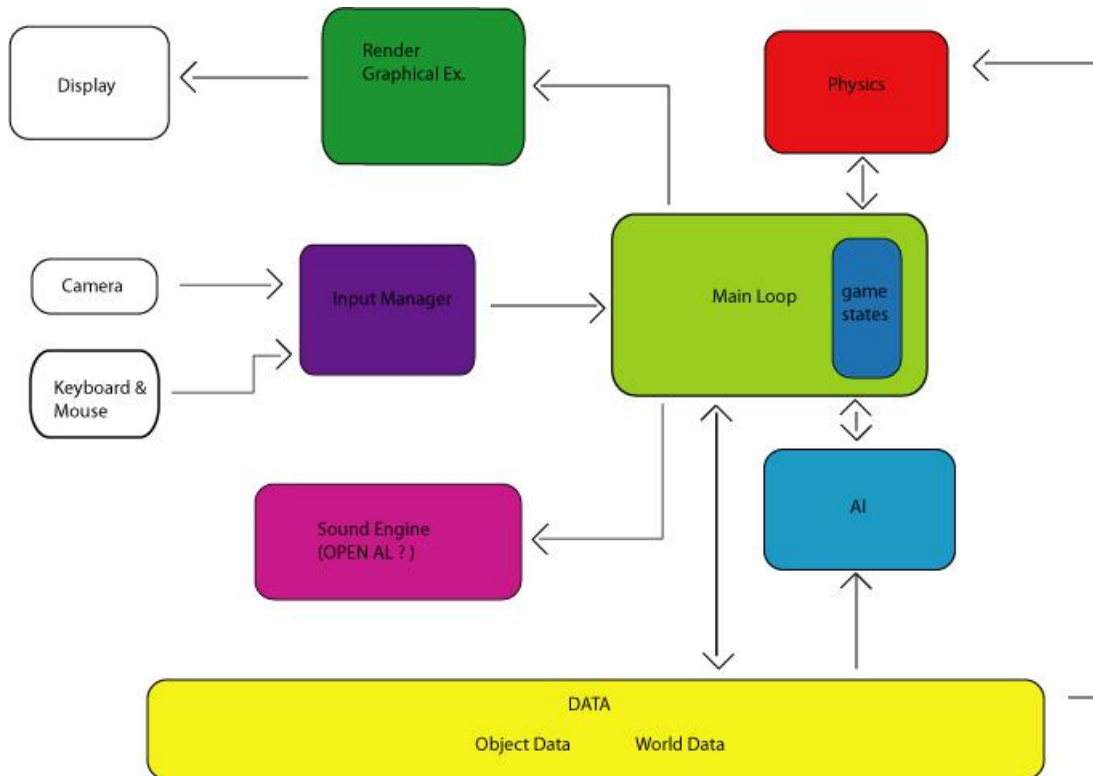
#### 5.1 Main framework

A solid framework is key to succeeding in building a complete game. Having a flexible and solid framework powering your game makes your overall game easier and faster to code, read and understand, and also less error-prone and more stable. It started with deciding upon how the state managers and the main application works together, then the game states and the rest were added. Here is the final UML diagram of the main framework that whole engine was built upon. ( See next page)



## 5.2 Architecture

Every game engine, from the simplest to the most complex, requires some means of updating the internal state of every game object over time. It is an extremely difficult job to build a perfect architecture in this sense. Our aim was to at least try to find a solution to the problem of deciding upon a version.



#### A ) Input Manager:

- finding input devices
- attaching and configuring input devices
- read control values from input devices
- maps physical inputs to game controls

Input Manager was programmed mostly specific to the game itself. Since the genre we have picked was FPS, the camera movements and the navigation was built accordingly.

#### B) Game States

The most crucial part of the main framework was the states of the game. Whole framework was built upon the states. Stack fashion was used to handle the changes between the states. In the beginning of the game there are no elements in the stack and as soon as the game is executed first state is pushed to the stack. Afterwards, depending on the previous state and the next state, whenever a new state is acquired, either the previous one is popped or deleted from the stack or the new state is pushed over the previous state without deleting the last one from the state. It is very important to analyze the states and execute accordingly.

- Start State
- Play State
- Menu State
- Dead State

## 6 Design (for each extension)

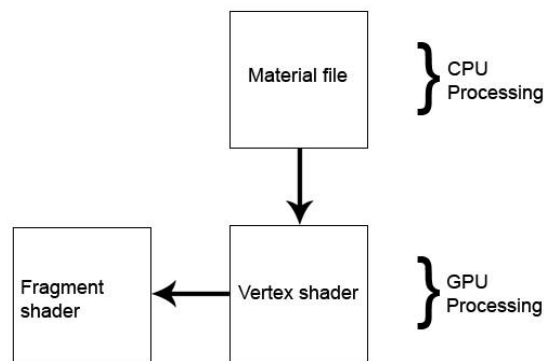
### 6.1 Classes

#### 6.1.1 Graphics

The graphics extension had a simple design approach, as everything was already integrated in Ogre's engine. So there was no need for doing extra effort in integrating or interfacing our framework with the graphics extension.

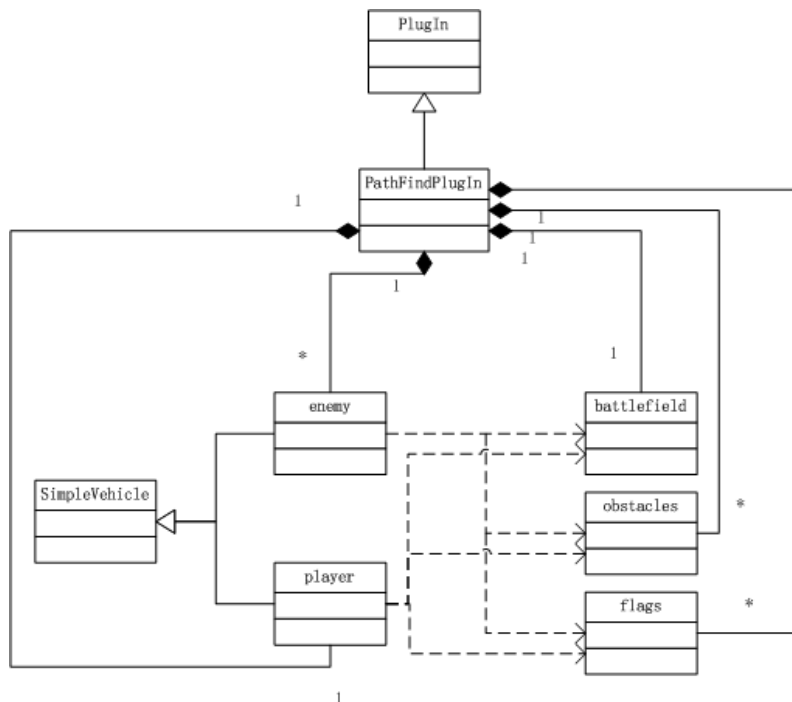
As mentioned before the main task of the extension is to create a programmable GPU shader for blurry reflections, which practically needed creating 3 different pieces of code:

- 1- A modified material file.
- 2- A Vertex shader.
- 3- A fragment shader.



#### 6.1.2 AI

The extension is based on the OpenSteer API, so the AI extension is a plug-in of the OpenSteer.



## Map

In this simulation engine, the map is constructed by the logic map and the visible map. The logic map is defined separately because of using OpenSteer. The map has three main components, such as the battlefield class, obstacles class and flags class.

The battlefield presents the range of this map, and limits the movement of the player, in order to prevent the movement out of range.

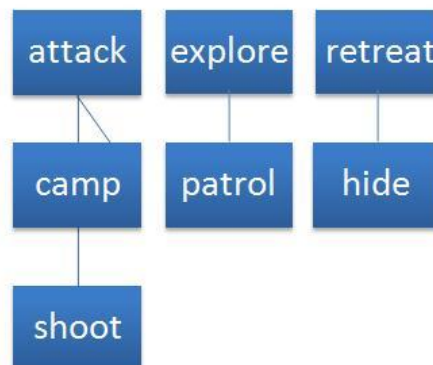
Obstacles on the map can provide partly of the collision detection. Moreover, the obstacles on the map can be detected by the AI-enemies, to make the avoiding and hiding.

Defining the flags on the map is for the path-finding algorithm. The AI-enemies which are in the path-finding mode will find the shortest path to chase the player.

## Enemy

The enemy class inherits the simplevehicle class of OpenSteer library.

- SOAR hierarchy



As the figure shows, the activities of the AI-enemy follow a SOAR hierarchy. The SOAR has three main components, attack, explore and retreat.

The state under explore is patrol. The enemy will patrol around a 'home' position and look around to search for the player object.

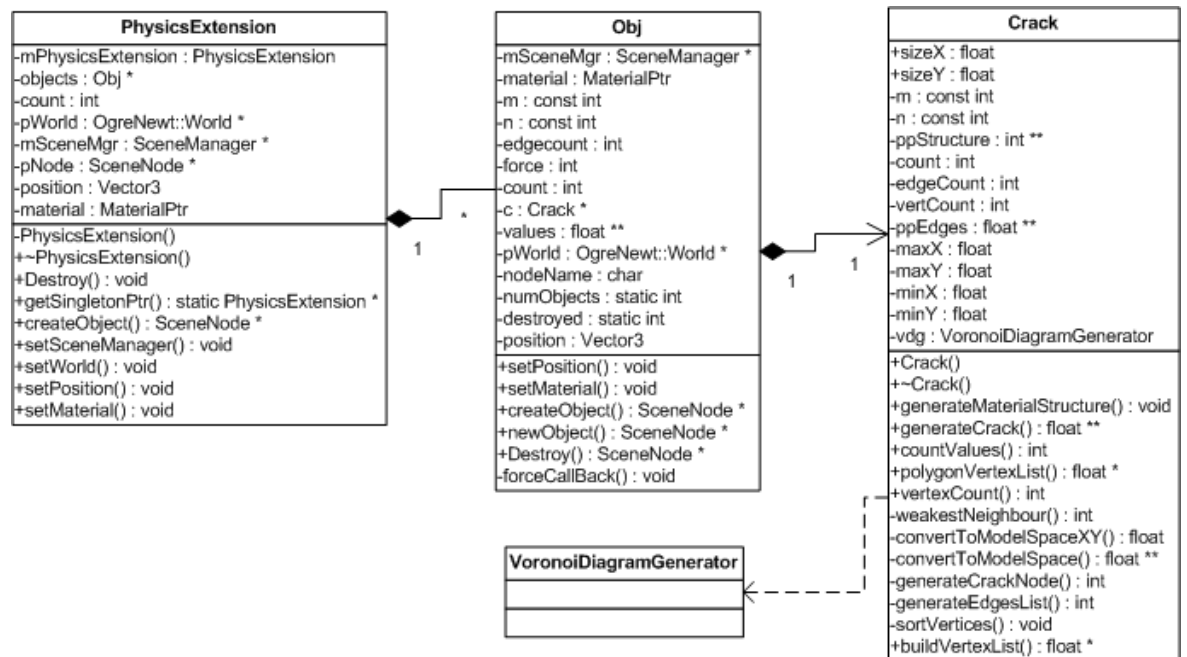
Under the attack category, if the enemy found the player object, it will camp and shoot the player which is in the shooting range. While the player moves out of the shooting range, the enemy will start to chase the player object.

Moreover, the enemy could turn to the hide state if it has too low hit points. As we know, the player will keep shooting the enemy objects in the game, so I designed a 'hide' state to command the enemy to find the nearest obstacle to hide behind. For the level of difficulties, some of the enemies will not have this function.

The path finding state is a special state which is out of the SOAR hierarchy. Imaging a situation that an enemy is under attack and sending a help signal. Then other enemies on the map will know the position and get to the player. For that reason, there should be a path-finding function for the enemy to find the player as soon as possible without aimlessness. The path-finding algorithm is implemented by the Dijkstra algorithm, which asks for an array to define the path info.

The decision-making of the enemy is base on the distance and implemented by the fuzzy-logic. If the player is in the shooting range, the enemy will camp and shoot, otherwise it will chase. The fuzzy-logic is not necessary in a FPS game actually, but it is done.

### 6.1.3 Physics



The physics extension is divided in four basic classes: PhysicsExtension, Obj, Crack and VoronoiDiagramGenerator. The PhysicsExtension is the top level class and is responsible for the handling of the destructible objects in the scene through the use of Obj class instances.

The Obj class handles the low level properties of individual objects such as size, initial position, material as well as the physical forces applied to it. Furthermore, when the object is destroyed, the Obj class generates the fragments through the use of the Crack class.

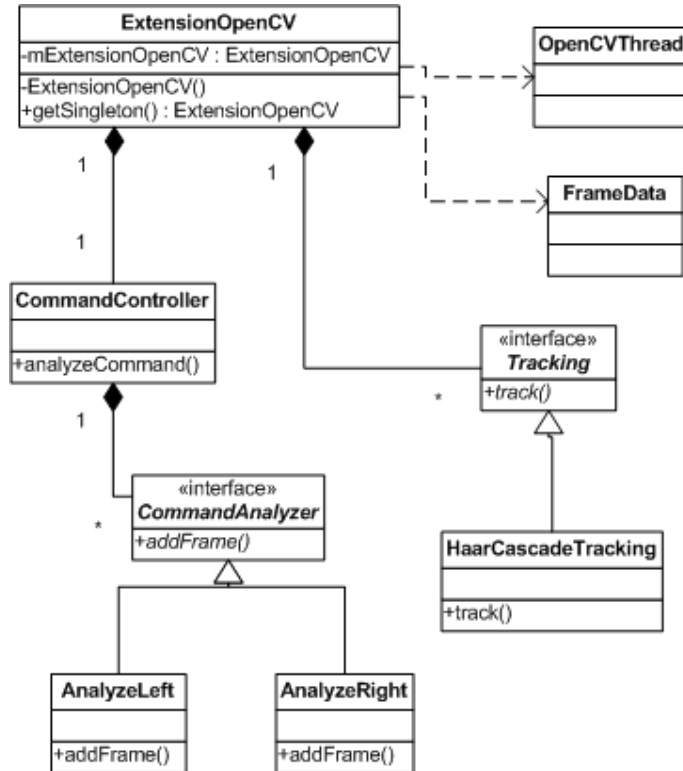
The Crack class is the one responsible for the generation of crack patterns. It makes use of the VoronoiDiagramGenerator class to create the Voronoi Diagram based on the crack pattern. Each of the polygons in the diagram is then made available to the Obj class for tessellation.

The VoronoiDiagramGenerator class, as its name already says, is used for the sole purpose of generating the Voronoi Diagrams.

The API available for the users is limited to the PhysicsExtension class. This means that all properties that can be set for the objects are available as

PhysicsExtension methods and all the other classes are transparent to the user.

#### 6.1.4 Multimedia Extension [banana]



This design is very flexible for many cases. First, user can add another Haar Cascade tracking type by change cascade file. Second, They can add more tracking algorithm easily by extends Tracking interface. Finally they can add more moving pattern of tracking object by extends interface CommandAnalyzer. For example, we use AnalyzeLeft class for detect move left pattern. This design has more advantage, because user can use this extension easily by only ExtensionOpenCV class. I use singleton pattern, so user can use it from any class. We use OpenCVThread class to be new thread that do all work of this extension, so this extension faster than using only one thread. Additionally, this extension don't use Ogre3D as library, so user can use this extension with another game engine. Thus, I send movement data and image of each frame to game engine.

#### 6.2 Interaction

These is interaction of each extension. We choose to use Facade pattern as much as possible, because user can use each extension easier by use only one class.

#### Multimedia Extension

Class ExtensionOpenCV

```

static ExtensionOpenCV& getSingleton()
static ExtensionOpenCV* getSingletonPtr()
int initOpenCV()
int endOpenCV()
void cvLoopThread()
bool isFinish()
  
```

```

unsigned char* getFrameImage()
int getCameraImageWidth()
int getCameraImageHeight()
int getCameraImageDepth()
int getCameraImageSize()
void setFaceType(FaceType faceType)
CommandType getCommand()

```

### Physics Extension

Class PhysicsExtension

```

static PhysicsExtension* getSingletonPtr()
SceneNode *createObject(float *vertexList, int start, int num)
void setSceneManager(SceneManager *sceneMgr)
void setWorld(OgreNewt::World* pW)
void setPosition(Vector3 pos)
void setMaterial(MaterialPtr mat)

```

### AI Extension

Class PathFindPlugIn

```

int createEnemy(Vec3 pos)
Enemy* getEnemy(int identi)
void deleteEnemy(int identi)
void open ()
void update (const float currentTime, const float elapsedTime)
void close ()
const AVGroup& allVehicles ()

```

Class Player

Vec3 getPosition(void)

Vec3 forward()

Class Enemy

```

int getID()
void setState(StateAI s)
StateAI getState()
void setDie(bool isDie)
bool isDie()
Vec3 forward()
Vec3 position()
void setHP(int hp)
int getHP()

```

### Graphics Extension

This extension creates only material and tech demo can use that material.

#### 6.3 Implementation (for each extension)

##### 6.3.1 Graphics

The implementation is done in three parts:

1. Modified material file: Each 3D mesh in the world is bound to a specific material through a proprietary script in a material file. In the Extension, some attributes had to be declared like the name of used shaders, what language they used, and the variables sent from the engine to the shaders.
2. Vertex shader: The vertex shader is mainly responsible for doing the matrix calculations to transform vertices from object space to world space, and then calculate the reflection vector that will be used.
3. Fragment shader: The fragment shader is responsible for applying the blurr value through forcing a certain LOD of the mipmaps of the

environment map. And by using the reflection vector it determines the texture coordinates used for reflection. Then Fresnel approximation is applied to enhance the realism of the reflection.

#### 6.3.2 AI

1. create a plugin for OpenSteer
2. build a logic-game map in the extension by using the border, obstacle and path class.
3. create a player object and use the position control interface to connect to the keyboard controller.
4. create several enemy objects and set the start position and start state.

#### 6.3.3 Physics

The PhysicsExtension class is implemented as a singleton and can handle several objects. When a new object is created, a new Obj instance is added to an array. When the Obj is instantiated, a Crack instance is associated with it. For speed purposes, in the Crack instance, the material structure is already generated as well as the crack pattern (Voronoi Diagram).

The material structure is generated by simply adding random values to a matrix. A hit point is then set in this matrix and a recursive method spreads through the nodes with the smallest numbers. Using these points as generating points, the Voronoi Diagram is created and so is the crack pattern.

When the PhysicsExtension::Destroy() method is called, the Destroy() method of the Obj instance to be destroyed is called. This method retrieves the vertex list for each fragment of the object and draws them on the screen. This vertex list is built by the Crack::generateEdgesList() method by iterating through the line fragments generated by the Voronoi Diagram. When the line is an edge of the polygon being handled, both ends (of the line) are added to the list.

When one of the ends of the line is added, tests are made to ensure the corners of the object are also added when appropriate. This list is then sorted in counterclockwise order by transforming the points to polar coordinates and sorting their angles from lowest to highest.

After having the fragments draw to the screen, their collision primitives are created through the Newton Physics engine. For performance reasons, thin boxes had to be used to enclose the fragments which can lead to weird behaviour sometimes.

Finally, each fragment is attached to its own Ogre::SceneNode and each of these nodes are attached to a parent Ogre::SceneNode which is returned by the PhysicsExtension::Destroy().

#### 6.3.4 Multimedia

We use Haar-like features to detect user's face because this algorithm is very flexible. We can change object to detect by change only cascade file. We use Boost library for create new thread. We show user's face on right-top of the screen, so user can see himself. After we work for this tech demo, We add smiley face to user's display face, so this tech demo is funnier then before. This smiley face change based on user's life point.

### 6.4 Integration and testing[banana]

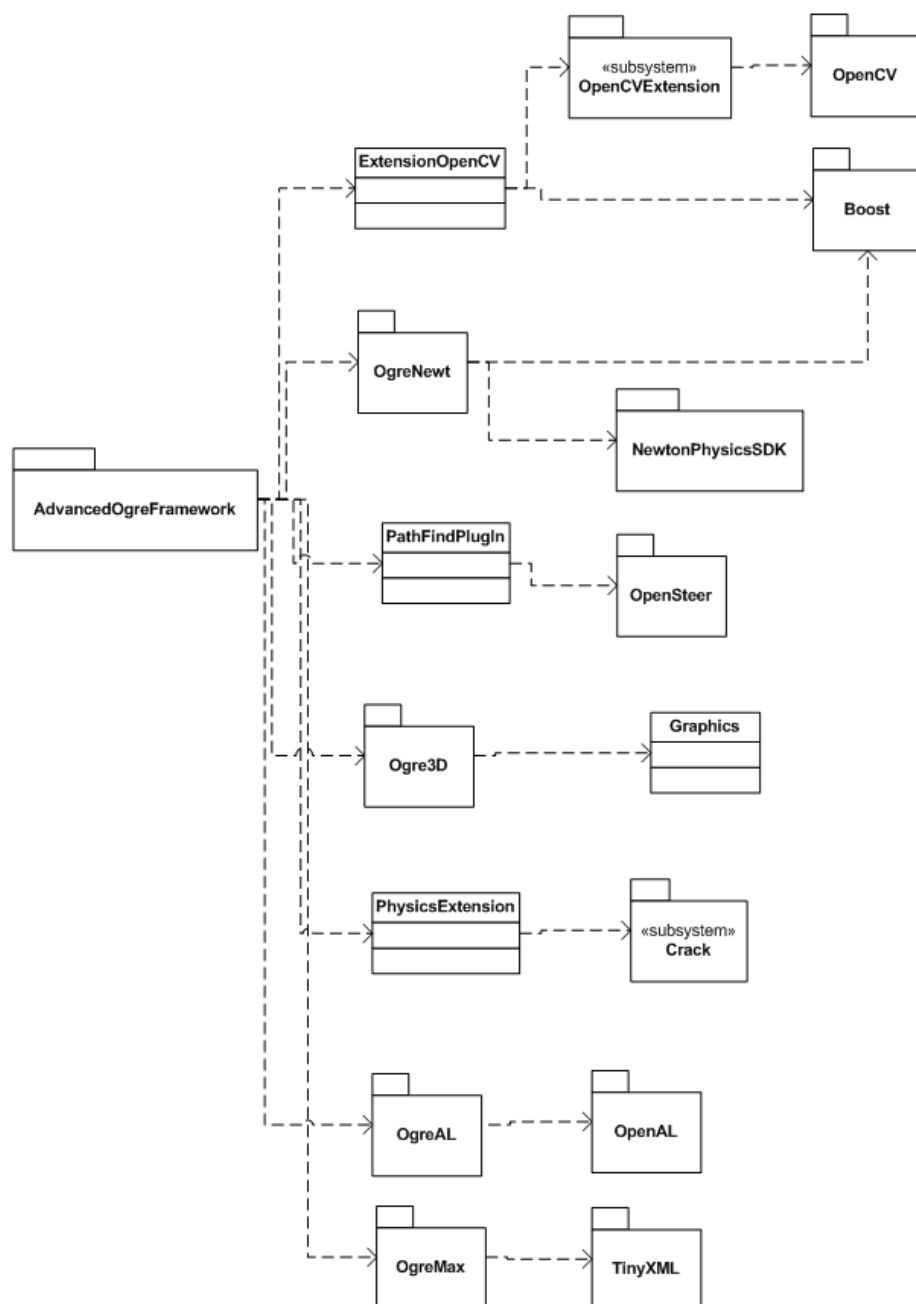
We have a lot of library from every extension, so we have to manage these library. We create environment variable to reference to folder of each library, therefore we can work in many computer without modify Visual C++ project. Then we can use SVN in the same file. In order to use SVN we should commit only important file. We structure folder of this tech demo by separate each extension, library and tech demo. We group all of resource files in one folder.

## 7 Tech demo

- o Description

This tech demo combines every extension together for making FPS game. This game is one player because we have AI extension.

- o Overall Design



- Implementation[Banana]

After integration, we implement game play of tech demo. We design Character and FPSEnemy class by using factory pattern, so we can create enemy so easily and we can add another type of enemy so easily too. We detect building collision by OpenSteer library instead of physics library because Opensteer have to use this data for calculate steering algorithm. We detect collision of character and gun by using ray and cylinder in physics. We crack plane as mirror of the building. We create physics plane at the bottom of this mirror because mirror will crack and drop on this plane.

We create real FPS game because we can win and lose. We will lose this game when player die. And we will win this game by kill all of the enemy. We also have walk, shoot and die animation of an enemy. We use sound effect from counter-strike.

- Screenshots





## Appendixes: Extension proposals

1- **Graphics Extension Description:** The basic Idea is to write a programmable shader (most probably with GLSL) and link it to the ogre3d frame work. The shader is responsible for calculating blurred reflection instead of the regular sharp reflections produced by environment maps. It uses mipmapping and level of details to produce these reflections.

More over to achieve more realistic results I'll add Fresnel effects to the reflections.

2- **AI Extension Description:** The extension of AI & script engine is focusing on the logic of the opponent or NPC creatures in FPS games. The AI of creatures in games can be divided into two parts, the path-finding and decision-making. The path-finding function should be a 3-D path-finding algorithm if possible. However, at least it could be a 2D path-finding engine which could be implemented by OpenSteer. In addition, Dijkstra algorithm could be involved in to help to find the shortest path.

The decision-making engine will be implemented in the Fuzzy logic way. The decision-making engine controls the action of creatures, and it decides when, whether and in which way to response any provoking signals made by the players.

3- **Physics Extension Description:** The goal of this extension is to provide the capability to generate destructible objects in real time to the simulation engine.

To achieve this, a cracking pattern algorithm should be implemented. This algorithm should generate a fracture to the 2D objects that will be then divided into more polygons. This will hopefully be realistic enough to significantly enhance user experience.

Finally, if possible, the extension should be able to deal with different sorts of materials.

4- **Multimedia Extension Description:** The extension will use Real time computer vision library (OpenCV). Camera will detect user face while playing game. Using face recognition algorithm to detect user face from camera. Then this extension send direction command or another command to engine by using position of the face in many frames. Additionally, extension send capturing frame to engine, then display frames in texture or interface.