*Asian Journal of Computer and Information Systems (ISSN: 2321 – 5658)*
*Volume 02 – Issue 04, August 2014*

# Development of a Real-Time Strategy Game

Oluwafemi J. Ayangbekun[1], Ibrahim O. Akinde[2]

[1] Department of Information Systems, Faculty of Commerce,
University of Cape Town, South Africa.
*Corresponding author email: phemmyc {at} yahoo.com*

[2] Department of Computer Science, College of Information and Communication Technology,
Crescent University, Abeokuta, Nigeria.

**ABSTRACT—** ***In this paper, we iterate in detail the steps involved in the process of developing a real-time strategy game. We give a brief review of the history of RTS games and take a quick look at previously developed RTS games. We present an eight-section divide and conquer methodology in designing the game. Finally, we develop a spin-off RTS game with major emphasis on level design, gameplay mechanics, user interface development and resource management.***

**Keywords—** Mesh, Previs, Kismet, Fog of War

## 1. INTRODUCTION

Real-Time Strategy games (RTS) are a genre of computer and video games. They take place in real-time and involve resource gathering, base building, technology development and high-level control over individual units. Within a real-time strategy game, players become the leader of a colony or military base [1]. For example, in Age of Empires, the player controls an explorer, which is a unit capable of exploring the map, creating a settlement and acquiring special buildings for the player. At the beginning of the game several players are teleported into the world, where they begin by constructing a base. Real-time strategy games have a strong economic side, with players in Age of Empires constructing, upgrading and managing a variety of buildings that produce the three basic resources – food, wood and coin. Players invest these basic resources into improving their economy, fortifying their base, strengthening their military, and developing various forms of technology. Once constructed, players maneuver their military units around the world, attacking and engaging enemy units, the ultimate goal being to destroy the opponent's explorer and level the base of any opponents foolish enough to challenge their supremacy.

Underneath the surface game-play, video games are fundamentally about making decisions and exercising skills. A car racing simulation involves a great deal of skill in controlling the vehicle along with decisions involving the choice and setup of the vehicle [1]. Real-time strategy games, while varying greatly in content and style, are unified by a set of common decisions that their players make [1]. These decisions involve a variety of challenging problems that players are simultaneously solving: resource allocation - developing their economy; force composition - training and equipping an effective military; opponent modeling - estimating the location and composition of enemy forces; spatial reasoning - predicting incoming attacks or defensive vulnerabilities, while hiding and misleading their enemies about their own intentions. In contrast to other types of games, the decisions involved in RTS games concentrate player involvement around making high level, long term strategic decisions [1]. The paradigm is designed to draw players into the game world giving them a set of interesting decisions to make, along with compelling reasons about why they are making them and their resulting consequences. Real-Time strategy games present a unique opportunity for research, containing a variety of interesting research problems within an integrated and motivated whole [1].

In developing a real-time strategy game, the following aspects are essential:

- The playing field.
- The graphics engine.
- The game interface.
- Management of unit data.
- Artificial Intelligence Agent Navigation for successfully moving units from point A to point B.
- Unit states and actions.
- Unit interaction.
- Designing the game mechanics.

- Designing the races featured in the game.
- Creating balance, considering both the character stats/costs/spells, and whether one race has an environmental advantage/disadvantage with regards to map layout.

## 2. LITERATURE REVIEW

During the last decade, the scientific community has acknowledged that real-time strategy games constitute rich environments for researchers to develop complex resource management schemes and battle strategies which can be related to real world events. Much of the research that can be found about the development of real-time strategy games concerns the mixture of genres and sub-genres of video games to create an immersive experience for players. An example is the combination of the role-playing game genre and the real-time strategy genre in the popular game, Frozen Hearth.

### *2.1 Real-Time Strategy Games*

The genre that is recognized today as "real-time strategy" emerged as a result of an extended period of evolution and refinement. Games that are today sometimes perceived as ancestors of the real-time strategy genre were never marketed or designed as such at the original date of publication. As a result, designating "early real-time strategy" titles is problematic because such games are being held up to modern standards. The genre initially evolved separately in the United Kingdom, Japan, and North America, afterward gradually merging into a unified worldwide tradition. At least one source considers Intellivision's Utopia by Don Daglow (1982) to be an early real-time strategy game. In Utopia, two players build resources and carry out combat by proxy. It contains the direct-manipulation tactical combat now common in that the players can assume direct control over boats to sink the opponent's fishing boats. However, the game used a turn-based strategy interface, one where the turns are timed [2]. BYTE in December 1982 published as an Apple II type-in program Cosmic Conquest. The winner of the magazine's annual Game Contest, its author described it as a "single-player game of real-time action and strategic decision making". The game has elements of resource management and war-gaming [3]. Another 1982 example is Legionnaire on the Atari 8-bit family, written by Chris Crawford for Avalon Hill. This was effectively the opposite of Utopia, in that it offered a complete real-time tactical combat system with variable terrain and mutual-help concepts, but lacked any resource collection and economy/production concepts. As a result, this game might be better considered an early installment of the real-time tactics, or RTT, genre.

In the United Kingdom, the genre's beginning can be traced to Stonkers by John Gibson, published in 1983 by Imagine Software for the ZX Spectrum, and Nether Earth published on ZX Spectrum in 1987. In North America, the oldest game retrospectively classified as real-time strategy by several sources [4] [5] is The Ancient Art of War (1984), designed by Evryware's Dave and Barry Murry, followed by the sequel The Ancient Art of War at Sea in 1987, although Dani Bunten Berry's (of M.U.L.E fame) Cytron Masters (1982), developed by Ozark Softscape and released by SSI, also has been considered the earliest game of the genre [6] [7]. In Japan, the genre's beginning can be traced to Bokosuka Wars (1983), an early strategy RPG (or "simulation RPG"); the game revolves around the player leading an army across a battlefield against enemy forces in real-time while recruiting/spawning soldiers along the way, for which it is considered by Ray Barnholt of 1UP.com to be an early prototype real-time strategy game. This led to several other games that combine role-playing and real-time strategy elements, such as the 1988 Kure Software Koubou computer strategy RPGs, First Queen and Silver Ghost, which featured an early example of a point-and-click interface, to control characters using a cursor [8]. Another early title with real-time strategy elements was Sega's Gain Ground (1988), a strategy-action game that involved directing a set of troops across various enemy-filled levels [9] [10]. TechnoSoft's Herzog (1988) is regarded as a precursor to the real-time strategy genre, being the predecessor to Herzog Zwei and somewhat similar in nature, though primitive in comparison [11]. In 1989 Bullfrog released Populous for the Commodore Amiga which was the first god game and also a real-time strategy game. Scott Sharkey of 1UP considers Cytron Masters and other real-time examples prior to Herzog Zwei to be tactical rather than strategic, due to most lacking the ability to construct units or manage resources [12]. Herzog Zwei, released for the Sega Genesis in 1989, is the earliest example of a game with a feature set that falls under the contemporary definition of modern real-time strategy [13]. In Herzog Zwei, though the player only controls one unit, the manner of control foreshadowed the point-and-click mechanic of later games. It introduced much of the genre conventions, including unit construction and resource management, with the control and destruction of bases being an important aspect of the game, as were the economic/production aspects of those bases [12].

Notable as well are early games like Mega Lo Mania by Sensible Software (1991) and Supremacy (also called Overlord – 1990). Although these two lacked direct control of military units, they both offered considerable control of resource management and economic systems. In addition, Mega Lo Mania has advanced technology trees that determine offensive and defensive prowess. Another early (1988) game, Carrier Command by Realtime Games, involved real-time responses to events in the game, requiring management of resources and control of vehicles. The early game Sim Ant by Maxis (1991) had resource gathering and controlling an attacking army by having them follow a lead unit. However, it was with the release of Dune II from Westwood Studios (1992) that real-time strategy became recognized as a distinct genre of video games [14].

### *2.2 Previously Developed Real Time Strategy Games*

Frozen Hearth is a real-time strategy game that mixes in role-playing game elements to create a unique experience set in a dark fantasy realm where almost everything is an enemy. Frozen Hearth features single player, multiplayer, and co-op play to help keep gameplay experiences fun and unique. Frozen Hearth was developed by Immanitas Entertainment and Epiphany Games, and it features heroic characters with skill-based character progression, powerful spells, and many hero items.

Hostile Worlds is a squad based real-time multiplayer game. Two to eight players, divided into two teams, battle each other on a desolate alien world. Each team's goal is to collect a certain amount of alien artifacts while constantly being under attack by the opposing team and AI controlled aliens. Hostile Worlds' core experience consists of choosing one's own tactic and play style by the player constantly customizing his squad during the game: Several unit types are available and each comes with a unique set of characteristic abilities which can be unlocked by leveling up. Each squad's backbone is the squad commander who provides crucial abilities like healing and cloaking.

StarCraft is a military science fiction real-time strategy video game developed and published by Blizzard Entertainment and released for Microsoft Windows on March 31, 1998 [15]. Set in a fictitious timeline during the Earth's 25th century, the game revolves around three species fighting for dominance in a distant part of the Milky Way galaxy known as the Koprulu Sector: the Terrans, humans exiled from Earth skilled at adapting to any situation; the Zerg, a race of insectoid aliens in pursuit of genetic perfection, obsessed with assimilating other races; and the Protoss, a humanoid species with advanced technology and psionic abilities, attempting to preserve their civilization and strict philosophical way of living from the Zerg. Many of the industry's journalists have praised StarCraft as one of the best [16] and most important [17] video games of all time, and for having raised the bar for developing real-time strategy games [17]. Blizzard Entertainment's use of three distinct races in StarCraft is widely credited with revolutionizing the real-time strategy genre [18]. All units are unique to their respective races and while rough comparisons can be drawn between certain types of units in the technology tree, every unit performs differently and requires different tactics for a player to succeed. The enigmatic Protoss have access to powerful units and machinery and advanced technologies such as energy shields and localized warp capabilities, powered by their psionic traits. However, their forces have lengthy and expensive manufacturing processes, encouraging players to follow a strategy of the quality of their units over the quantity [19].The insectoid Zerg possess entirely organic units and structures, which can be produced quickly and at a far cheaper cost to resources, but are accordingly weaker, relying on sheer numbers and speed to overwhelm enemies [20]. The Terrans provide a middle ground between the other two races, providing units that are versatile and flexible. They have access to a range of more ballistic military technologies and machinery, such as tanks and nuclear weapons [21]. Although each race is unique in its composition, no race has an innate advantage over the other. Each species is balanced out so that while they have different strengths, powers, and abilities their overall strength is the same. The balance stays complete via infrequent patches (game updates) provided by Blizzard [22]. StarCraft features artificial intelligence which scales in difficulty, although the player cannot change the difficulty level in the single-player campaigns. Each campaign starts with enemy factions running easy AI modes, scaling through the course of the campaign to the hardest AI modes. In the level editor provided with the game, a designer has access to four levels of AI difficulties: "easy", "medium", "hard" and "insane", each setting differing in the units and technologies allowed to an AI faction and the extent of the AI's tactical and strategic planning [23]. The single-player campaign consists of thirty missions, split into ten for each race.

## 3. RESEARCH METHODOLOGY

The methodological approach taken in developing the real-time strategy for this project is divided into 8 specific areas listed below, in no specific order. While some aspects of the approaches can be simultaneously combined, others may require spontaneous installments or preceding approaches. For example, the steps taken in designing the Real-Time Camera can be combined with those taken in rendering it through the User Interface. However, the Game Design has to precede the Level Design.

### *3.1 Game Design*

To develop this game, CryEngine 3, Unity 4, Shiva 3D and Unreal Development Kit (UDK) were compared to choose the most suitable engine for real-time strategy. At the end of the analysis, UDK was selected based on availability of previous work and system capabilities.

The UDK is a game engine developed by Epic Studios and it is the free version of Unreal Engine 3. A previous real-time strategy game built using UDK is Hostile Worlds and this project will combine elements from that game. A number of assets including models, scripts and tutorials are required. These were be purchased through modeling websites like Mixamo and TurboSquid and/or developed using 3D applications like Fuse, 3DS Ma, Maya and/or Blender. After purchase, the assets are then imported into the UDK for implementation.

There are a lot of permutations in terms of AI for real-time strategy. Programmers are known to use algorithms like A*. But for this project, the explicit use of path nodes and navigation points are required. Unreal Kismet, a visual scripting tool in UDK, and UnrealScript, UDK's proprietary programming language were used to provide the artificial intelligence features in the game. The cinematic and animation sequences for the game were done in the Unreal Development Kit using Unreal Matinee and Unreal Kismet. Unreal Kismet is a built-in application in Unreal Development Kit that allows developers to design games using visual scripting. Unreal Matinee can read the nodes in Kismet and use them as triggers to play the sequences.

Leveling-up is and RPG feature which was incorporated in the game. The algorithm is precise and simply indicates that when the player reaches a certain pre-defined score, his level increases and his 'stats' also increase. His stats may include the population of his army, the strength of his army and his base and others. Leveling-up is an excellent way to increase the play and replay value for the game. The score system will be dynamic and will be triggered by different actions in-game such as defeating monsters, collecting artifacts and deploying units. When this increases to a certain level, the player may level-up.

For the user interface, Scaleform was extensively used in the design. The workflow is as thus: sketches of the concept design of the interface were rendered and cleaned in Photoshop and then imported into Flash Professional integrated with Scaleform for animation. Real-time strategy games are usually played using a combination of the keyboard and mouse and that method will also be followed in this project.

### *3.2 Resource Management*

Economy is rather important in real-time strategy. It is possible for the player to upgrade his army by using resources which will be available either at specific points or throughout the game. The primary resources in the game are shards, which may be used to purchase units. The game allows for the use of various strategies to be used as regards economic development. For the long-run, it is possible to develop heavily in economic growth while leaving military weak at first only to balance this out in the end-game. This strategy allows the player to have excess resources which are always useful in a real-time strategy game. Another strategy is to develop both economic growth and military strength side by side. Here, the player has just enough resources to either build an army or to develop infrastructure but not both. Players are advised to experiment and come up with their own strategies as well.

### *3.3 Battle Systems*

Real-time strategy veterans always have various techniques for battle. One such method may be to use a combination of different types of soldiers to try to achieve a balance. Another may be to invest heavily in just one specific rank and upgrade that rank to the maximum. Also, defense systems such as turrets and towers may be included in the game to increase the options for military tactics in-game. Deploying and upgrading units require the use of resources and the amount of resources required may vary, depending on the type of unit and its rank. The units deployed by the player follow scripts written in Kismet and UnrealScript that allow the units to open fire on enemies, gather resources and protect the base. Navigation points and path nodes are also developed for the game.

### *3.4 Real-Time Camera*

The real-time aspect of the game requires a camera that delivers a top-down or isometric view of the game. This will be one of the first problems to solve in developing the game as the default camera is usually a first-person camera or a third-person camera.

### *3.5 Level Design*

Level design is a complex process which requires numerous sketches and editing as well as experimenting within the editor to get it right. But Unreal Development Kit is a high-end engine that can deliver stunning environments in record time. Hence, designing and sculpting the level does not take too long.

### *3.6 User Interface Design*

The user interface is comprised of the menu screens that control the game mode and the Heads-Up-Display screens that show the player's status in-game. They were designed with Photoshop, Flash Professional and Scaleform. The flow for the user interface development is shown in Figure 1.

The heads-up-display interface can either be very simple or very complex, depending on the style or theme intended to be used. For this project, the HUD includes a mini-map interface, a shard interface, a score interface, an abilities interface and a status interface. The mini-map interface consists of height maps calculated using the levels created in the UDK and a cursor and a compass to show the location of the player and the locations of targets and objectives. The score and shard interfaces displays the player's scores and shards respectively. The abilities interface provides a means of access to the player's unit abilities and skills. The status interface shows the current status of the player's units in terms of health and ammunition.

The game modes were designed in a menu format, with each mode having its own menu and submenu. The front-end menu is the first menu that the player interacts with as it leads to the other menus. Next is the main menu which has the options menu, credits menu and game modes as sub menus. The options menu has its submenu composed of radio buttons, sliders and check boxes, and the credits menu simply rolls out the cast of the production team. The game modes are the join game and host game modes which are used to join or host a game respectively, which use the architecture provided in the original Hostile Worlds game.

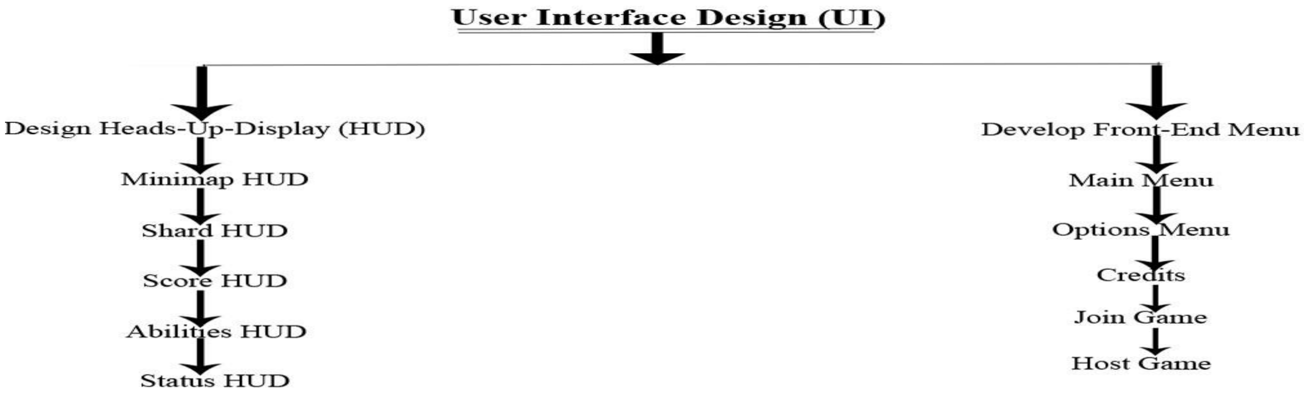


**Figure 1:** User Interface Workflow

### *3.7 Sound Design*

Background music and menu music will be composed and mastered in Fruity Loops Studio. New sound files may also be created in the studio. These music files will be saved with the '.mp3' extension and then imported into the UDK editor for final set-up in Kismet. In Kismet, sound cues are created to enable events to control the playback of music.

### *3.8 Gameplay Mechanics*

As illustrated in Figure 2, the gameplay mechanics of the game are the structural highlights in terms of gameplay value available for the game. Basically, the gameplay mechanics are what drive the core gameplay. Simply put, meshes are 3D objects in a game. There are two main types: Skeletal and Static. Static meshes are just that: they are meshes simply used for decorating the world and include bridges, pillars, and houses and so on. Skeletal Meshes like robots, humans, birds and even monsters, however, have bones attached to them which make it possible to apply generic animations to them to make them for interaction with the world. For this project, a number of skeletal meshes were imported into the editor for purposes of originality and diversity. These meshes replaced or supported the skeletal meshes provided by default in the editor. Once this was done, the new meshes will be integrated into the core gameplay of the game. This means that all scripts concerning artificial intelligence and navigation as well as animation were applied to the new skeletal meshes.

One of the many features of Unreal Development Kit is the ability to apply post-process effects to a player's camera, just before the scene gets rendered to the monitor. So essentially, all the calculations involving the pixels and everything else on-screen are already generated, and then just before they are displayed on the screen, the post-process effects are applied. Post-process effects may include color changes; blur level tweaks and depth-of-field changes. To create these effects, first a "Post-process volume previs" is created. This is a cuboid that envelopes the area the effects will be applied. Next, the depth-of field element is introduced. Depth-of-field controls the level of detail of an object that is displayed with respect to distance to that object. Finally, ambient sounds are added to the level and color changes are implemented.

During gameplay, the player may pick up certain artifacts that reward the player with shards. These shards may be used to purchase special upgrades for the player's units or economy. The algorithm written for the shard system was converted to script in Unreal Kismet or UnrealScript and this in turn was linked to the Heads-Up-Display interface that shows the amount of shards the player has.

Different units may possess different abilities in-game. There may be up to 4 different types of units, each with its own unique abilities and ability requirements. The script containing the intelligence for the abilities was linked with the Heads-Up-Display concerned with abilities. Abilities are what make a unit special, or unique. Some may have sustained abilities which, once triggered, remain active for a while until canceled. Others may be passive or active. Passive abilities are abilities that do not require activation in-game – they are ever present. Active abilities have to be activated to function and are usually instant.

The score system is dynamic and is affected by numerous triggers in-game. Depending on which event is triggered, the player's score may increase. For example, if the player defeats an enemy, captures an artifact, or protects a base successfully, his score may increase. The script for the score system was also linked with the Heads-Up-Display that shows the score.

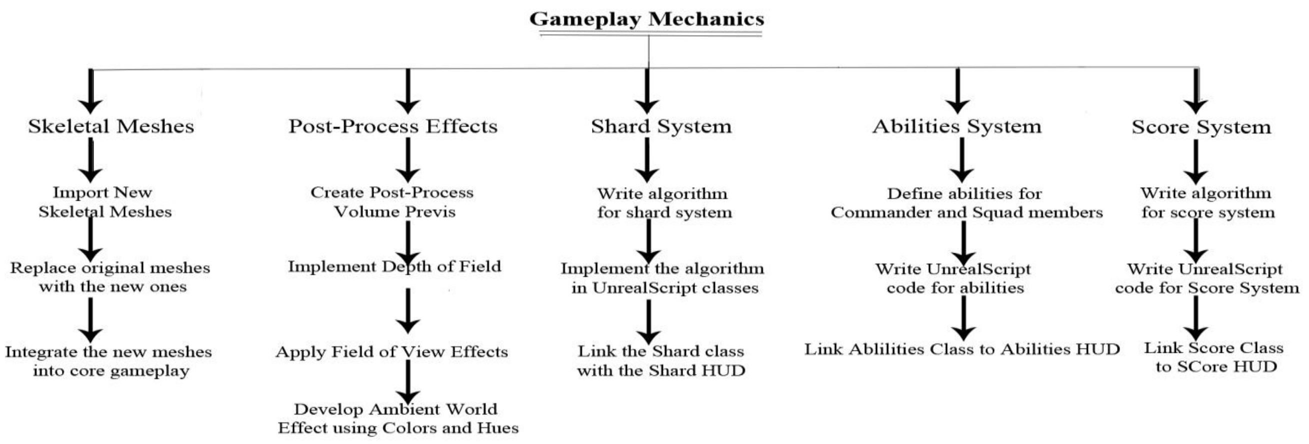

Gameplay Mechanics
Skeletal Meshes
Import New Skeletal Meshes
Replace original meshes with the new ones
Integrate the new meshes into core gameplay
Post-Process Effects
Create Post-Process Volume Previs
Implement Depth of Field
Apply Field of View Effects
Develop Ambient World Effect using Colors and Hues
Shard System
Write algorithm for shard system
Implement the algorithm in UnrealScript classes
Link the Shard class with the Shard HUD
Abilities System
Define abilities for Commander and Squad members
Write UnrealScript code for abilities
Link Ablilities Class to Abilities HUD
Score System
Write algorithm for score system
Write UnrealScript code for Score System
Link Score Class to SCore HUD

**Figure 2**: Gameplay Mechanics

## 4. SYSTEM ANALYSIS AND DESIGN

For this project, the game developed is named Hostile Worlds – Nemesis (HWN) and relies heavily on the architecture provided by Nick Pruehs and Marcel Koehler in designing the original Hostile Worlds game. However, HWN is re-designed in terms of the user interface, sound systems, level designs and some aspects of gameplay.

### *4.1 Level Design*

The level design process is one that involves a lot of planning and designing before actual representation is implemented in stages, as explained below. Drawing up the initial sketches, schematics and plans for the level takes quite a level of imagination. A lot of re-drawing and re-designing of the concept is required to produce the final schematic which becomes the blueprint.

Designing the terrain requires a lot of imagination and some skill. The terrain is set to be of the desert type, with a fog of war and complete lighting. Also, a 'Kill Z' is set, which is the point in 3D space with x, y and z co-ordinates that serves as a kill zone. An example of this can be seen in a game such as Darksiders II, where falling off of a cliff reduces the player's health to zero and forces a re-spawn. Decoration layers are added that can be used to generate random instances of a specified static mesh provided in the content browser. Contours, heights, slopes, ridges, bridges, paths, roads, boundaries and vegetation area are all sculpted on the terrain.

Vegetation for the level is sparse as the terrain is of the desert type. There may be the occasional oasis. As in Hostile Worlds – Nemesis' predecessor, Hostile Worlds, the terrain is mainly sandy and is of the desert type. However, in HWN, sub-terrains that are vegetative are provided. These terrains can be teleported to and contain lush forests.
Factors like wind and sunlight are also provided in the Actor Classes section in the content browser of the UDK. Trees, as well as static objects and skeletal meshes can be affected by these factors directly. Sunlight is generally provided by the Dominant Directional Light type amongst the four main types of lights including the Sky light, the Point light, and the Spot Light.

A number of static meshes are added to the level to increase detail. These static meshes have their detail modes set to 'low' to reduce computational time by the engine and also to conserve memory. Some of the meshes used include crash-landed UFOs, radio towers, barrels, metal chunks, statues, bridges, machines and wires.

Applying materials to static meshes and the terrain is not so difficult. The difficult part is designing the materials to be used. First, these materials are designed in Photoshop and then imported into the UDK. Or they are created using the UDK's in-built Material Editor.

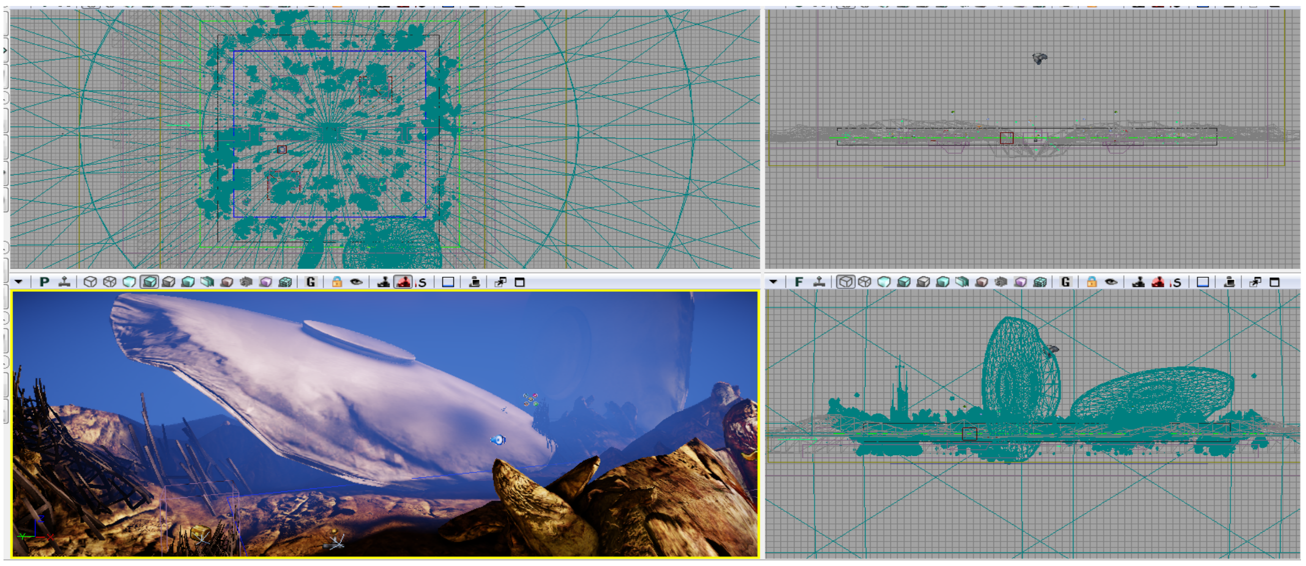


**Figure 3**: Level Design (Blueprints).

Volumes are drawn using the BSP brush in the UDK editor. The BSP brush is the brush used to create any kind of shape that can be made into a solid 3D shape or used as a volume. Volumes are of many types including: Blocking, Lightmass Importance, Post-Process, Color Scale, Water, Space, Lava, Kill Z, Reverb, Force-Field, Physics, Portal, Environment, Gravity and many more. However, the volumes concerned with this project are the blocking, lightmass importance, reverb and post-process volumes. The blocking volume is used to prevent the game camera and the player's controlled units from accessing a specific area bounded by the volume. The lightmass importance volume is used to calculate the light requirement computation of the level with respect to each object that can be affected by lighting. It is also used to compute the shadows of each object affected by the lighting of the level. The reverb volume is used to provide a reverb effect to ambient sounds throughout the level and is mostly concerned with sound cues and background music. The post-process volume is a special volume that calculates unique on-screen effects such as the degree of focus of an object, the bloom level of an object, the saturation of the screen, ambient occlusion and the motion blur amount as well as the scene's highlights, mid-tones and shadows.

### *4.2 Gameplay Mechanics*

The way the game is played and the mechanisms put in place to facilitate the way it is played make up the gameplay mechanics. There are quite a few areas to be developed and these include the way the player is spawned into the game, the artificial intelligence of the player's units, the artificial intelligence of the enemy units, the efficient utilization and accumulation, as well as allocation of resources, the voice-over and announcer systems, the use of teleportation portals in the game, the real-time camera, the way the fog of war reacts to the player's movements, the game programming in Kismet and UnrealScript, and how the units' abilities are referenced in the game.

Spawn points in the game are of two types: Player Start and Spawn Points. The player start introduces the player into the game in real-time in 3D space. The spawn points are used to bring the player's commander unit 'back to life' when said unit perishes in combat or when the player terminates the unit voluntarily. There is usually a cool down before the unit can be re-introduced into the game and the spawn points are the only places on the map that the unit may be re-spawned from.

The artificial intelligence for this project is divided into six layers: The Object, Actor, The Controller and the AI Controller, The HWAI Controller, and the HWN Kismet Sequence. The Object class is the base class all objects. It is a built-in Unreal class and it shouldn't be modified by mod authors. It is found in the core file of the sources folder under development in the UDK hierarchy. It contains all the constructs, functions, methods and procedures, as well as subroutines for all objects in the game and it is the file from which all other object related files are extended.
The Actor class is the base class of all actors and gameplay objects. A large number of properties, behaviors and interfaces are implemented in Actor, including: display, animation, physics and world interaction, making sounds, networking properties, actor creation and destruction, actor iterator functions and message broadcasting. It extends the Object class directly.

Controllers are the base class of players or AI. Controllers are non-physical actors that can be attached to a pawn to control its actions. Player Controllers are used by human players to control pawns. Controllers take control of a pawn using their Possess() method, and relinquish control of the pawn by calling UnPossess(). Controllers receive notifications for many of the events occurring for the Pawn they are controlling. This gives the controller the opportunity to implement the behavior in response to this event, intercepting the event and superseding the Pawn's default behavior. The Controller class extends the Actor class.

AI controllers are the base class for AI. AI controllers implement the artificial intelligence for the pawns they control. It contains parameters for controlling the movement of pawns to auto-adjust around corners, setting the difficulty level of pawns, getting the team index, resetting actors to their initial states (used when restarting level without reloading), enabling fire modes and weapons on pawns and so on. The AI Controller class extends the Controller class.

The Hostile Worlds AI Controller controls all the pawns of Hostile Worlds, receiving and carrying out basic orders. The Hostile Worlds source files are included in this project without duplication or modification. This is because the Hostile Worlds pawns extend properties of the engine's core source files, like the Object class. However, in terms of utilization, modification is made regarding the availability, forms, and presence of abilities, weapons, skeletal meshes and artificial intelligence these changes are culminated under the Hostile Worlds Nemesis sequence. The Hostile Worlds AI Controller extends the AI Controller, and contains parameters such as the number of frames a pawn is considered stuck moving if the distance from the target does not change, the destination location of the actor's current order, the target unit of the actor's current attack order (the attack order is defined in the class file and a link to it is provided in the ActionScript user interface files for interaction and it is stored temporarily in the engine's memory before implementation), the ability the actor has been ordered to use, the previous location of the unit, and so on. Since these files have already been conscripted by Nick Pruehs and Marcel Koehler in the production of the original Hostile Worlds game, it would be in-efficient to waste time re-writing these files. Hence they have been included in the development source files of Hostile Worlds- Nemesis as is.

While Kismet may be a visual scripting tool, it is just as powerful and efficient as writing the code in a source file in UnrealScript. The only major differences are: the files written in UnrealScript are called directly by the engine on startup while the sequence generated in Kismet only starts up when the level is loaded; and UnrealScript involves traditional code writing while Kismet uses nodes and targets to create a network of events and triggers. The Hostile Worlds – Nemesis sequence contains specific triggers for specific events in game such as inclusion of cinematic sequences using Unreal Matinee. It also contains specific animation sequences and triggers for events in-game.

### *4.3 Resource Allocation and Utilization*

Resource allocation is primarily implemented in the Player Controller class by declaring a parameter that stores and initializes the number of shards available to players at the start of the game. While the amount of resources allocated to the player at the start of the game is constant, the player may accrue more by defeating enemies and gathering artifacts and securing bases.

Resource utilization is also implemented in the Player Controller class by declaring a parameter that measures the amount of resources that is spent on purchasing units and using abilities. Parameters that apply cool downs to abilities, and parameters that control the maximum amount of units a player can possess may hamper resource utilization, but this is balanced it adds the strategy element to the game.

### *4.4 Voice-Over and Announcer Systems*

The voice-over systems are integrated into the Player Controller class and are extended for each of the various types of units available in the game. The voice-over system enables the units to provide auditory feedback to the player when an order has been issued. They also provide feedback when the units perform actions based on their AI. The announcer system uses the voice of an English female to provide status reports of the events occurring in the game in real-time and also to provide status reports on the status of the units. The announcer system is defined in the game class and is enabled when the game level is loaded. The actual sound files used by the voice-over and announcer systems are included in the UDK's content browser and are of two types: sound files and sound cues.

### *4.5 Teleportation Portals*

One of the defining characteristics of Hostile Worlds – Nemesis is the introduction of a portal system that allows the 'teleportation' of units and enemies between any two points in 3D space. The portals are included in the content browser and the mechanism for their functioning is quite simple. The first portal is set to direct whatever passes through it to the second portal by using URL that is simply the second portal's tag. The portal's tag is what identifies the portal as an instance in the command console of the engine.

### *4.6 Real-Time Camera*

The traditional camera provided in the game is the first-person camera. The secondary camera provided is the third-person camera. The former is usually used for First-Person Shooter games and the latter is usually used for Role-Playing games, but can also be used for Third-Person Shooter games. Since a real-time camera was not included in the engine by default in UDK, the real-time camera was developed by Marcel Koehler. It extends the Camera class provided by Epic Games and is fully scrollable. It contains parameters such as the maximum distance of the camera from the terrain, the pitch angle of the camera, the yaw angle of the camera, the roll angle of the camera, the camera's rotation speed, parameters that measure whether the camera is currently being scrolled or not, and parameters that identify which input device (mouse or keyboard) is being used to scroll the camera. Again, re-writing or re-coding this file is in-efficient as it adequately addresses all the highlights of a real-time camera, and hence is used as is.

### *4.7 Fog of War*

The fog of war is implemented in two ways: by using a Fog of War component in the Actor class and by using a Fog of War manager that manages the correct rendering of all fog of war volumes, revealing and hiding all units (friendly and enemy) as appropriate. It extends the actor, and contains parameters that identify the current map to be affected, the mask that tells where the player has vision, the index of the team the player rendering the fog of war belongs to, the textures used as opacity parameters for the fog volumes, the textures that indicate 'no vision' for a particular height level, the colour that indicates 'no vision' on the fog of war texture and so on. As the player moves on the terrain, the fog of war clears to show hidden units and objects that were clouded by the fog. The radius of the fog is constant but can be modified in the components panel of the editor. The fog of war may also be turned off completely by using the button that corresponds to the fog of war in the mini-map interface.

### *4.8 Abilities*

The abilities in Hostile Worlds – Nemesis are written in UnrealScript and the base Ability class extends the Actor class. The abilities include cloaking, artillery support, electro-magnetic pulse mine and grenade detonation, acquisition of artifacts, shot aiming, air strikes, charging the enemy, recharging, repairing, scanning, and focusing of fire. Abilities can targeted at units, locations or have an area of effect on units and/or locations. All subsequent abilities extend the Ability class which extends the Actor class. Parameters included in the Ability class are the name and description of the ability, the cool down period before the ability may be re-used, the range of the ability, the number of shards required to activate the ability, whether the ability has been unlocked, the unit the ability belongs to, the error message to show when the player tries to use an ability that is still on cool down, the sound to play when the ability is triggered, whether to display a radius for the ability or not and so on.

### *4.9 User Interface Development*

The user interface is made up of the heads-up-display and the game modes. The development of the interface is done mainly in Flash Professional and Photoshop. Initial concept designs are drawn that outline the schematics of the interfaces and then the individual elements are designed in Photoshop and then imported into Flash Professional for animation, using Scaleform CLIK and 3Di (IJCSIS, 2014). The completed files are then imported into the UDK for integration. To enable interaction, Mouse Event and Keyboard Event functions are called from their respective class files. These function correctly in the standalone '.swf' flash files. However, to enable interaction in the game, the 'fscommand' command is used to connect triggers to events directly in the engine's console.

The heads-up-display is the interface that the player directly interacts with during gameplay. It is made up of the status, mini-map, abilities and the shard and score displays. The programming for the heads-up display is executed in Flash Professional's proprietary language, ActionScript. There are class files in UnrealScript that link to the ActionScript class files to enable efficient pipelining.

The game modes are the Join Game and Host Game modes and the menus include the Options menu, the Credits menu, the Front-end menu and the Main menu. The Front-end menu houses the main menu which houses the options menu, credits menu and the game modes. The options menu is made up of radio buttons, checkboxes and sliders that can be used to affect each parameter such as difficulty and map selection differently. When these buttons are tweaked or selected, the corresponding "fscommand" is called in the game engine and the corresponding event is triggered. For example, if there was an option to enable screen desaturation in the game, the "fscommand" would link it to the corresponding screen desaturation variable in the Post-Process volume component window and translate the changes directly.

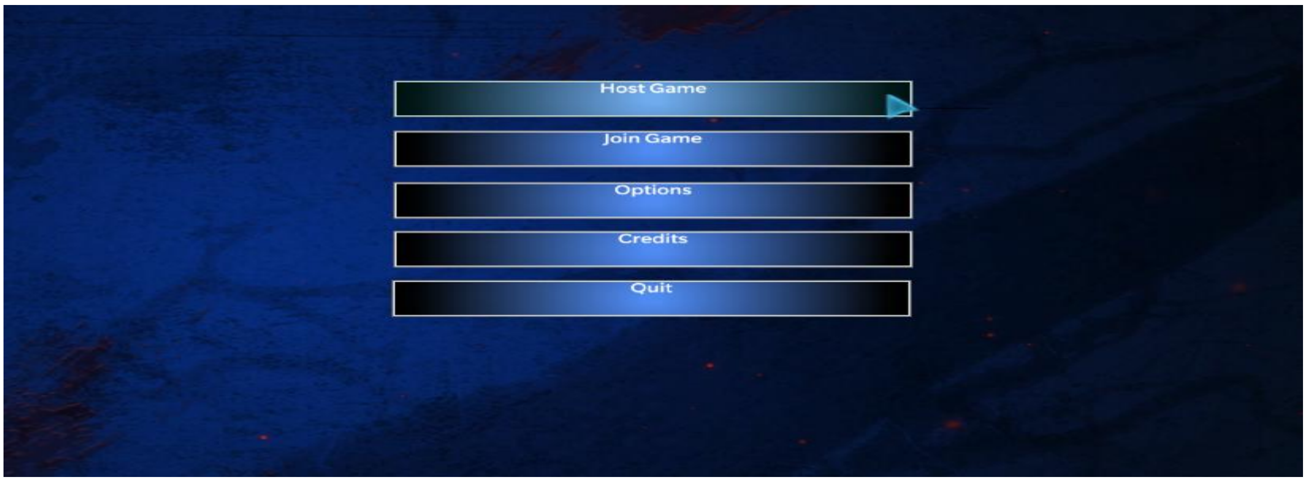


**Figure 4**: Main Menu User Interface.

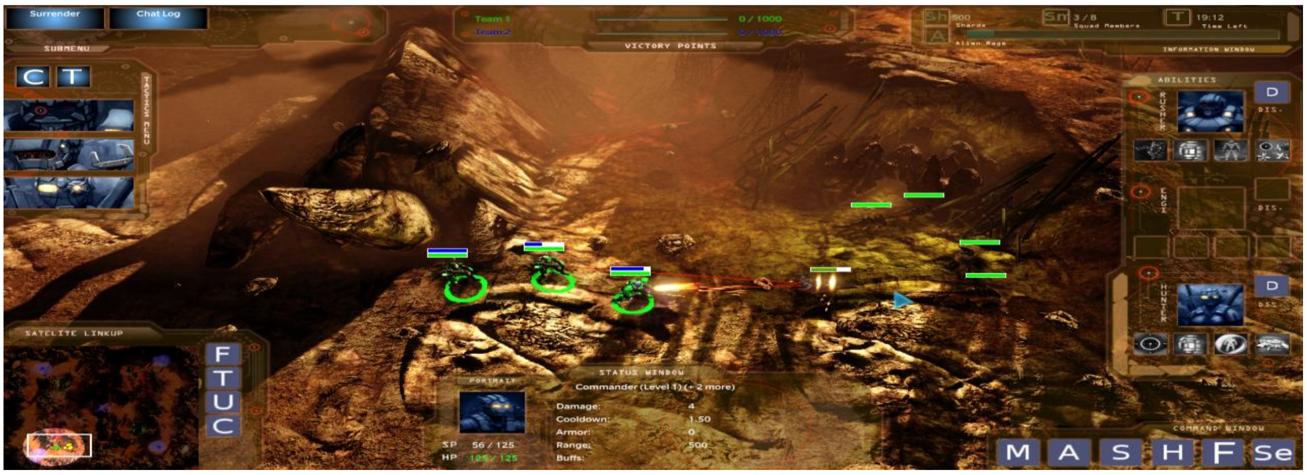


**Figure 6**: Heads-Up-Display Interface.

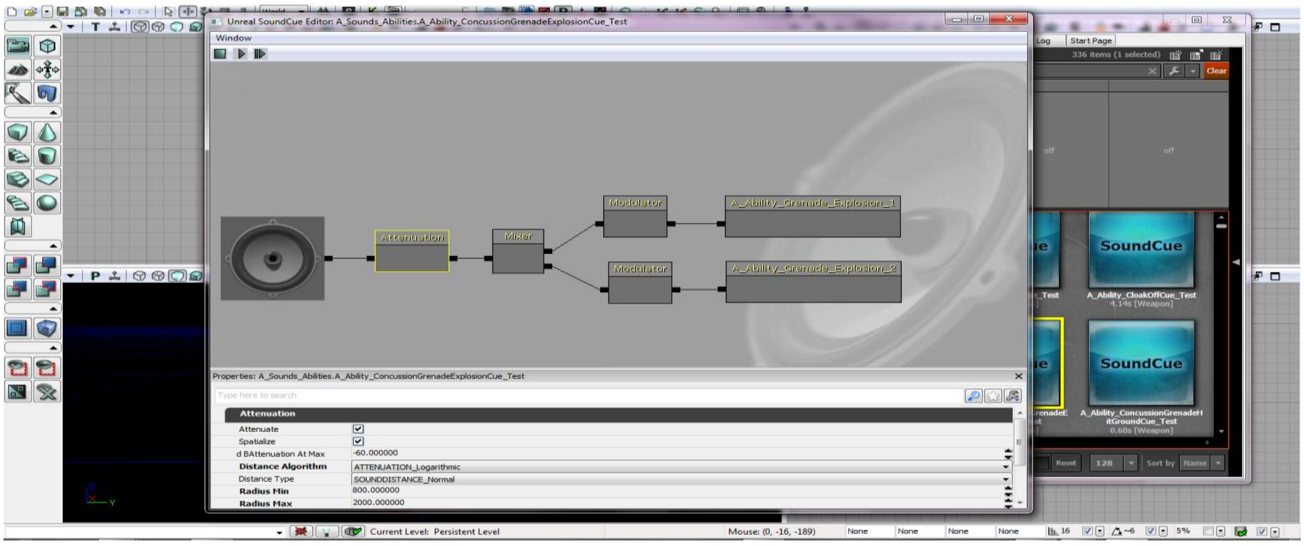
### 4.10 Sound Design

Designing and developing the sound files for the game is a complex process that actually starts with conceptualization of what the atmosphere in the game should ‘feel’ like. The inclusion of the Reverb Volume in the game allows direct manipulation of any sound file introduced into the game at runtime. The reverb effect, along with every other kind of effect can also be produced in Fruity Loops Studio. A recommended alternative for sound producers who cannot acquire or use FL Studio is Adobe Audition, although Audition is not free either.

FL Studio is the audio editing software of choice used for this project because of its complexity, efficiency, effectiveness and robustness. For the voice-over and announcer systems, the recording feature is used to produce high quality audio for feedback. The background audio is mastered in FL Studio using a combination of the playlist, step sequencer, piano roll, browser and the mixer. The playlist contains all the patterns, audio clips and automation clips included in the soundtrack. The step sequencer contains the individual elements and instruments imported from the browser used in creating current the patterns and audio clips. The browser contains a list of all the instruments, sound files, audio clips and elements that could be used in FL Studio. The mixer is used to master the beat and contains slots for the application of effects such as reverb, panning, volume, compressor, equalizer, filter, mute, stereo enhancer amongst others.

The sound cues are unique audio files because they contain both audio clips and programming that direct how effects are applied to the clip. The programming elements affect the use of attenuation, concatenation, delay or latency, cross fading, looping, modulating, mixing, oscillating, randomizing and the sound node wave parameters. Interesting effects can be created by tweaking the settings of these elements and by combining two or more in various ways. The possibilities are endless. Sound cues are used in HWN to create an otherworldly feel in-game and it functions appropriately.

Background creates a ground music in HWN is ambient and also has the reverb effect applied. This creates a sense of danger at all times during gameplay and ensures a heightened level of alertness from players. Ambient music includes forms of music that put an emphasis on tone and atmosphere over traditional music structure or rhythm.


Unreal SoundCue Editor: A_Sounds_Abilities.A_Ability_ConcussionGrenadeExplosionCue_Test
Window
Attenuation
Mixer
Modulator
Modulator
A_Ability_Grenade_Explosion_1
A_Ability_Grenade_Explosion_2
SoundCue
A_Ability_CloakOffCue_Test
SoundCue
A_Ability_ConcussionGrenadeHitGroundCue_Test

Properties: A_Sounds_Abilities.A_Ability_ConcussionGrenadeExplosionCue_Test

| Attenuation | |
|---|---|
| Attenuate | ✓ |
| Spatialize | ✓ |
| d BAttenuation At Max | -60.000000 |
| Distance Algorithm | ATTENUATION_Logarithmic |
| Distance Type | SOUNDDISTANCE_Normal |
| Radius Min | 800.000000 |
| Radius Max | 2000.000000 |

Current Level: Persistent Level    Mouse: (0, -16, -189)

**Figure 7**: The Sound Cue Editor in UDK.

## 5. CONCLUSIONS

### *5.1 Conclusions on the Game Engine – UDK*

The Unreal Engine 3 offers the most features of all found engines that are relevant for this project. The compatibility of these features is thoroughly proven by the community; we don't run the risk of having to use unverified and unpredictable 3rd party components. Unreal has a very big community and offers a wealth of knowledge that developers can rely on. Even if the Unreal Engine 3 might be cumbersome at first, it is a great opportunity to really get to know this state of the art game engine and explore it to its core. Another point not to neglect is UDK's revenue share model, which provides free access to this great game engine on development – payment is only necessary if the game results in a financial success later on.

### *5.2 Conclusions on the User Interface and Level Design*

The main visual style is semi-realistic with high quality hand painted textures and low-poly models like in StarCraft. The color scheme for each level is not very rich but saturated and contrasted with a lot of black. All units the player can interact with are distinguished from the background. This way, a very dark, mysterious, dreary, haunting and dangerous atmosphere is achieved.

The main level shows a red planet with a very bright sun. The color scheme is mostly yellow and red. It feels like a dusty desert where a human cannot survive without protection. The dark plants and architectural structures create an even more dangerous feeling. There is not much architecture on top of the surface, because this is more of an outpost for the aliens and all life forms live under the surface or in caves.

### *5.3 Conclusions on Gameplay*

Hostile Worlds – Nemesis allows the collection of artifacts to gain victory points. The first player with **1000** victory points wins the game, depending on the amount of points set before the game commences. Enemies can be killed to gain shards and up to eight squad members may be in the player’s army at any time. The first three squad members are free. After these initial three are spawned, each following unit costs 450 Shards. Each squad member has up to four abilities which can be unlocked by promoting it or upgrading it. Promoting squad members costs shards and increases the units’ levels (better stats). New abilities can be learned and all squad members automatically attack with their standard attack if close to an enemy unit. If the Commander is killed, he may be resurrected after 15 seconds by pressing the appropriate button and choosing a spawn location (green areas on the mini-map).

In conclusion, Hostile Worlds – Nemesis excels in all areas of a real-time strategy game, including gameplay design, level design, programming, user interface design and sound design.

### *5.4 Recommendations*

The Unreal Development Kit, Flash Professional and Photoshop were used extensively in the production of Hostile Worlds - Nemesis. The UDK has a lot of tutorials on game development but lacks essentially in the real-time strategy department. A lot more tutorials would go a long way to assist both newbie developers and veterans alike.

Adobe Flash Professional is a very powerful tool for user interface development. However, a bit more personnel support is required. Otherwise, it is the complete solution to user interface development. Autodesk Scaleform Gfx, integrated into Flash, made it possible to develop stunning user interfaces that always leave room for even more improvements. Scaleform Gfx, combined with Flash, is the ideal solution for user interface development.

Photoshop, another product from Adobe, is the world's most robust image-editing software in existence. It has a very wide array of tools and options to create virtually anything as regards images. Photoshop CS6 also supports video and sound, as well as 3D- model editing, although Adobe After Effects is better suited to video editing. In any case, Photoshop is recommended to anyone who wants to literally "bring ideas to life", as it is powerful enough for that not to be an understatement.

## 6. REFERENCES

[1] Christopher E. Miles, "Co-evolving Real-Time Strategy Game Players", ProQuest, pp. 7-10, USA, 2007.
[2] William Cassidy, "Utopia", GameSpy (Hall of Fame Article), USA, 2004
[3] Alan Satori-Angus, "Cosmic Conquest", Byte Magazine, USA, 1982.
[4] Dan Adams, "The State of the RTS", IGN Magazine, 2006.
[5] Lindsay Fleay, "The Historical RTS List", The RTSC Games List, USA, 2011.
[6] Matt Barton, "The History of Computer Role-Playing Games Part 2: The Golden Age (1985-1993)", pp. 3, USA, 2007.
[7] Dani Bunten Berry, "Game Design Memoir", USA, 2007.
[8] Kurt Kalata, "So What the Heck is Silver Ghost", Hardcore Gaming 101, 2010.
[9] Levi Buchanan, "Top 10 Renovation Games", IGN Magazine, USA, 2008.
[10] Anoop Gantayat "Sega Ages: Gain Ground", IGN Magazine, 2004.
[11] Ian Erickson, "Herzog Zwei", IGN Entertainment: Classic Gaming (Genesis - Game of the Week Article), USA, 2012.
[12] Scott Sharkey, "Hail to the Duke", 1UP Magazine, USA, 2011.
[13] IGN, "Amiga Reviews: Battlemaster", ZZap! Issue 68, pp. 45, USA, 1990.
[14] Bruce Geryk, "A History of Real-Time Strategy Games", GameSpot, USA, 2006.
[15] Blizzard Entertainment, "StarCraft's 10-Year Anniversary: A Retrospective", Blizzard Entertainent, USA, 2008.
[16] Interactive Entertainment Today,"Top 100 Games", Edge, USA, 2007.
[17] Boba Fatt, "The 52 Most Important Video Games", GamePro, USA, 2007
[18] Andrew Park, "The Greatest Games of All Time", GameSpot (PlanetScape: Torment), USA, 2005.
[19] Greg Kasavin, "StarCraft Strategy Guide: The Protoss Conclave – Units and Structures", GameSpot, USA, 1998.
[20] Greg Kasavin, "StarCraft Strategy Guide: The Zerg Swarm – Units and Structures", GameSpot, USA, 1998.
[21] Greg Kasavin, "StarCraft Strategy Guide: The Terran Dominion – Units and Structures", GameSpot, USA, 1998.
[22] Blizzard Entertainment, "Blizzard Support: StarCraft", Battle.Net, USA, 2008.
[23] Prima Fast Track Guide, "StarCraft – StarEdit Tutorial", CreepColony, USA, 2008.