1. INTRODUCTION ................................................................................................................................. 3
1.1. Project Scope ......................................................................................................................................... 3
1.2. Design Constraints and Limitations ................................................................................................. 4
1.3. Design Goals/Objectives ................................................................................................................. 4
2. DESIGN .................................................................................................................................................. 5
2.1. Dataflow diagrams ............................................................................................................................. 5
2.2. State Diagrams ................................................................................................................................. 14
2.3. ER Diagram ......................................................................................................................................... 14
2.4. Data dictionary ................................................................................................................................. 16
2.5. Sequence diagrams .......................................................................................................................... 21
2.6. Class diagrams ............................................................................................................................... 25
3. USE CASES ........................................................................................................................................ 32
3.1. Admin Use Case .............................................................................................................................. 32
3.2. User Use Case ................................................................................................................................. 33
4. NETWORK DESIGN .......................................................................................................................... 34
4.1. SERVER ............................................................................................................................................ 34
4.2. Client ................................................................................................................................................ 51
5. ARTIFICIAL INTELLIGENCE DESIGN .......................................................................................... 54
5.1. Artificial Intelligence Functionality Scenario ................................................................................. 54
5.2. AI Algorithmic description ............................................................................................................. 56
5.3. AI’s Data Flow Diagram ................................................................................................................. 57
5.4. AI’s State Transition Diagram ....................................................................................................... 58
5.5. Chasing and Evading ....................................................................................................................... 59
5.6. Flocking ........................................................................................................................................... 60
5.7. Pathfinding ...................................................................................................................................... 61
5.8. Fighting ........................................................................................................................................... 63
6. SCREEN DESIGN SAMPLES ........................................................................................................... 63
6.1. Entrance Screen ................................................................................................................................ 63
6.2. Main Game Play Moment Screen .................................................................................................... 64
6.3. Inventory Screen ............................................................................................................................... 64
7. PHYSICS ENGINE ............................................................................................................................. 64
8. SOUND ENGINE .................................................................................................................................. 68
9. CODING STANDARDS ...................................................................................................................... 69
9.1. Source code organization ................................................................................................................. 69
   Include Statements ............................................................................................................................ 70
   Multiple Inclusion of a header file ....................................................................................................... 70
9.2. Naming Conventions ....................................................................................................................... 71
9.3. Source Documentation ................................................................................................................. 72
9.4. Programming Conventions .......................................................................................................... 74
10. GANNT CHART ................................................................................................................................. 77
1. INTRODUCTION

1.1. Project Scope

1.1.1. Project Description

Project Core5 is a Massive Multiplayer Online Role Playing Game project. The scenario, defined for the game, affects the intent of the game. The game will proceed according to our scenario which will be described in the next parts of this document.

1.1.2. Project Features

Basically, from the design point of view our project will be composed of:

Network: to make multi-playing possible we are going to design the network of the game. The server side will be performed on Linux operating system and the client side will be performed on Windows operating system. Since the big and challenging part of our Project is this network topic, we will implement a complete model of a multi-client interacting server. On the other hand, to make the game more realistic in terms of response refresh times, synchronization is also a big issue that will be considered in particular.

3D Graphics: to build a powerful interface which will help both understandability and functionality modern games make use of complex graphic components. In our Project, we will pay big attention to developing rich user interfaces and attractive graphics by making use of Microsoft DirectX.

Artificial intelligence: to make the game more realistic, the living creatures will be created. In order to develop an attractive scenario those creatures should have the ability to think. Thinking will of course be implemented by our AI engine. But in order not to increase the complexity and avoid redundant processing time, the AI engine should provide an efficient algorithm and design.

Game Physics: in the light of being realistic and attractive, multiplayer game should provide well calculated and smooth movements. Beside that, it should check whether a specific move violates the physical rules of the game world. For example: collision detection, speed calculations, flight trajectories etc.

Sound System: to attract the player and make them feel excited and embedded in the scenario, rich sounds and themes will be provided. Each movement and character action will be associated with a
specific sound. Furthermore, the depth of the sound will be calculated in order to give a 3D sound surrounding environment. Sound system will be accomplished with the help of OpenAL.

1.2. Design Constraints and Limitations

1.2.1. Performance

We plan to distribute our game to a large audience varying from modern architecture to old systems. So performance is a vital task for providing a smooth execution even in the old architectures having limited resources. Hereafter it is the developer’s mission to provide an efficient and well-structured design. Optionally the developers may consider valuable the possibility of offering a number of different releases, designed to be executed on different target systems distinguished by their resource richness (DDRAM, Video Ram, CPU, etc.).

1.2.2. Time issue

The game is due the last days of the spring semester 06/07, so we have an important time limitation. For this reason we are forced to keep the level of details at a certain threshold. However we will make it possible to provide as much details as it is required for a normal implementation of the proposed scenario.

1.3. Design Goals/Objectives

1.3.1. Usability:

The user should be able to control the game without constraint. Since visual and audible aids are important for the player to comprehend the game, these aids should be attractive and understandable. Menus should be designed as clear as possible in order to prevent user lost in menus. Game flow and game scenario should be smooth in order to make the game adaptable.

1.3.2. Reliability:
Breakdowns should be removed. The game is reliable as far as it is error free so almost all the bugs should be debugged. Since multithreading will be used, all the possible deadlocks should be determined and then prevented. Network connection should not be dropped unexpectedly. Uncontrolled cheating will be prevented in order to provide justice.

1.3.3. Security:

The network security should be accomplished faultless. In order to ensure the money transfer safely, it is necessary to contact with professional and reliable corporations.

1.3.4. Supportability:

The game will be up to change and the updates will be distributed via internet. Production defects will be compensated.

2. DESIGN

2.1. Dataflow diagrams
2.1.1. DFD0

2.1.2. Client DFD1
2.1.3. Server DFD1
2.1.4. Client Network DFD2
2.1.5. Client Sound DFD2
2.1.6. Client Graphic Engine DFD2
2.1.7. Server AI DFD2
2.1.8. Server Physics DFD2
2.2. State Diagrams

The information about the state of the players is planned to be stored on the server. The main purpose of this approach is to achieve synchronization of data between clients. As stated in the diagram, personal user information, the characters the user has, the configuration and the statistical information the characters have are stored on the database server. The gameplay related data (character level, the number of kills, ...) will be saved at predetermined frequencies which is supposed to prevent data loss due to possible server failures and revive the server from that info. The user accounting information is also used by the web registration system and used by the game server upon login request from the user.

2.3. ER Diagram
2.4. Data dictionary

The data dictionary is presented here to give extended information about the ER diagram of the database server.

<table>
<thead>
<tr>
<th>Name</th>
<th>userID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned long unique</td>
</tr>
<tr>
<td>Description</td>
<td>Every user have unique user ID’s to distinguish them</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>nick</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>char(20) not null default</td>
</tr>
<tr>
<td>Description</td>
<td>Nick that selected by user at signing up to the game database and it will be used as his name in entire game</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>char(20)</td>
</tr>
<tr>
<td>Description</td>
<td>The name of the user that is written on certificate on birth card</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>char(20)</td>
</tr>
<tr>
<td>Description</td>
<td>Password that user defines and uses for logging in the game</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>char(40)</td>
</tr>
<tr>
<td>Description</td>
<td>The mail address that user use to sign up to the game and that activation link to be sent and also game news and advertisements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>membershipType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>char(40)</td>
</tr>
<tr>
<td>Description</td>
<td>The type that determines membership type of user, they are gold premium, platinum and free.</td>
</tr>
<tr>
<td>Name</td>
<td>membershipDuration</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Where it is used</td>
<td>USER</td>
</tr>
<tr>
<td>Type</td>
<td>char(40)</td>
</tr>
<tr>
<td>Description</td>
<td>Duration that is the amount of day that membership to be expired</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Type</td>
<td>char(20)</td>
</tr>
<tr>
<td>Description</td>
<td>Name that identifies user characters, it gives user the opportunity to give different names for each characters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>modelID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned long</td>
</tr>
<tr>
<td>Description</td>
<td>The ID that determines the display of the character by chosen characters’ shape at character creation mode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The value that determines other properties of the character and permits some new abilities by leveling up</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The value that to be completed to one level up, this value increases at higher levels</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The value that gives damage by to the enemy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>agility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>intelligence</td>
<td>The value that determines the speed of the character and it also determines to escape from hit by enemies</td>
</tr>
<tr>
<td>wisdom</td>
<td>The value that determines mana generation speed and base mana point, and it is also a prerequisite for some spells</td>
</tr>
<tr>
<td>endurance</td>
<td>The value that determines the resistance of character to the spells</td>
</tr>
<tr>
<td>constitution</td>
<td>The value that determines base hitpoint and generation rate of it</td>
</tr>
<tr>
<td>fame</td>
<td>The value that determines the user rankings</td>
</tr>
<tr>
<td>stamina</td>
<td>The value that determines the resistance to tiring</td>
</tr>
<tr>
<td>type</td>
<td>The type refers to potion, weapon, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Where it is used</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intelligence</td>
<td>CHARACTER</td>
<td>unsigned int</td>
<td>The value that determines the speed of the character and it also determines to escape from hit by enemies</td>
</tr>
<tr>
<td>wisdom</td>
<td>CHARACTER</td>
<td>unsigned int</td>
<td>The value that determines mana generation speed and base mana point, and it is also a prerequisite for some spells</td>
</tr>
<tr>
<td>endurance</td>
<td>CHARACTER</td>
<td>unsigned int</td>
<td>The value that determines the resistance of character to the spells</td>
</tr>
<tr>
<td>constitution</td>
<td>CHARACTER</td>
<td>unsigned int</td>
<td>The value that determines base hitpoint and generation rate of it</td>
</tr>
<tr>
<td>fame</td>
<td>CHARACTER</td>
<td>unsigned int</td>
<td>The value that determines the user rankings</td>
</tr>
<tr>
<td>stamina</td>
<td>CHARACTER</td>
<td>unsigned int</td>
<td>The value that determines the resistance to tiring</td>
</tr>
<tr>
<td>type</td>
<td>ITEM</td>
<td>char(20)</td>
<td>The type refers to potion, weapon, etc.</td>
</tr>
<tr>
<td>Name</td>
<td>itemID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Where it is used</td>
<td>ITEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>The id that uniquely determines the item</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Damage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>ITEM</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The amount that an item gives damage to the enemy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>ITEM</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The amount that an item gives resistance to the physical stress</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>ITEM</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The amount that determines the number that you have this item type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>ITEM</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The amount that determines item’s effect distance as diameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>EFFECT</td>
</tr>
<tr>
<td>Type</td>
<td>char(20)</td>
</tr>
<tr>
<td>Description</td>
<td>The type that determines the effect type, as strength, manapoint, hitpoint etc. increase or decrease</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Effected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>EFFECT</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The id that determines effect uniquely</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>gainAmount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>EFFECT</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The amount that an effect gives to or takes from you.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>KILL RECORD</td>
</tr>
<tr>
<td>Type</td>
<td>char(20)</td>
</tr>
<tr>
<td>Description</td>
<td>The creature type that you killed, every type has some different properties that needed by experience level etc calculations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>KILL RECORD</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The amount that you killed of this type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>KILL RECORD</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The level of the killed creature, it is needed by experience level fame etc calculations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>shapeID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>MODEL SHAPE</td>
</tr>
<tr>
<td>Type</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Description</td>
<td>The id that determines the model uniquely</td>
</tr>
</tbody>
</table>
2.5. Sequence diagrams

2.5.1. Login sequential diagram

![Login sequential diagram]

2.5.2. Trade sequential diagram
2.5.3. Item picking sequential diagram
2.5.4. Attack sequence diagram
2.6. Class diagrams
**PhysicEngine**
- collision : cCollisionDetection
- force : cNetForce
- calculateStep() : void

**CollisionDetection**
- noOfCollidedObjects : int
- objects : CollidedObject
- calculateCollisionVolume() : void
- detectObjectInCollisionVolume() : void
  + getCollidedObjects() : int*
  + getNoOfCollidedObjects() : int

**CollidedObject**
- property : cPhysicalProperty
- setProperty() : void
+ getProperty() : cPhysicalProperty

**NetForce**
- collidedObjects : CollidedObject
- worldIterate : cWorldIterate
- physicalProperty : cPhysicalProperty
- setCollidedObjects(in collidedObjects : int*, in noOfCollidedObjects : int) : void
- calculateNetForce() : void
+ worldIterate() : void

---

dPhysicEngine calculates the next step of every object in the physical world. The simulation is a combination of two sequential operations: collision detection and net force calculation. In the last step, the engine sets new physical values of objects.

**Design Decisions:**
- ODE will be used for simulating
3. USE CASES

3.1. Admin Use Case

- Kick Player
- Communicate
- Transport Player
- Ban Player
- Monitor
- View world
3.2. User Use Case
4. NETWORK DESIGN

4.1. SERVER

4.1.1. Server Functionality Scenario:

Server functions in a multi-threaded fashion, making use of POSIX “p thread” library. There will be primarily those kinds of threads:

- **Connection Listener**
- **Query Listener**
- **Request Handlers for each client**

1. **Connection Listener Daemon Thread:**

   This thread, (actually working like a synchronized daemon) , will handle new connections. New connections will be caught by UNIX sockets' listen routine. After catching the new connection this thread will read all identification information sent by the client. By identification information we mean name, password, etc. After checking the validity of this account the login status of this client will be updated. In case the login information is not valid the connection listener thread will sent a negative reply to the client, otherwise will send a confirmation code. Next this thread will fork off a new request handler thread for the new connection, (see 1.3). This thread will keep on repeating the connection listening routines until explicitly stopped by the server object.

1.2) **Query Listener Daemon Thread:**

   This thread, also working in a daemon fashion, will handle clients' requests. The thread will iterate to listen for incoming requests from currently active clients. The requests, (any type of action inquiry), will trig the execution to the corresponding thread (there will be one thread for each open connection). So, this thread will wake up the request handler thread and will sleep until the request is handled. This thread will keep on this executing cycle until stopped by the server.

1.3) **Request Handler Threads:**
These threads will handle each client's sent request by communicating with the game engine. After sending the requested action to the game engine, the threads will wait for the engines' response to this action. When the response is ready the request handler will sent the responses to each related client. Afterward the threads will sleep on their semaphores, i.e., will wait until another request has come.

An pictorial description of the 'new connection' and 'new request' functionalities for the server's scenario is shown below:

### 4.1.2. Algorithmic Description of the thread handlers

#### 2.1) Connection Listener Daemon Thread:

Repeat from 1 to MAX_CONNECTION_NO
{
  1. Listen for the new connection
  2. Read the login info from the connection
  3. Validate the login info from the server's internal database
  4. if( login info is not valid )
     {

35
4.1. Send a negative confirmation to the client

else
{

4.1. Send a positive confirmation to the client
4.2. Increment the no of connection counter by one via a special method making use of mutexes. This issue is of vital importance since all threads will make use of this counter, consequently there won’t exists some possibility for the race condition to cause any data hazards.
4.3. Set the socket flags non-blocking to avoid blocking the thread in case data is missing on that socket. This aspect is important in the query handler thread which continuously probes for requests from the connected socks.
4.4. Fork a new thread giving its identification arguments consisting of the connection socket id and the client name.

}  

5. Sleep to yield the scheduling to other threads

2.2) Query Listener Daemon Thread:

1. read current no of connections (via a mutex protected method)

repeat from 1 to CURRENT_NO_OF_CONNECTIONS
{

2. read from the sock(at iteration index), whether there is any request coming on that channel.

If( read is successful )
3. wake up the request handler thread # (iteration index),
   by issuing
   a signal call to the semaphore it is waiting.

4. If the query listener thread has been running more than a
   specified amount of time stop it, to give a look at new incoming
   connection. In other words, to yield the execution to the
   connection listener thread.

2.3) Request Handler Threads:

1. Read the requested action from the socket associated with
   the current request handler thread.
2. Send the action to the game engine and then wait until
   responses are ready.
3. After reading the resulting replies from the game engine,
   send the replies to each affected client via each client's
   socket.
4. wait on the appropriate semaphore
4.1.3. Server’s Data Flow Diagram

Server’s data flow diagram is primarily characterized by alternating between reading clients requests and servicing them via communicating with the game. In the honor of this task, we implement reading by making use of the socket reading routines in the connection listener thread. Hereafter the query listener iteratively probes for a request from the active connections. In case it catches a request it signals a semaphore and switches the context to the request handler thread. The request thread reads the request, sends it to the game engine, waits for replies and transmits the replies to affected clients.
4.1.4. **Server's State Transition Diagram**

Control flow in the server flows like the following: Server instance forks two threads namely the connection listener and the query listener. Only one of them will execute, of course, so we made use of a concurrency node to demonstrate that. It doesn’t matter in fact which one of them executes, since they alternate by issuing sleep routines. After that the query listener probes until catches a request. As it catches the request it passes the control to the appropriate request handler thread by signaling the appropriate semaphore. The request handler satisfies the request by calling the game engines, sends the replies to affected clients and waits on its semaphore. The control then returns to the query listener keeping on repeating this interaction. Once in a defined time slice the query listener gives execution to the connection listener to avoid starvation.
4.1.5. Server Synchronization Algorithms

Synchronization

Fairness is the main principle of developing a game, in the sense that all players should have equal chances and the game should provide fair judgments in all circumstances. You have probably noticed the aspect of fairness in modern games, while playing you feel that even in complex situation the correct order of actions is carried out. Regardless of game genre, the realism of online game play depends on how well the underlying network allows game participants to communicate in a timely and predictable manner.
Carrying out the right order of actions in scenarios evolving multiple actors (namely players) is called synchronization. Synchronization is one of the most challenging parts of a nondeterministic game. By nondeterministic we mean a special kind of scenario where the requests come in an unpredictable and irreproducible order. To go deeply into the analysis of synchronization bottle-necks, first we need to discuss the source of synchronization errors: basically network latency and jitter.

Latency:

Latency refers to the time it takes for a packet of data to be sent from its source address to its destination. Similarly the term RTT (Round Trip Time) is used to denote the time it takes for a packet to go to its destination, plus the time it takes for a response packet to go back from the destination to the source. In some online game players'/developers’ jargons the term ‘lag’ is used as a synonym for RTT.

Jitter:

Variance in latency from one packet to another is called Jitter. The formal definition of jitter is: Jitter, in electronics and telecommunications, is an abrupt and unwanted variation of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles. Jitter is a significant factor in the design of almost all communications links (e.g. USB, PCI-e, SATA, OC-48). Source: Wikipedia.

Speaking more informally, jitter is the dispersion of the changes in latency from one sent packet to another. Let us explain latency and jitter by some observations:

Assume a path is showing an average 100 ms latency. Obviously it might exhibit latencies of 90 ms and 110 ms for every alternate packet – fairly noticeable jitter in the short term, even though the long-term average latency is constant. Alternatively, the path might exhibit latency, that is, drifting – 90ms, 95 ms, 100 ms, 105 ms, 110 ms, 105 ms, 100 ms . . . and so on. For our discussion, it is sufficient to know that latency can change slowly or rapidly from one packet to the next.

The following picture will help understand the basic relationship between latency & jitter.
**Reasons Causing Latency:**

To be precise there are plenty of causes and defects causing delays, but we are focusing only on theoretical, namely predictable ones. The rest depends on internal organization of protocols and are much more difficult to be explained.

1. **Propagation delays between long distances:**

   The speed of transitions through copper wires or at the best case through optical fibers is finite. That is it will take more time to pass a longer distance. A rule of latency caused by a defined distance away from the destination is stated to be:

   \[ \text{Latency (ms)} = \frac{\text{distance of link in kilometers}}{300} \]

2. **Serialization delays:**

   The packets (and their subdivisions) are sent one after another in a serial fashion. The finite period taken to transmit an IP packet one bit at a time is referred to as serialization latency. This period of time depends on the speed of the link (in bits per second) and the length of the packet being sent. A rule of latency caused by serialization is stated to be:
Latency (ms) = \frac{8 \times \text{(Link layer frame length in bytes)}}{\text{(Link speed in Kbps)}}

3. **Queuing delays:**

When we are sending lots of packets through our connection, those packets are not sent immediately to the destinations. In contrary, they are queued in a queue holding the packages requested to be sent for one or many clients. The internal organizations differs from one architecture to another; however there will always be a queue created as long as the mean sending time is longer than the mean packet arrival time. Under such circumstances the time it takes for a packet to be submitted depends on the time it takes to send any packets that are in front of the queue with respect to our packet. In other words it will wait until previous requests are satisfied. We call this delay as “queuing delay”.

4. **Packet size variations:**

Intuitive enough, it will take more time to send a packet of size 23 kb than one of size 230 kb. That is explained by the principle of serialization explained above (‘one bit at a time’ transfer mechanism).

5. **Data corruption (bit errors):**

Murphy's Law: “Whatever can go wrong, will go wrong”.

Packets sent through a network may get lost on the way, or worse their content may get corrupted, that is changed. Those errors may be due to electrical glitches in hardware, due to signal-to-noise ratios in the digital-to-analogue-to-digital conversion process, etc.

To avoid data corruption, many systems offer a solution based on extra bits to control the format of remaining bits. For example ‘parity bits’ check whether the number of ones or zeros in the remaining bits is odd or even, by convention.

Consequently when receiving a packet the system performs tests to see if any bit-error have occurred. If a data corruption is caught then an algorithm is performed to repair the error. This is known as forward error correction, (FEC). In any case, uncorrected bit errors are usually discovered through cyclic redundancy check (CRC) calculations at both the transmitting and receiving end of a link.
Finally any error that cannot be corrected is resent, right after a request from the destination to resend the same packet. The time required to resend corrupted packages increases the latencies for those packets.

That was a brief introduction to latencies and jitter which are the primary causes to synchronization problems in a network. As stated above the jitter is the variance of the latencies of packets and the two terms are linearly related as shown on the example below.

![Graph showing the relationship between average jitter per map and average latency per map.](image)

**Figure 5.5** Jitter versus latency measured from an active Quake III Arena server in Australia [ARM2004]
Another concrete example is presented below, showing the effects of latency in a popular rugby game. (Courtesy of: John Wiley and Sons, Online and Game Programming, June 2006)
Latency Compensation Techniques

In order to get rid of the ugly errors, like the ones pictured above, many approaches have been developed and we will discuss some of them in the following lines. All of the approaches have two main principles in common: they either correct the latency or they hide its effects from the users by implementing various tricks.

Prediction:

Instead of waiting for the server to respond to each client action before satisfying it, the client can predict the server response, allowing the game client to respond to user input immediately, by predicting player and opponent movements before getting responses from the server. Briefly stating the main implementation is to perform locally all the actions that can be predicted, that is we predict the opponent’s actions based on knowledge on the opponent and knowledge on the laws of specific game under consideration. With opponent prediction, the location of a unit that is controlled by another player (or computer) is estimated. The estimation starts with the last known position of the unit
and computes its current, predicted position based on the speed and direction it was traveling. This predicted position is then used until the unit owner sends an update of the new location, speed or direction or both. This update would be sent when the unit owner determines that the other clients cannot accurately predict the position within a predetermined threshold. Let us illustrate the idea with an example inspired from a flight simulator game.

There is an important point to be clarified in this example. By saying threshold we mean an error tolerance, so; obviously the prediction is not 100% accurate. However it will give you an idea about the position of the plane. If the threshold is small you will get very near approximations.

**Time Manipulations:**

These kinds of manipulations try to repair the errors of latency. We will discuss two time manipulation algorithms:
- Time Delay
- Time Warp

**Time Delay:**
The idea lying in the time delay method is simple. It suggests waiting a little amount of time, so that even slower connections can submit their requests. Meanwhile the incoming requests are buffered, until the delay time expires. Once the expiration occurs all buffered requests are carried out after getting sorted out correctly based on submission times. Note that, in order to know the exact time we need a unique time in all clients and we need to put a timestamp on each packet.

As seen on the picture, the time delay method solves the cases where the Client 2 command was send before Client 1 command, but has arrived later due to latency.

Furthermore let us go back to one of our examples presented in the “WHY AND HOW DOES LATENCY EFFECT SYNCHRONIZATION?” section. Let us see how the time delay method solves that problem:
However, still there exists a powerful disadvantage regarding the time delay technique. What happens if a client’s connection is very slow, such that its request comes even after the delay time limit? In such case the same abnormality is repeated. One can propose the idea of putting some connection speed constraints to clients, so that slow connections will not be able to connect the game. But this proposal is not good in the commercial plane. Right below we will see an algorithm that solves those problems.
Time Warp:

A widely used time manipulation mechanism is to have a server to rollback (or time warp) the events in a game to the time when a client command was sent. The player provides input on the basis of the current state of the game at the client. Because of the lag between the client input and the server receiving the command, the state at the server may have changed. For example, the player shoots at an opponent at time t0, but by the time the message arrives at the server at time t2, the opponent had moved at time t1. Using time warp, the server rolls back the events it had processed since the client provided the input (roll back to time t0 in the above example). In this case, the server might determine that this older event has a bearing on subsequent events, changing their effect to make the global world state consistent. For example, the server may determine the player had hit and killed the opponent, meaning the opponent movement at time t2 was invalid. The algorithm performing the time warp idea is stated as:

- Receive packet from client
- Extract information (user input)
- Elapsed time = current time – latency to client
- Rollback all events in reverse order to current time – elapsed time
- Execute user command
- Repeat all events in order, updating any clients affected
- Repeat.

Data Comprehension:

By compressing the packets we are, in fact, reducing the size of packets. There are plenty of compression algorithms and most of them derive from the famous Huffman coding. As a consequence, reducing the packet size leads to smaller latency since the network has to submit less number of bits.
Peer To Peer:

Peer to peer is a relatively new networking paradigm, propagandizing the idea that any terminal can behave both as a server and as a client at the same time. We can use peer-to-peer connections between players in order to avoid extra load in the server. Concretely, a server need not know everything that players perform. For example voice chat can be made by a peer-to-peer connection between two clients instead of communicating via the server. The same discussion is valuable for text messages, provided that they are considered to be of little importance so that there is no need to save them.

Visual Tricks:

By visual tricks we mean visual animations evolving time-based frame refreshes. In fact the visual tricks are not related directly to networking, but they are valuable in hiding the effects of latency. For example when a player shoots in a “shoot-them-up” game, a lot of calculations have to be done to carry out the damage effects caused by the shoot. As a consequence the user may experience some millisecond delay right after the action. To hide the effect of complex calculations we may put a post-shooting animation to take out the attention of the player, while on the same time we can earn valuable time to compute the statuses’ changes caused by the damages.

4.2. Client

4.2.1. Client Functionality Scenario:

Client functions which is in a multi-threaded fashion, is composed of two threads, nameley:

- Sender
- Receiver
Before starting the explanation of these threads, firstly the connection process at the client-side will be explained at this part.

**Connection Process**

Since we use *connection-based* method, connection between client and server must be established before communication. Firstly, we create a socket in order to handle the communication (sending and/or receiving the messages, the actions) between the client and server. We used stream-based full-duplex communication (SOCK_STREAM) as the type of the socket, address format is host and port number (AF_INET) as the domain argument while creating the socket. Then, the connection process starts. We get the IP address of the server from its name. Then we try to connect with the created socket ID, given host address and given host port. After the establishing of connection, first the name and password of the user is sent to the server, one time at each connection. These information are checked at the server side and after the validation, the connection is established with the given user name. The online list which is held at the server-side is updated.

**Sender**

This thread will handle the action and message sending process. Sending process of actions and messages is repeated in an infinite loop in the “sending function” during the connection, until the user leaves the game which closes the socket of the client. This infinite sending action is controlled with the help the multi-threading. Sending thread is killed and waked up repeatedly. When the client closes the connection the online list which is held at the server side is updated.

**Receiver**

This thread will handle the action and message receiving process. Receiving process of actions and messages is repeated in an infinite loop in the “receiving function” during the connection, until the user leaves the game which closes the socket of the client. This infinite receiving action is controlled with the help the multi-threading. Receiving thread is killed and waked up repeatedly. When the client closes the connection the online list which is held at the server side is updated.
Algorithmic Description

Getting connection info

1. open the file which has the host info and is given by us
2. if the file cannot be opened successfully throw error
3. if succeeded read the information from the file namely, host address and host port.
4. close the file

Creating the socket

1. use host and port number (AF_INET) address format as the domain argument of the socket
2. use stream-based full-duplex communication (SOCK_STREAM) as the type argument of the socket
3. call the socket() function
4. if the socket cannot be created throw error
5. if successful get a file descriptor (socket ID) and save this info

Connecting to the server

1. get the IP address of the host from the name
2. put this info into the "struct hostent*" which holds info about the machine
3. fill struct sockaddr_in which is the internet socket address stuct for the internet family

```c
{
    sin_family = AF_INET; /* address format is host and port number */
    sin_port = ; /* is the port number */
    sin_addr = ; /* the IP address */
}
```
4. call connect() function with the socket ID, the filled address struct and its size
5. if unsuccessful throw error
6. if connection is successful send the name and password of the user
7. if server sends the validation, connection is established
   else inform the user and ask for the correct name and password 3 times
   if incorrect name and password entered 3 times close the socket

Sending function
1_ while (true) /* infinite loop */
{
   if (the new action object is created or changed
   || the user typed a chat message)
      clean the buffer
      copy the object to the buffer
      send the buffer with: the socket ID, pointer of buffer,
      the size of the buffer and the flag (0)
}

Receiving function

1_ while (true) /* infinite loop */
{
   clean the buffer
   receive the message from the server and put it into the buffer
}

5. ARTIFICIAL INTELLIGENCE DESIGN

5.1. Artificial Intelligence Functionality Scenario
Artificial intelligence is one of the major components of this project, as well as the most difficult one to implement precisely. In this phase of the design we will make use of the classing game playing algorithms such as alpha-beta pruning making use of heuristics functions evaluation.

In the game scenario we mentioned about non-human creatures as a part of the characters’ set. The behavior of all these creatures will be controlled by our artificial intelligence engine. The functionality potential of the creatures varies with respect to their types and their tendency to attacking and/or working. For example a dragon’s primary tendency is toward attacking and causing damage to opposite characters, while a rabbit has greater attitude towards gathering food and escaping from potential damage risks.

The design will be composed of:

- AI Engine
- Alpha-Beta Engine
- Heuristic Evaluation

**AI Engine:**

This is the primary engine of our AI component. The game engine will call the AI engine after each action to give opportunity to AI bots to respond to those actions, (bot = creature throughout this document). The engine will look for affected bots around the last action positions. In case there is any, it will search for best moves of each bots. To accomplish this task the AI engine calls the Alpha-Beta engine giving the bot id as its parameter. Finally it updates the bots status and returns the bots’ moves to the game engine.

**Alpha-Beta Engine:**

The alpha-beta engine is the “brain” of our AI system. It is provided a bot and it finds the best move for that bot. To accomplish that task it search for opposite players nearby the bot, and finds the best moves to be applied toward those players. Finding the best moves is completed by 2-ply alpha-beta...
pruning search, making use of heuristic evaluation in each base step. In the end it returns the move providing maximum gain among best moves.

**Heuristic Evaluation:**

The heuristic evaluation serves to evaluate the current status of the bot issuing an action and the surrounding players. After the evaluation it will return a value ranging from MAX_PROFIT to MIN_PROFIT. During the evaluation process it will make use of some heuristic functions which shapes the bot’s attitudes to searching for best move, for example: force a rabbit to basically gather food.

**Proposal (to be implemented in future phases):**

Even though it won’t be implemented in the present phase, the game shall provide the capability of machine learning. Our bots should learn from their experiences: successes and fails. The machine learning will be implemented by making use of neural networks. So the game bots will train the neural network for a considerably long time, (otherwise it will not show accurate results), then will search for an estimation of the present world condition through the neural network.

5.2. **AI Algorithmic description:**

**AI Engine:**

1. Request position of the last action from the game engine.
2. Get all bots nearby last action.
3. For each bot in 2. do
   

3.1) Calculate best move by the Alpha-Beta engine.
3.2) Apply the best move to the bots’ database.

Alpha-Beta Engine:

1. Get all players nearby the bot provided as argument.
2. For each player in 1. do :
   {  
     2.1 Calculate best move toward the player indexes by the iteration counter, (obviously one of the players in 1), by making use of Heuristic Evaluation. 
   }
3. Apply the move having greatest value, (profit), among ones calculated in ‘2.1’.

Heuristic Evaluations:

1. Evaluate world status of the bot under consideration by making use of basically these Heuristic functions:
   - Maximum total damage to opposite players/teams (let it be \( h_1 \))
   - Escape in case of death risk (let it be \( h_2 \))
   - Search for valuable objects and steal them (let it be \( h_3 \))

Note: The final heuristic function will be a linear combination of the above functions:
\[ H_f = c_1 \cdot h_1 + c_2 \cdot h_2 + c_3 \cdot h_3, \] where \( c_i \), \( i=1..3 \), are the coefficients describing the priority of each heuristic.

5.3. AI’s Data Flow Diagram

AI engine gathers actions positions from the game engine, and then gets the nearby bots from the world’s database. Afterwards it gives the id of one of the bots to the alpha-beta engine and waits for the best move of this bots. Alpha-beta engine, on the other part, finds the best move by interacting with the Heuristic Evaluator and then checks if the move is legal by communicating with the Physics Engine.
Lastly, the Heuristic Evaluator calculates the condition of the bot and its players around by checking from the world’s database the bot’s attributes.

5.4. AI’s State Transition Diagram

The control passes from the game engine to the AI engine after each action requested by the clients. The AI engine will pass the control to the alpha-beta engine for each bot found on a specific
(namely circular) area around the last action requested. The alpha-beta engine will pass controls to the Heuristic Evaluator for each base step in the ply-search algorithm.

5.5. Chasing and Evading

The primary purpose of our creatures will be to attack and give damage to opponent players. However, there may be cases during which a creature is forced to escape. The choice depends upon the strength of the opponent forces compared to the strength of the creatures, multiplied with their numbers.
E.g.: two creatures shall attack one knight, but a creature shall escape from three knights. In our terminology attacking will be denoted as chasing and escaping will be denoted as evading. Then the chaser will be denoted as predator and the evader will be denoted as prey. Let us see a piece of code showing the basic operations to implement chasing and evading:

<table>
<thead>
<tr>
<th>PREDATOR CHASING PREY</th>
<th>PREY EVADING FROM PREDATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (predatorX &gt; preyX)</td>
<td></td>
</tr>
<tr>
<td>predatorX--;</td>
<td></td>
</tr>
<tr>
<td>else if (predatorX &lt; preyX)</td>
<td></td>
</tr>
<tr>
<td>predatorX++;</td>
<td></td>
</tr>
<tr>
<td>if (predatorY &gt; preyY)</td>
<td></td>
</tr>
<tr>
<td>predatorY--;</td>
<td></td>
</tr>
<tr>
<td>else if (predatorY &lt; preyY)</td>
<td></td>
</tr>
<tr>
<td>predatorY++;</td>
<td></td>
</tr>
</tbody>
</table>

5.6. Flocking

Flocking means living and getting organized in groups. When creatures of our game stay in groups they are harder to be beaten and can profit of multiple shoots (one from each client) to kill the opponent(s) quickly. The basic algorithm is to follow the position of the neighbor creatures, that means follow their average position and their average direction. Since the same method will be applied from all the creatures, a coherent grouping will result.
This flocking method gives good results, but it has some disadvantages in those cases when a group will encounter an obstacle in the way. To avoid coming around in a silly way, we have to develop an obstacle avoiding algorithm. The algorithm resides in the fact that creatures can see the map in front of them and they can change directions as a whole. The idea is illustrated below:

Another remarkable strategy to make the group intelligent is to assign a leader for each group. Then all the members of a flock will follow the leader. In our game the leader can be the strongest creature.

5.7. Pathfinding
Path finding will be a simplified task in our game. The basic A* algorithm will be used to search a path, so that the creature avoid obstacles and finds any valuable weapons, or treasures in the way.
5.8. Fighting

The approach to be used in fighting detection and response resides behind a scan and evaluate principle. Let us give a brief algorithmic pseudo-code of our creature fighting approach.

```plaintext
1. Check the visible scan area (based on a defined radius).
2. For each fighting creature in the scan area do
   a. Check the danger of the creature
      i. Danger = f( no of opponents, remaining life of the creature)
   3. Go near the creature having greatest danger.
   4. Fight against the weak creature opponents.
```

As a conclusion the creature will never remain idle. It will check for any fighting possibility, which is the most aprioristic operation. Otherwise it will consider the movement algorithms.

6. SCREEN DESIGN SAMPLES

6.1. Entrance Screen
With the help of ODE especially collision events happen almost real. However, the more closely the simulation approximates Newton physics the more work load to the PCs and servers. In addition, in the known examples of MMORPGs physics simulation is reduced to a minimum level which gives at least a feeling of real world.

When game engine creates a world by `cWorld` its physics infrastructure is created by `cPhysicsWorld`. Afterwards basic attributes are set:

- The overall gravity of the world.
- The overall friction of the surface and objects. It should be noted that individual frictions of the objects may be set after.
- Maximum contact of one step. It is obvious that the more collision (contact) the more load to system. After setting this value it would be altered when new objects added to world for efficiency. Overall bounce value and minimum bounce velocity of the world, in other words, when objects collide if their bounce value is low the possibility to bounce is low also when an object collides if its velocity is less than minimum bounce velocity it does not bounce.
- Simulation level. If this level is low then the a step of simulation maps to a relatively bigger real time, conversely, if level is 10 simulation takes 0.1 second time period as unit time which is harder to calculate. It is also possible to increase and decrease this value after.

- The ground surface using `cTerrain`. Using the instance of `cTerrain` necessary values like `triMesh`, vectors, norms of surface are gathered.
The camera of the game using cCamera. Suppose you have two bodies (A and Camera) collide. The motion of A should affect the motion of Camera as usual, but Camera should not influence A at all. The Camera needs collision response so that it doesn't enter into any scene objects by mistake, but the motion of the camera should not affect the simulation. This is solved by:

when the collision is detected, we don't create a contact joint between A and Camera as we normally would. Instead, we attach the contact joint between camera and 0 (the static environment). That way the body A will appear to Camera as though it is static and unmovable. This approach may result in some penetration between A and Camera, but this will not be a problem in many applications.

The simulation works step by steps that is in each step using current conditions new conditions (future positions, rotations etc.) are determined. In each step physical calculations like collision, force is done with a callback function.

Up to this point the general properties are set, however not much done for individual objects. The properties of objects are set after adding them to object container m_pObjectContainer. It should be noteworthy that all properties have default values for stability. A group of these properties are directly possessed by cObject itself:

- Shape; the shape of the object can be
  1. Cylinder
  2. Box
  3. Sphere
  4. triMesh
  5. Joint.
  6. Capsule: Objects like players doesn't have a regular shape.

These types are used for redundant simulation. In other words, using these types a request for the radius of a box or a request for the one side of the sphere etc. would be eliminated.

- Triangular mesh data. If the shape is data, a key value is set which points this mesh is the ground or a regular object.
- Body ID; cPhysicsWorld deserves an index to treat all objects separately. Eventually, not all object have same properties.
- Geometry ID; every body have a geometry id which refers to a shape.

Other physical properties of the objects are hold in cPhysicalProperty:

- Coordinates; the coordinates are used by simulation to detect collisions and determine the future coordinates.
• Velocity; have effects on collision, bounce, friction etc.
• Rotation; is used for objects which may be rotate like a sphere.
• Friction; each object have different surface so different friction constants.
• Mass; is an important physical property used in Newton physics.
• Density; is used with mass to calculate volume of the objects for gravity calculations.
• Radius; the sphere shaped and cylinder shaped objects have radius.
• Sides of box; box shaped object have a, b and c sides.
• length; the cylinder have a length.

This were the values to be set for a good simulation. If we look deeper to simulation:
1. What units are used?
   Metric, MKS (Meter-Kilogram-Second)
2. How do we make upright capsule for player encapsulation?
   We set the orientation back to upright and the angular velocity to zero after every time step. For the ability to climb stairs or overcome small, steep obstacles we use a ray for "feet", and a spring to push the character upwards [with some damping].

3. How do we stop things from rolling off into infinity, or pendulums from swinging forever?

   ODE models a universe with frictionless hinges and no rolling friction, so problems like those are fairly common. The solution is to apply a force to simulate rolling friction.

   Rolling friction can be approximated fairly well using damping. Strictly speaking, damping is a force that reduces the amplitude of an oscillating system over time, but since the math is similar we'll be using the damping terminology here.

   To apply damping to a moving body:
   • Compute the velocity of the object that's supposed to slow down.
   • Multiply that velocity by a damping coefficient, like -0.01 -- this is your damping force
   • Apply that force with dBodyAddForce.

   Tweak the damping coefficient to get the object to slow down at the rate you want.

   When simulating rolling friction, we only apply the damping when the object is actually touching the ground.

4. How do we create an immovable body?
In other words, how do we create a body that doesn't move, but that interacts with other bodies? The answer is we create a geom only, without the corresponding rigid body object. The geom is associated with a rigid body ID of zero. Then in the contact callback when we detect a collision between two geoms with a nonzero body ID and a zero body ID, we simply pass those two IDs to the dJointAttach function as normal. This creates a contact between the rigid body and the static environment.

5. Can we use ODE with DirectX?

We plug in consistent numbers in one coordinate system, and apply all the conventions of that coordinate system, then the numbers we get out are, still, in that coordinate system. The only thing we may need to worry about is triangle winding order; if our mesh uses a winding order that's different from the default for the coordinate system we're using, we need to invert the winding of your triangles.

```c
D3DXVECTOR3 convertVector(const dReal *in)
{
    D3DXVECTOR3 out;
    out.x=(float) in[0];
    out.y=(float) in[1];
    out.z=(float) in[2];
    return out;
}

D3DQUATERNION convertQuaternion(const dReal *in)
{
    D3DQUATERNION out;
    out.x=(float)in[1];
    out.y=(float)in[2];
    out.z=(float)in[3];
    out.w=(float)in[0];
    return out;
}
```

Second argument can be the value returned by dBodyGetRotation(bodyID)
8. SOUND ENGINE

The sound simulation of the whole game is handled with OpenAL (a cross-platform 3D audio API) and ALUT (OpenAL Utility Toolkit). In sound engine applications, the main problem is the hardware support. It is important to know how many channels the hardware supports.

At first, sound files—.ogg or .wav—should be loaded to buffers. Afterwards, some properties of the sound source should be set:

- Velocity; if the source is moving, the listener is directly affected.
- Coordinate; if the source is far away from the listener, it may not be heard by him.
- Loop; whether it will play nonstop or once.

It should be noted that loading sound files can be dynamically done, also unloading them.

After setting source properties, listener properties are set:

- Velocity; if the listener is moving, the hearing of sound is directly affected.
- Coordinate; if the source is far away from the listener, it may not be heard by him.

Up to now all necessary values are set and sounds are ready to play. Now any sound can be played, paused, stopped or repeated.
9. CODING STANDARDS

9.1. Source code organization

9.1.1. Files and project organization

File names could be a mix of upper and lower case characters. The content of the file should be clearly seen by these names.

For the files which contain class definitions or implementations; the name should be same as the class name. Moreover, these files should contain exactly one class, except inner classes and private classes.

In order to avoid name conflicts the name of the headers should contain prefixes. For instance: If two separate module (main window, database) have GUI classes than they should be named as CMainWindowGUI.h and CDataBaseGUI.h”.

9.1.2. Header Files
**Include Statements**

Include statements should not contain the exact path. For example this statement is wrong:

```c
#include "/Utilities/GUI/Include/CMainWindowGUI.h"
```

but

d this statement is true:

```c
#include "CMainWindowGUI.h"
```

However, in this cases make files should be configured.

Moreover,

```c
#include "../Include/CMainWindowGUI.h" is a right statement in case the include files gathered in an exact relative location.
```

**Multiple Inclusion of a header file**

In order to avoid multiple inclusion.

In windows:

```c
#pragma once
#ifndef C_MAIN_GUI_H
#define C_MAIN_GUI_H
... Rest of Header File ...
#endif //C_MAIN_GUI_H
```

Moreover, only depended headers should be include by header files.

In Unix:

```c
#ifndef C_MAIN_GUI_H
#define C_MAIN_GUI_H
... Rest of Header File ...
#endif //C_MAIN_GUI_H
```
9.2. Naming Conventions

9.2.1. Function Names

Function names should tell what they do. These names should be a combination of words which have an uppercase first letter; except the first word. For example:
getSize(), setFirstMember(), print().

Function names should have a verb + object structure: printReport(), calcLimitOfMove().

The description of the return value can be used for function names: currentColor(),
nextEmptyCell(), screenReady().

Use more specific verbs instead of elastic verbs: processOutput() is wrong however,
formatAndPrintOutputToScreen() is right.

The properties of the classes should not be directly accessed instead use functions such that:
getColorValue(), setColorValue(). For boolean use “is” instead of get:
getFileReady() vs  isFileReady().

Function arguments should have “a_” prefix: a_intArraySize;

Example:

```plaintext
void sendOutputToDevice(string a_strDeviceName )
{
    /***************/
}
```

9.2.2. Class Names

These names are combination of strings which all words start with capitalized letters. Also, these names should have a capitalized “C” for the first letter: CNetworkClient.

9.2.3. Variable Names

Variable names should tell the content of the variable. These names should be a combination of words which have an uppercase first letter; except the first word.
The variables that are member of classes should have “m_” prefix.
Constant variables should have the #define statement structure: MAX_SIZE, DATE_FILE_NAME.

Pointers should have “p_” intermediate structure: an integer pointer ; p_nSizeOfFile;
A table for variables:
<table>
<thead>
<tr>
<th>Type</th>
<th>Member</th>
<th>Local</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>m_nName</td>
<td>nName</td>
<td>a_nName</td>
</tr>
<tr>
<td>char</td>
<td>m_cName</td>
<td>cName</td>
<td>a_cName</td>
</tr>
<tr>
<td>string</td>
<td>m_strName</td>
<td>strName</td>
<td>a_strName</td>
</tr>
<tr>
<td>boolean</td>
<td>m_bName</td>
<td>bName</td>
<td>a_bName</td>
</tr>
<tr>
<td>short</td>
<td>m_sName</td>
<td>sName</td>
<td>a_sName</td>
</tr>
<tr>
<td>long</td>
<td>m_lName</td>
<td>lName</td>
<td>a_lName</td>
</tr>
<tr>
<td>pointer(integer)</td>
<td>m_p_nName</td>
<td>p_nName</td>
<td>a_p_nName</td>
</tr>
<tr>
<td>Structure Circle</td>
<td>m_Circle</td>
<td>Circle</td>
<td>a_Circle</td>
</tr>
</tbody>
</table>

9.3. Source Documentation

Inline comments should have “//” style and they should have the same indent as the code they describe.

9.3.1. Module Comments and Revision history

Module comments should be placed on top of every file and should have the structure:
9.3.2. Commenting Functions
9.4. Programming Conventions

9.4.1. Layout Styles
void checkSomething(int a_nFirstParameter, string a_strSecondParameter)
{
    char cCurrentChar;
    doSomething();
    while (condition)
    {
        doSomething();
        doSomethingElse();
        if (condition)
        {
            doSomething();
        }
        else
        {
            doSomething();
        }
    }
    switch (condition)
    {
    case CASE_1:
        doSomething();
        break;
    case CASE_2:
    {
        doSomething();
        break;
    }
    default:
        doSomething();
    }
}
Wrong:

```java
if ('?0' <= inputChar && inputChar <= '?9') || ('?a' <= inputChar && inputChar <= '?z') || ('?A' <= inputChar && inputChar <= '?Z')
{
    doSomething(inputChar);
}
```

Right:

```java
if ('?0' <= inputChar && inputChar <= '?9') ||
    ('?a' <= inputChar && inputChar <= '?z') ||
    ('?A' <= inputChar && inputChar <= '?Z'))
{
    doSomething(inputCharput);
}
```

9.4.3. Large Function Calls

The method used in large functions calls is to align parameters to the end of the function name each on separate line:

```java
drawLine( Window.north,
    Window.south,
    Window.east,
    Window.west,
    currentWidth,
    currentHeight);
```

9.4.4. Error Handling

In failures objects should pass to error state not an undefined state. For instance: an error of opening a file shouldn't put the object to an locked state instead object should do less things than usual.
10. GANNT CHART
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start Date</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Learning DirectX API</td>
<td>21/11/2004</td>
<td>15d</td>
</tr>
<tr>
<td>2</td>
<td>Camera Implementation</td>
<td>28/11/2004</td>
<td>3d</td>
</tr>
<tr>
<td>3</td>
<td>Terrain Rendering</td>
<td>29/11/2004</td>
<td>6d</td>
</tr>
<tr>
<td>4</td>
<td>Tile Handling Implementation</td>
<td>07/12/2004</td>
<td>4d</td>
</tr>
<tr>
<td>5</td>
<td>Silent Character Model Implementation</td>
<td>22/12/2004</td>
<td>11d</td>
</tr>
<tr>
<td>6</td>
<td>Character Bone Animation Implementation</td>
<td>04/01/2005</td>
<td>10d</td>
</tr>
<tr>
<td>7</td>
<td>GUI Implementation</td>
<td>04/01/2005</td>
<td>10d</td>
</tr>
<tr>
<td>8</td>
<td>Simple World Editor Implementation</td>
<td>16/02/2005</td>
<td>12d</td>
</tr>
<tr>
<td>9</td>
<td>Lighting &amp; Shadowing Implementation</td>
<td>03/03/2005</td>
<td>13d</td>
</tr>
<tr>
<td>10</td>
<td>Sky &amp; Water Implementation</td>
<td>19/03/2005</td>
<td>12d</td>
</tr>
<tr>
<td>11</td>
<td>Terrain Optimization (Polygon Projection)</td>
<td>26/03/2005</td>
<td>11d</td>
</tr>
<tr>
<td>12</td>
<td>Extending World Editor</td>
<td>19/04/2005</td>
<td>11d</td>
</tr>
<tr>
<td>13</td>
<td>Special Effects (Reflection, Refraction, ...)</td>
<td>29/04/2005</td>
<td>4d</td>
</tr>
<tr>
<td>14</td>
<td>Snow, Multiplayer Implement</td>
<td>31/01/2006</td>
<td>11d</td>
</tr>
<tr>
<td>15</td>
<td>Server Network Implementation</td>
<td>26/12/2006</td>
<td>13d</td>
</tr>
<tr>
<td>16</td>
<td>Login Details Implementation</td>
<td>03/01/2007</td>
<td>15d</td>
</tr>
<tr>
<td>17</td>
<td>Alpha beta Testing implementation</td>
<td>17/01/2007</td>
<td>10d</td>
</tr>
<tr>
<td>18</td>
<td>Pathfinding algorithm implementation</td>
<td>30/01/2007</td>
<td>10d</td>
</tr>
<tr>
<td>19</td>
<td>Packet compression implementation</td>
<td>21/02/2007</td>
<td>10d</td>
</tr>
<tr>
<td>20</td>
<td>Total Implementation</td>
<td>08/03/2007</td>
<td>13d</td>
</tr>
<tr>
<td>21</td>
<td>Total Time (Week Method)</td>
<td>26/03/2007</td>
<td>15d</td>
</tr>
<tr>
<td>22</td>
<td>User Experience Method Implementation</td>
<td>19/04/2007</td>
<td>15d</td>
</tr>
<tr>
<td>23</td>
<td>Geometry Bounding Geometry Physics</td>
<td>10/05/2007</td>
<td>13d</td>
</tr>
<tr>
<td>24</td>
<td>Geometry Collision Detection</td>
<td>11/05/2007</td>
<td>18d</td>
</tr>
<tr>
<td>25</td>
<td>Geometry OpenAL with user files</td>
<td>11/26/2007</td>
<td>10d</td>
</tr>
<tr>
<td>26</td>
<td>Geometry Triangle Geometry Physics</td>
<td>12/09/2007</td>
<td>15d</td>
</tr>
<tr>
<td>27</td>
<td>Geometry Bones Physics</td>
<td>12/17/2007</td>
<td>10d</td>
</tr>
<tr>
<td>28</td>
<td>Geometry Physics Optimization</td>
<td>10/02/2007</td>
<td>10d</td>
</tr>
<tr>
<td>29</td>
<td>Geometry OpenAL with egg files</td>
<td>17/01/2007</td>
<td>16d</td>
</tr>
<tr>
<td>30</td>
<td>Geometry Advanced Input Handling</td>
<td>09/03/2007</td>
<td>25d</td>
</tr>
<tr>
<td>31</td>
<td>Geometry Calculating GUI Height</td>
<td>29/04/2007</td>
<td>40d</td>
</tr>
<tr>
<td>32</td>
<td>Vratal and Window Socket Programming Client</td>
<td>19/05/2007</td>
<td>14d</td>
</tr>
<tr>
<td>33</td>
<td>Vratala Windows Multimedia in Client instance</td>
<td>11/04/2007</td>
<td>20d</td>
</tr>
<tr>
<td>34</td>
<td>Vratala Windows Client Networking Client</td>
<td>12/01/2007</td>
<td>14d</td>
</tr>
<tr>
<td>35</td>
<td>Vratala Windows Client Position Action Clients</td>
<td>11/08/2007</td>
<td>10d</td>
</tr>
<tr>
<td>36</td>
<td>Vratala Windows Client Linux Synchronization</td>
<td>12/14/2007</td>
<td>15d</td>
</tr>
<tr>
<td>37</td>
<td>Vratala Linux Server Database Design</td>
<td>12/17/2007</td>
<td>13d</td>
</tr>
<tr>
<td>38</td>
<td>Vratala Shell Performance Analysis</td>
<td>17/06/2007</td>
<td>13d</td>
</tr>
<tr>
<td>39</td>
<td>Vratala Web Server Database Implementation</td>
<td>19/06/2007</td>
<td>21d</td>
</tr>
<tr>
<td>40</td>
<td>Vratala Game Engine Item Crafting implementation</td>
<td>09/03/2007</td>
<td>18d</td>
</tr>
<tr>
<td>41</td>
<td>Vratala Game Engine Item Programming Client</td>
<td>22/09/2007</td>
<td>15d</td>
</tr>
<tr>
<td>42</td>
<td>State Game Engine Item Rendering Implementation</td>
<td>08/03/2007</td>
<td>24d</td>
</tr>
<tr>
<td>43</td>
<td>State Game Engine Item Crafting Implementation</td>
<td>25/03/2007</td>
<td>15d</td>
</tr>
<tr>
<td>44</td>
<td>State Game Engine Item Database</td>
<td>12/22/2007</td>
<td>25d</td>
</tr>
<tr>
<td>45</td>
<td>State Web Site Design Implementation</td>
<td>16/04/2007</td>
<td>20d</td>
</tr>
<tr>
<td>46</td>
<td>State Windows Socket Programming Client</td>
<td>28/01/2008</td>
<td>16d</td>
</tr>
<tr>
<td>47</td>
<td>State Windows Networking Client</td>
<td>11/03/2008</td>
<td>10d</td>
</tr>
<tr>
<td>49</td>
<td>State Windows Client Socket Position Action Stack</td>
<td>12/09/2008</td>
<td>10d</td>
</tr>
<tr>
<td>50</td>
<td>State Windows Client Linux Synchronization</td>
<td>12/17/2008</td>
<td>13d</td>
</tr>
<tr>
<td>51</td>
<td>State Linux Server Database Design</td>
<td>12/25/2008</td>
<td>12d</td>
</tr>
<tr>
<td>52</td>
<td>State Server Performance Analysis</td>
<td>18/11/2008</td>
<td>12d</td>
</tr>
<tr>
<td>53</td>
<td>State Analysis Report Preparation</td>
<td>11/14/2008</td>
<td>14d</td>
</tr>
<tr>
<td>54</td>
<td>State Design Documentation initial</td>
<td>11/12/2008</td>
<td>12d</td>
</tr>
<tr>
<td>55</td>
<td>State Demonstration Preparation II</td>
<td>20/11/2008</td>
<td>10d</td>
</tr>
<tr>
<td>56</td>
<td>State Demonstration Preparation</td>
<td>12/09/2008</td>
<td>13d</td>
</tr>
<tr>
<td>57</td>
<td>State Design Documentation Final</td>
<td>17/12/2008</td>
<td>14d</td>
</tr>
<tr>
<td>58</td>
<td>State Demonstration Preparation Final</td>
<td>08/01/2009</td>
<td>20d</td>
</tr>
<tr>
<td>59</td>
<td>State Testing &amp; Debugging</td>
<td>22/05/2007</td>
<td>14d</td>
</tr>
</tbody>
</table>