CHAPTER - 3

Study of Related Technology

3.1 Introduction
3.2 Tools and Technologies
  3.2.1 Operating System: Windows 7 Enterprise
  3.2.2 The .Net Framework
3.3 Windows API
3.4 Adobe Photoshop CS
3.5 Unity 3D game engine
3.7 JAVA Script
3.8 3D Max
3.9 Collision Technique
3.10 Summary
Chapter 3: Study of Related Technology

3.1 Introduction

This chapter gives introduction to the tools and technologies used to design and implementation of the AI game system for steer behavior. Detailed information of each tool is beyond the scope of this document, yet, each tool is described with enough detail so that reader can become familiar with that tool.

3.2 Tools and Technologies

During the research work, following tools and technologies are used to simplify the job of analysis, design, and coding activities.

3.2.1 Operating System: Windows 7 Enterprise

Windows 7 is an operating system produced by Microsoft for use on personal computers, including home and business desktops, laptops, net books, tablet PCs, and media center PCs. It was released to manufacturing on July 22, 2009, and became generally available for retail worldwide on October 22, 2009, less than three years after the release of its predecessor, Windows Vista. Windows 7's server counterpart, Windows Server 2008 R2, was released at the same time. Windows 7 is succeeded by Windows 8 [51]. Of course, being a Microsoft product, Windows 7 is not a freeware or open source operating system. License is necessary to use it [51, 52].

3.2.1.1 Windows 7 Hardware Requirements

Windows 7 have mainly two versions; 32-bit, and 64-bit. Table 3.1 [24] shows minimum hardware requirements to use Windows 7 operating system.
3.2.2 The .NET Framework

The .NET Framework is a development framework created by Microsoft to enable developers to build applications that run on Microsoft and other platforms. The .NET framework has two components: (i) Common Language Runtime and (ii) .NET Framework class library. The Common Language Runtime (CLR) is the agent that manages your .NET applications at execution time. Applications that run on top of the CLR are known as managed code; all others are known as unmanaged code [51, 52].

The .NET Framework [23] class library is a comprehensive set of reusable classes that provides all the functionalities your applications need. This library enables you to develop applications ranging from desktop Windows applications to ASP.NET web applications, and Windows Mobile applications that run on Pocket PCs [51, 52].

<table>
<thead>
<tr>
<th>Component</th>
<th>Minimum System Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor (CPU)</td>
<td>1- GHz or faster, 32-bit (x86) or 64-bit (x64) processor</td>
</tr>
<tr>
<td>Memory</td>
<td>1 GB RAM (32-bit)</td>
</tr>
<tr>
<td></td>
<td>2 GB RAM (64-bit)</td>
</tr>
<tr>
<td>Graphics Processor</td>
<td>Support for DirectX 9 graphics with WDDM 1.0 or higher driver</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>16 GB available disk space (32-bit)</td>
</tr>
<tr>
<td></td>
<td>20 GB available disk space (64-bit)</td>
</tr>
</tbody>
</table>

Table 3.1: Windows 7 Hardware Requirements [51, 52]
Chapter 3: Study of Related Technology

(i) Common Language Runtime: The Common Language Runtime (CLR) is the virtual machine in the .NET Framework. It sits on top of the Windows operating system. A .NET application is compiled into a byte code format known as MSIL (Microsoft Intermediate Language). During execution, the CLR JIT (just-in-time) compiles the byte code into the processor’s native code and executes the application. The CLR also provide services like memory management, garbage collection, thread management, exception handling, and security[51,52].

(ii) .NET Framework Class Library: The .NET Framework class library contains classes that allow you to develop the following types of applications:

- Console applications
- Windows applications
- Windows services
- ASP.NET web applications
- Web services
- Windows Communication Foundation (WCF) applications
- Windows Presentation Foundation (WPF) applications
- Windows Workflow Foundation (WF) applications

The library’s classes are organized using a hierarchy of namespaces. For example, all the classes for performing I/O operations are located in the “System.IO” namespace, and classes that manipulate regular expressions are located in the “System.Text.RegularExpressions” namespace[51,52].

The .NET Framework class library is divided into two parts

- Framework Class Library (FCL)
Chapter 3: Study of Related Technology

- Base Class Library (BCL)

The BCL is a subset of the entire class library and contains the set of classes that provide core functionalities for your applications. BCL is available to all the languages using the .NET Framework. It encapsulates all the common functions such as file handling, database access, graphics manipulation, and XML document manipulation [51].

The FCL is the entire class library and it provides the classes for you to develop all the different types of application listed previously [23].

Following figure shows components of .NET Framework. All the components mentioned above are shown according to their hierarchy. In the figure 3.1, the layer consisting of ADO.NET and XML deals with database. (ADO – ActiveX Data Object, and XML - Extensible Markup Language)

Figure 3.1: Components of .NET Framework [51,53]
3.2.2.1 IDE: Microsoft Visual Studio 2008
IDE means Integrated Development environment. IDE provides development environment for coding various applications. In the past, it seemed that Microsoft had just as many development environments as it had languages or technologies. For example, before the introduction of Visual Studio .NET 2002, web development required one environment, Visual Basic development another, and C++ development yet another. You had to choose the appropriate development environment for the specific type of programming you were trying to accomplish. With the release of the new Visual Studio Integrated Development Environments (IDEs), you can now build all the possible .NET classes, components, and applications from a single environment – Visual Studio [51].

Visual Studio enables you to build any type of .NET components or application. When you use this tool, you can choose any of the Microsoft .NET compliant languages for building your applications; plus it allows you to create Windows Forms, XML Web services, .NET components, mobile applications, ASP.NET applications, and more [23, 51].

Visual Studio 2008 was released in early 2008, by Microsoft. It is an extremely versatile and powerful environment for developing .NET applications. Visual Studio 2008 has following features which make developer’s job easy [23][51].

(i) Components of IDE: Visual Studio 2008 has components like toolbar, menu bar, tool box, design pane, solution explorer, properties pane, and error list pan.

(ii) Code and Text Editor which includes code snippets, IntelliSense statement completion, IntelliSense support for object properties, methods and events, and Refactoring support (restructuring of code).
Chapter 3: Study of Related Technology

(iii) Debugging tools to observe run time behavior of the program.
(iv) Unit testing feature by auto-generation of the code to test your application.

3.2.2.2 Server-side scripting language: C# .NET

C# is an object oriented programming language. It inherits many of the best features of C++ and Microsoft Visual Basic, but with some of the inconsistencies and anachronisms removed, resulting in a cleaner and logical language. C# also contains a variety of useful new innovations that accelerate application development, especially when used in conjunction with Microsoft Visual Studio .NET [51,55]. In C# .NET, the C# compiler specifically targets .NET which means that all code written in C# will always run within the .NET Framework. This has two important consequences for the C# language [55]:

(i) The architecture and methodologies of C# reflect the underlying methodologies of .NET.
(ii) In many cases, specific language features of C# actually depend upon features of .NET, or of the .NET base classes.

One important thing to make clear is that C# is a language in its own right. Although it is designed to generate code that targets the .NET environment, it is not itself part of .NET. Some features are supported by .NET but not by C#, and some features of the C# language are not supported by .NET (for example, some instances of operator overloading) [51, 55].

3.3 Windows API

When writing applications for a PC running Microsoft Windows we need to use the Windows API (sometimes referred to as the Windows 32 API or Win32) [24]. When writing games we use it as a base for our
Chapter 3: Study of Related Technology

DirectX applications. We also may want to use it to create game tools, like level editors, effects editors, resource managers etc. When working on a Playstation 1 game I used the API to create level editors that created world data that could then be passed to the Playstation. So even if you intend to work only with consoles it is still useful to learn to program. DirectX of course runs under Windows and some knowledge of the API is needed in order to get a DirectX application up and running [51].

The Windows API provides a lot of functionality that we will not be using. For our minimum needs we need to be able to create a window and receive user input. We may also want some dialogs and some menus the user can interface with[51].

3.4 Adobe Photoshop CS

Adobe Photoshop CS [25] is a raster graphics editor developed and published by Adobe Systems for Windows and OS X.

Photoshop was created in 1988 by Thomas and John Knoll. Since then, it has become the de facto industry standard in raster graphics editing, such that the terms "photoshopping" and "photoshop contest" are born. It can edit and compose raster images in multiple layers and supports masks, alpha compositing and several color models including RGB, CMYK, Lab color space (with capital L), spot color and duotone. Photoshop has vast support for graphic file formats but also uses its own PSD and PSB file formats which support all the aforementioned features. In addition to raster graphics, it has limited abilities to edit or render text, vector graphics (especially through clipping path), 3D graphics and video. Photoshop's feature set can be expanded by Photoshop plug-ins, programs developed and distributed
Chapter 3: Study of Related Technology

independently of Photoshop that can run inside it and offer new or enhanced features [51].

Photoshop’s naming scheme was initially based on version numbers. However, on October 2003, following the introduction of Creative Suite branding, each new version of Photoshop was designated with "CS" plus a number; e.g. the eighth major version of Photoshop was Photoshop CS and the ninth major version was Photoshop CS2. Photoshop CS3 through CS6 were also distributed in two different editions: Standard and Extended. In June 2013, with the introduction of Creative Cloud branding, Photoshop’s licensing scheme was changed to that of software as a service and the "CS" suffixes were replaced with "CC". Historically, Photoshop was bundled with additional software such as Adobe ImageReady, Adobe Fireworks, Adobe Bridge, Adobe Device Central and Adobe Camera RAW[51].

Alongside Photoshop, Adobe also develops and publishes Photoshop Elements, Photoshop Lightroom, Photoshop Express and Photoshop Touch. Collectively, they are branded as "The Adobe Photoshop Family".

3.5 Unity3D game engine

As of 2012, Unity (or Unity3D) is the most popular game engine used for developing 3D cross-platform mobile games for iOS and Android. Unity is an industry-level engine comparable in capabilities to Unreal game engine, Torque and CryEngine, yet it costs 1-2 orders of magnitude less which makes it popular among small studios and indie developers [57].

Unity3D can be used by programmers, artists and game designers. It has a game editor with an interface resembling one of 3D modeling
packages (3D MAX, Maya, Blender). A game developer populates the game scene with 3D objects (game objects), assigns materials to them, places cameras and lights.

To define behaviors of the game objects, the developer creates scripts and assigns them to game objects. For each level of the game the developer creates a separate game scene. Then the developer selects the target platform (for example, iOS), and the Unity3d engine creates a build ready for installation onto that platform. The programmer’s role is mostly to write the scripts.

Scripts for Unity3D can be written in C#, JavaScript or Boo (a simple Python-like language). All scripts for this project were written in C# and JavaScript inside this project which is included with the Unity3D engine.

**Figure 3.2:** Game editor of the Unity3D game engine [26, 57]
Figure 3.2 shows a state of the GUI of the game editor of Unity3D engine. The Inspector panel on the right shows game components on the Main Camera game object. One of the components is a Camera script which determines the game-specific behavior of the camera. The Hierarchy panel above shows game objects in the scene. The Project panel below shows game assets (scripts, textures, sounds, scenes, materials etc.) of the project. The editor allows to run the game inside an in-built simulator (on the left).

### 3.5.1. Component-based programming

A game world is represented in Unity3D as a scene populated with game objects. Each game object has a Transform: a component which determines position, rotation and scale of the object in 3D space. Apart from that, every game object may have a set of other components which define its behavior and properties. Unity3D implements a component-based programming principle which uses a concept of components instead of building inheritance hierarchies [56, 57].

In usual object-oriented programming, classes inherit from other classes to override behaviors, e.g. Cat and Dog can inherit from Mammal class override its say() method so that the cat would say "meow" and the dog say "woof".

If pure hierarchical approach is used, than a game typically ends with having gigantic classes inside which conceptually unrelated methods are tightly coupled. For example, a Cat class would be able to say its name, render itself on the screen, collide with other objects, serialize itself into an XML file, find path to a given point on the map, show a tooltip and do a lot of other things [26]. Such class would have...
hundreds of methods and dozens of variables scattered around the inheritance hierarchy.

In components-based model the game object is just a container instead of base class. It has a list of components, one component per property or behavior, which makes the system very modular and allows to have much smaller and manageable classes (see Figure 3.2 and Figure 3.3).

If a game object has a RigidBody component, it is considered by the engine as a physical body and is processed by the physics engine. For example, forces will be applied to it. It will fall down if gravitation is enabled for it.

If the object has a Collider component (various types of colliders are available including simple but imprecise BoxCollider and slow but accurate Mesh Collider), the object will start colliding with other objects with colliders.

Renderer components allow the object to be visible for cameras. Audio Source component allows the object to emit sounds and Audio Listener (typically placed on the main camera of the scene) -to detect sounds.
Chapter 3: Study of Related Technology

Artificial Intelligent Game System for Steer Behavior
3.5.2. Prefabs

A game may not only use game objects which were initially placed in the scene by the game designer but also require to create objects in runtime (for example, in a strategy game place newly produced units). This can be done programmatically by creating a game object and adding all of the required components to it and setting its properties.

For easier instantiation of objects Unity3D allows to save some objects as Prefabs (prefabricated objects) which can then be copied into the scene in runtime. Prefabs in Unity3D act as blueprints for objects. Instantiating a game object from a prefab is conceptually very close to instantiating an instance of a class in object-oriented programming.

A common practice is to create a prefab for every object that may be used in the game.

3.5.3. Plugins

Unity3D provides an AssetStore - a portal where developers can purchase game assets (for example, 3D models or textures) as well as plugins (collections of scripts which add functionality to the game editor or allow to implement parts of the currently developed game easier). In the development of "MyAI" the following plugins were used:

- NGUI-plugin to create GUI of the game in a better way than the in-built GUI system of Unity3D allows
- Lumos-plugin to gather in-game analytics.
Chapter 3: Study of Related Technology

3.5.4 Camera in Unity 3D

Namespace: UnityEngine
Inherits from: Behaviour

Description

A Camera is a device through which the player views the world [57].

A screen space point is defined in pixels. The bottom-left of the screen is (0,0); the right-top is (pixelWidth,pixelHeight). The z position is in world units from the camera. A viewport space point is normalized and relative to the camera. The bottom-left of the camera is (0,0); the top-right is (1,1). The z position is in world units from the camera. A world space point is defined in global coordinates (eg. Transform.position).

Static Variables[57]

| allCameras | Returns all enabled cameras in the scene. |
| allCamerasCount | The number of cameras in the current scene. |
| current | The camera we are currently rendering with, for low-level render control only (Read Only). |
| main | The first enabled camera tagged "MainCamera" (Read Only). |

Table 3.2: static variable[57]

Variables

| actualRenderingPath | Actually used rendering path (Read Only). |
| aspect | The aspect ratio (width divided by height). |
| backgroundColor | The color with which the screen will be cleared. |
| cameraToWorldMatrix | Matrix that transforms from camera space to world space (Read Only). |
| clearFlags | How the camera clears the background. |
### Chapter 3: Study of Related Technology

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cullingMask</td>
<td>This is used to render parts of the scene selectively.</td>
</tr>
<tr>
<td>depth</td>
<td>Camera's depth in the camera rendering order.</td>
</tr>
<tr>
<td>depthTextureMode</td>
<td>How and if camera generates a depth texture.</td>
</tr>
<tr>
<td>eventMask</td>
<td>Mask to select which layers can trigger events on the camera.</td>
</tr>
<tr>
<td>farClipPlane</td>
<td>The far clipping plane distance.</td>
</tr>
<tr>
<td>fieldOfView</td>
<td>The field of view of the camera in degrees.</td>
</tr>
<tr>
<td>hdr</td>
<td>High dynamic range rendering.</td>
</tr>
<tr>
<td>layerCullDistances</td>
<td>Per-layer culling distances.</td>
</tr>
<tr>
<td>layerCullSpherical</td>
<td>How to perform per-layer culling for a Camera.</td>
</tr>
<tr>
<td>nearClipPlane</td>
<td>The near clipping plane distance.</td>
</tr>
<tr>
<td>orthographic</td>
<td>Is the camera orthographic (true) or perspective (false)?</td>
</tr>
<tr>
<td>orthographicSize</td>
<td>Camera's half-size when in orthographic mode.</td>
</tr>
<tr>
<td>pixelHeight</td>
<td>How tall is the camera in pixels (Read Only).</td>
</tr>
<tr>
<td>pixelRect</td>
<td>Where on the screen is the camera rendered in pixel coordinates.</td>
</tr>
<tr>
<td>pixelWidth</td>
<td>How wide is the camera in pixels (Read Only).</td>
</tr>
<tr>
<td>projectionMatrix</td>
<td>Set a custom projection matrix.</td>
</tr>
<tr>
<td>rect</td>
<td>Where on the screen is the camera rendered in normalized coordinates.</td>
</tr>
<tr>
<td>renderingPath</td>
<td>Rendering path.</td>
</tr>
<tr>
<td>stereoConvergence</td>
<td>Distance to a point where virtual eyes converge.</td>
</tr>
</tbody>
</table>
### Chapter 3: Study of Related Technology

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stereoEnabled</td>
<td>Stereoscopic rendering.</td>
</tr>
<tr>
<td>stereoSeparation</td>
<td>Distance between the virtual eyes.</td>
</tr>
<tr>
<td>targetTexture</td>
<td>Destination render texture (Unity Pro only).</td>
</tr>
<tr>
<td>transparencySortMode</td>
<td>Transparent object sorting mode.</td>
</tr>
<tr>
<td>useOcclusionCulling</td>
<td>Whether or not the Camera will use occlusion culling during rendering.</td>
</tr>
<tr>
<td>velocity</td>
<td>Get the world-space speed of the camera (Read Only).</td>
</tr>
<tr>
<td>worldToCameraMatrix</td>
<td>Matrix that transforms from world to camera space.</td>
</tr>
</tbody>
</table>

Table 3.3: variables[56]

### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculateObliqueMatrix</td>
<td>Calculates and returns oblique near-plane projection matrix.</td>
</tr>
<tr>
<td>CopyFrom</td>
<td>Makes this camera’s settings match other camera.</td>
</tr>
<tr>
<td>Render</td>
<td>Render the camera manually.</td>
</tr>
<tr>
<td>RenderToCubemap</td>
<td>Render into a static cubemap from this camera.</td>
</tr>
<tr>
<td>RenderWithShader</td>
<td>Render the camera with shader replacement.</td>
</tr>
<tr>
<td>ResetAspect</td>
<td>Revert the aspect ratio to the screen’s aspect ratio.</td>
</tr>
<tr>
<td>ResetProjectionMatrix</td>
<td>Make the projection reflect normal camera’s parameters.</td>
</tr>
<tr>
<td>ResetReplacementShader</td>
<td>Remove shader replacement from camera.</td>
</tr>
<tr>
<td>ResetWorldToCameraMatrix</td>
<td>Make the rendering position reflect the</td>
</tr>
</tbody>
</table>

*Artificial Intelligent Game System for Steer Behavior*
### Chapter 3: Study of Related Technology

#### Artificial Intelligent Game System for Steer Behavior

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScreenPointToRay</td>
<td>Returns a ray going from camera through a screen point.</td>
</tr>
<tr>
<td>ScreenToViewportPoint</td>
<td>Transforms position from screen space into viewport space.</td>
</tr>
<tr>
<td>ScreenToWorldPoint</td>
<td>Transforms position from screen space into world space.</td>
</tr>
<tr>
<td>SetReplacementShader</td>
<td>Make the camera render with shader replacement.</td>
</tr>
<tr>
<td>SetTargetBuffers</td>
<td>Sets the Camera to render to the chosen buffers of one or more RenderTextures.</td>
</tr>
<tr>
<td>ViewportPointToRay</td>
<td>Returns a ray going from camera through a viewport point.</td>
</tr>
<tr>
<td>ViewportToScreenPoint</td>
<td>Transforms position from viewport space into screen space.</td>
</tr>
<tr>
<td>ViewportToWorldPoint</td>
<td>Transforms position from viewport space into world space.</td>
</tr>
<tr>
<td>WorldToScreenPoint</td>
<td>Transforms position from world space into screen space.</td>
</tr>
<tr>
<td>WorldToViewportPoint</td>
<td>Transforms position from world space into viewport space.</td>
</tr>
</tbody>
</table>

Table 3.4: Functions[56]

#### Static Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAllCameras</td>
<td>Fills an array of Camera with the current cameras in the scene, without allocating a new array.</td>
</tr>
</tbody>
</table>

Table 3.5: static Functions[56]
Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Does the object exist?</td>
</tr>
<tr>
<td>operator !=</td>
<td>Compares if two objects refer to a different object.</td>
</tr>
<tr>
<td>operator ==</td>
<td>Compares if two objects refer to the same.</td>
</tr>
</tbody>
</table>

Table 3.6: Operator

3.5.5 Unity 3D Editor Overview

The Main Editor Window is made up of several Tabbed Windows, called Views. There are several types of Views in Unity - they all have specific purposes which are described in the subsections below.

Figure 3.4 and Figure 3.5 shows Toolbar window and Profiler window in Unity 3D engine.
Figure 3.4: Toolbar window in Unity 3D [26,56]
Figure 3.5: Profiler window in Unity 3D [26, 56, 57]
Chapter 3: Study of Related Technology

3.5.6 Attaching to Unity players

To profile your game running on another device or a player running on another computer, it is possible to connect the editor to that other player. The dropdown Active Profiler will show all players running on the local network. These players are identified by player type and the host name running the player “iPhonePlayer (Toms iPhone)”. To be able to connect to a player, the player must be launched with the Development Build checkbox found in the Build Settings dialog. From here it is also possible to tick a checkbox to make the Editor and Player Autoconnect at startup[57].

Profiler Controls

![Profiler Controls](image)

Profiler controls [Figure 3.6 and 3.7] are in the toolbar at the top of the window. Use these to turn profiling on and off, navigate through profiled frames and so on. The transport controls are at the far right end of the toolbar. Note that when the game is running and the profiler is collecting data clicking on any of these transport controls will pause the game. The controls go to the first recorded frame, step one frame back, step one frame forward and go to the last frame respectively. The profiler does not keep all recorded frames, so the notion of the first frame should really be thought of as the oldest frame that is still kept in memory. The “current” transport button causes the profile statistics window to display data collected in real-time. The Active Profiler popup menu allows you to select whether profiling should be done in the editor or a separate player (for example, a game running on an attached iOS device)[57].

Artificial Intelligent Game System for Steer Behavior
Chapter 3: Study of Related Technology

1) Deep Profiling

[56,57] When you turn on Deep Profile, all your script code is profiled - that is, all function calls are recorded. This is useful to know where exactly time is spent in your game code.

Note that Deep Profiling incurs a very large overhead and uses a lot of memory, and as a result your game will run significantly slower while profiling. If you are using complex script code, Deep Profiling might not be possible at all. Deep profiling should work fast enough for small games with simple scripting. If you find that Deep Profiling for your entire game causes the frame rate to drop so much that the game barely runs, you should consider not using this approach, and instead use the approach described below. You may find deep profiling more helpful as you are designing your game and deciding how to best implement key features. Note that for large games deep profiling may cause Unity to run out of memory and so for this reason deep profiling may not be possible.

Manually profiling blocks of your script code will have a smaller overhead than using Deep Profiling. Use Profiler.BeginSample and Profiler.EndSample scripting functions to enable and disable profiling around sections of code.

2) View SyncTime

When running at a fixed framerate or running in sync with the vertical blank, Unity records the waiting time in “Wait For Target FPS”. By default this amount of time is not shown in the profiler. To view how much time is spent waiting, you can toggle “View SyncTime”. This is also a measure of how much headroom you have before losing frames.
Chapter 3: Study of Related Technology

Profiler Timeline

Figure 3.7: Profiler Timeline[56,57]
The upper part of the Profiler window displays performance data over time [see Figure 3.7]. When you run a game, data is recorded each frame, and the history of the last several hundred frames is displayed. Clicking on a particular frame will display its details in the lower part of the window. Different details are displayed depending on which timeline area is currently selected.

The vertical scale of the timeline is managed automatically and will attempt to fill the vertical space of the window. Note that to get more detail in say the CPU Usage area you can remove the Memory and Rendering areas. Also, the splitter between the timeline and the statistics area can be selected and dragged downward to increase the screen area used for the timeline chart.

The timeline consists of several areas: CPU Usage, Rendering and Memory. These areas can be removed by clicking the close button in the panel, and re-added again using the Add Area drop down in the Profile Controls bar.

Note that the coloured squares in the label area can control whether the associated timeline is displayed or not. To remove a sample from the display click on the colour key. The key will dim and the data will be removed from the graph. This can be useful to identify the cause of spikes in the CPU graph, for example.
3.5.7 Scripting in Unity 3D

Scripting is an essential ingredient in all games. Even the simplest game will need scripts to respond to input from the player and arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behavior of objects or even implement a custom AI system for characters in the game [56,57].

Scripting is a skill that takes some time and effort to learn; the intention of this section is not to teach you how to write script code from scratch but rather to explain the main concepts that apply to scripting in Unity.

3.5.8 Animation in Unity 3D

Mecanim Animation System

Unity has a rich and sophisticated animation system called Mecanim. Mecanim provides [56]:

- Easy workflow and setup of animations on humanoid characters.
- Animation retargeting - the ability to apply animations from one character model onto another.
- Simplified workflow for aligning animation clips.
- Convenient preview of animation clips, transitions and interactions between them. This allows animators to work more independently of programmers, prototype and preview their animations before gameplay code is hooked in.
- Management of complex interactions between animations with a visual programming tool.
• Animating different body parts with different logic.

Figure 3.8 Animation in Unity 3D[56]
Typical setup in the Visual Programming Tool and the Animation Preview window [see Figure 3.8]

**Mecanim workflow:** Workflow in Mecanim can be split into three major stages[56].

![Figure 3.8: Animation in Unity 3D](image-url)
Chapter 3: Study of Related Technology

1. **Asset preparation and import.** This is done by artists or animators, with 3rd party tools, such as Max or Maya. This step is independent of Mecanim features[56].

2. Character setup for Mecanim, which can be done in 2 ways:
   - **Humanoid character setup.** Mecanim has a special workflow for humanoid models, with extended GUI support and retargeting. The setup involves creating and setting up an Avatar and tweaking Muscle definitions.
   - **Generic character setup.** This is for anything like creatures, animated props, four-legged animals, etc. Retargeting is not possible here, but you can still take advantage of the rich feature set of Mecanim, including everything described below.

3. **Bringing characters to life.** This involves setting up animation clips, as well as interactions between them, and involves setup of State Machines and Blend Trees, exposing Animation Parameters, and controlling animations from code.

Mecanim comes with a lot of new concepts and terminology. If at any point, you need to find out what something means, go to our Animation Glossary.
Legacy animation system

While Mecanim is recommended for use in most situations, especially for working with humanoid animations, the Legacy animation system is still used in a variety of contexts. For example, you might use it when working with legacy animations and code (content created before Unity 4.0). Unity intends to phase out the Legacy animation system over time for all cases by merging the workflows into Mecanim[56].

3.5.9 Asset Import and Creation in Unity 3D

A large part of making a game is utilizing your asset source files in your GameObjects. This goes for textures, models, sound effects and behavior scripts. Using the Project View inside Unity, you have quick access to all the files that make up your game [56].
Chapter 3: Study of Related Technology

Artificial Intelligent Game System for Steer Behavior

Figure 3.9: Project View[56]

The Project View displays all source files and created Prefabs[ see Figure 3.9].

This view shows the organization of files in your project’s Assets folder. Whenever you update one of your asset files, the changes are immediately reflected in your game!

To import an asset file into your project, move the file into (your Project folder)->Assets in the Finder, and it will automatically be imported into Unity. To apply your assets, simply drag the asset file from the Project View window into the Hierarchy or Scene View. If the asset is meant to be applied to another object, drag the asset over the object.
3.6 Third Person camera setting Technique

If you want the camera to follow your character around the world you need to take into account the direction and position of the character to be followed[58,61].

You will probably want the camera to be a set distance away from the object and facing the same direction as the object. So the first thing to do is obtain the position and direction of the object. Once you have these you can position the camera a set distance from the object by multiplying the inverse of the object direction vector by the set distance away you want the camera to be:

![Figure 3.10: Third Person camera setting](image)

```cpp
// objPos - is the position of the object in the world
// objDir - is the normalised direction vector of the object in the world
// cameraPos and cameraDir are the position and direction of the camera that we want to find

// Position the camera 5 world units (dx) away from the object:
```
D3DXVECTOR3 inverseObjDir = -objDir;

cameraPos = objPos + (5.0f * inverseObjDir);

// Camera direction is just set to the same as the object:

cameraDir = objDir;

[58] The above will position the camera [see Figure 3.10] at the same height as the object, you may want the camera to always be above the object looking down. In this case you need to raise the camera position up and then calculate a new 'downward facing' camera direction.

[58] To do this you get the vector between the camera position and the object position. This then gives you the correct direction vector [see Figure 3.11].
Chapter 3: Study of Related Technology

Figure 3.11: Vector between camera and objecting position[58].

Following on from the previous code you would add code something like this:

```
// We want the camera to be 4 world units above the object it is looking at
cameraPos.y+=4.0f;

// We have to change the camera direction to look down toward the object
D3DXVECTOR3 newDir = objDir - cameraPos;
D3DXVec3Normalize(&newDir,&newDir);

// now the newDir is the correct camera direction
```
Chapter 3: Study of Related Technology

3.7 JAVA Script

JavaScript (JS) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed. It is also being used in game development, server-side network programming (with Node.js), the creation of desktop and mobile applications and mobile games[52,59,60].

JavaScript is a prototype-based scripting language with dynamic typing and has first-class functions. Its syntax was influenced by C. JavaScript copies many names and naming conventions from Java, but the two languages are otherwise unrelated and have very different semantics. The key design principles within JavaScript are taken from the Self and Scheme programming languages. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles[52,59].

The application of JavaScript in use outside of web pages—for example, in PDF documents, site-specific browsers, and desktop widgets—is also significant. Newer and faster JavaScript VMs and platforms built upon them (notably Node.js) have also increased the popularity of JavaScript for server-side web applications. On the client side, JavaScript was traditionally implemented as an interpreted language but just-in-time compilation is now performed by recent (post-2012) browsers [52, 60].

JavaScript was formalized in the ECMAScript language standard and is primarily used as part of a web browser (client-side JavaScript).
Chapter 3: Study of Related Technology

This enables programmatic access to objects within a host environment[59,60].

3.7. 1 Features

The following features are common to all conforming ECMAScript implementations, unless explicitly specified otherwise[52].

3.7.1.1 Imperative and structured

JavaScript supports much of the structured programming syntax from C (e.g., if statements, while loops, switch statements, etc.). One partial exception is scoping: C-style block scoping is not supported. Instead, JavaScript has function scoping (although, block scoping using the let keyword was added in JavaScript 1.7). Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, which allows the semicolons that would normally terminate statements to be omitted [59].

3.7.1.2 Dynamic

Dynamic typing as in most scripting languages, types are associated with values, not with variables. For example, a variable x could be bound to a number, then later rebound to a string. JavaScript supports various ways to test the type of an object, including duck typing[59].

3.7.1.3 Object-based

JavaScript is almost entirely object-based. JavaScript objects are associative arrays, augmented with prototypes (see below). Object property names are string keys. They support two equivalent syntaxes: dot notation (obj.x = 10) and bracket notation (obj['x'] = 10).
Properties and their values can be added, changed, or deleted at run-time. Most properties of an object (and those on its prototype inheritance chain) can be enumerated using a for...in loop. JavaScript has a small number of built-in objects such as Function and Date[59].

3.7.1.4 Run-time evaluation

JavaScript includes an evaluation function that can execute statements provided as strings at run-time [59].

DHTML allows scripting languages to change variables in a web page's definition language, which in turn affects the look and function of otherwise "static" HTML page content, after the page has been fully loaded and during the viewing process. Thus the dynamic characteristic of DHTML is the way it functions while a page is viewed, not in its ability to generate a unique page with each page load [59].

3.8 3D Max

Autodesk 3ds Max, formerly 3D Studio Max, is a professional 3D computer graphics program for making 3D animations, models, games and images. It was developed and produced by Autodesk Media and Entertainment. It has modeling capabilities, a flexible plugin architecture and can be used on the Microsoft Windows platform. It is frequently used by video game developers, many TV commercial studios and architectural visualization studios. It is also used for movie effects and movie pre-visualization [52].

In addition to its modeling and animation tools, the latest version of 3ds Max also features shaders (such as ambient occlusion and subsurface scattering), dynamic simulation, particle systems, normal map creation and rendering, global illumination, a customizable user interface, and its own scripting language.[52].
3.9 Collisions Technique

Handling collisions between world objects is obviously essential to computer games. Collision calculations can take time however so how accurate you make your collision detection will depend on your game type. If you are looking for very accurate physics you will want to carry out ray collision tests against all the triangles that make up your world, however for most cases you can get away with much less accurate tests. Bounding spheres and bounding boxes described below are a rather crude collision mechanism but are often all you need. Even if you do require more accurate collisions you will still use bounding shapes to allow you to reject whole objects before going down to the triangle level i.e. you want to avoid as many ray-triangle tests as possible so a first pass bounding box test will eliminate many triangles.

Bounding Box Collisions

Bounding boxes and bounding spheres can be used to find collisions between objects. They are fairly simple to calculate and the cost of carrying out the collision detection is relatively cheap. Note: there are other bounding volumes that can be used.

Bounding Box

A bounding box is simply a box that encloses all the geometry of a 3D object. We can easily calculate one from a set of vertex by simply looping through all the vertices finding the smallest and biggest x, y and z values. This gives us the bottom near left co-ordinate and the top far right co-ordinate. Call these minBounds and maxBounds.

Direct3D provides a function that calculates the bounding box for you:
D3DXComputeBoundingBox(...)  

This takes a vertex buffer, number of vertices and a byte size for your FVF and returns min and max bounds. A complete example using a mesh is shown below:

BYTE* pVertices=NULL;

hr=m_mesh->LockVertexBuffer(D3DLOCK_READONLY, (LPVOID*)&pVertices);
if (FAILED(hr))
    return FALSE;

D3DXVECTOR3 minBounds,maxBounds;

D3DXComputeBoundingBox((D3DXVECTOR3*)pVertices, m_mesh->GetNumVertices(), D3DXGetFVFVertexSize(m_mesh->GetFVF()), &minBounds, &maxBounds);

m_mesh->UnlockVertexBuffer();

The only new part of the above is the function D3DXGetFVFVertexSize, this is required because you need to tell the function how many bytes there are between each vertex (the stride), this is dependent on your custom vertex structure size[61].

So after calculating your min and max bounds manually or via the above method you have all you need to describe your model bounding box. Since this box is calculated in model space it is known as an axis aligned bounding box in model space. An example is shown below [see Figure 3.12][61].
Artificial Intelligent Game System for Steer Behavior

Figure 3.12: Bounding Box

**Bounding Sphere**

As you can see the bounding box is quite a good fit for a Dalek although the bottom being wider than the top means collisions with the head of the Dalek based solely on a bounding box would appear to happen without actually touching. Another method would be to use a
bounding sphere, a bounding sphere makes it much simpler to calculate collisions as it just defines a center point and a radius[28]. To determine if a collision has occurred between two objects with bounding spheres we can simply find the distance between their centers and see if this is less than the sum of their bounding sphere radius [see Figure 3.13][61].

![Figure 3.13 Bounding Sphere][61]

As you can see this is also not a very good fit and for ground based collisions may actually be worse than using bounding boxes (in the case of this Dalek). So the best choice of bounding shape is dependent on the original shape, if the original shape is box like then a box may be the best fit, if it is a ball then obviously a bounding sphere is a better fit. Also sphere collision detection is less costly than bounding box detection. Direct3D again provides a function to calculate the sphere for you from a set of vertices:
3.10 Summary

This chapter gives overview of the study of the technologies used to design and develop the Artificial Intelligent Game System for Steer Behavior system for proposed Pathfinding and Steer behavior in 3D game. It introduces windows 7 operating system, Visual Studio 2008, .NET framework, C#, 3D Max, Windows API, Photoshop CS, Unity 3D engine and JAVA Script. The main purpose of this chapter is to introduce the reader of this document to the above mentioned technologies.