



**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SPECIFICATION DOCUMENT FOR X-WARZ

**WONG SHANJUN SERENE
SORNUM KABILEN KADIRVELEN
DUBEY ROHIT KUMAR
DM6127 Introduction to Games Design**

TABLE OF CONTENTS

CONCEPT	3
CONCEPT ART	3
GAME PLAY.....	3
ART ASSETS.....	4
<i>3D Geometry.....</i>	<i>4</i>
Models.....	4
Sprites.....	5
Terrain	7
<i>Skeletal Animation.....</i>	<i>7</i>
ENVIRONMENTS	7
<i>Layering</i>	<i>10</i>
SOFTWARE AND TECHNICAL.....	10
<i>3D engine.....</i>	<i>10</i>
<i>Behaviours.....</i>	<i>10</i>
Pathfinding	12
Collision detection	13
<i>Executable.....</i>	<i>13</i>
X-WARZ GAME COMMANDS	14

CONCEPT

The game X-warz is inspired by the popular game Counter Strike. Similar to Counter Strike, X-warz is centered around a mission objective and to eliminate an opposing force. The process of conceptualizing is critical to the design of computer game, X-warz. Formulating ideas, conceiving possible bottlenecks, and abstracting features from existing games in the market is the crux in both developing and marketing the game. The process of conceptualizing involves data modeling, algorithm formulation, version control, and game testing.

The challenge in conceptualizing is for the team to find some common ground of expectation from which the game application can be developed, and what are the potentials and bottlenecks. Some extent of shared conceptualization among the team is necessary to facilitate refinements of the game.

CONCEPT ART

Concept art was used to previsualize art representation of characters, health packs, ammunition, landscape for referencing color compositions and creating the mood before it is implemented in the actual X-warz game. The team came up with draft images which illustrate the look, feel, design, colors of the game to be developed.

In the course of creating these art representations, the team addressed some technical pitfalls that would otherwise have arose during the development phase. For instance, X-warz gives the player a choice of the gun models to handle different enemies. This would, of course, enhance the overall playing experience; but it is also very costly in terms of the polygon count to create so many variations of the ammunition, enemy armor, and firing effects.

Therefore, by creating drawings of the environment, characters, color and lighting before the actual X-warz development gives the development team a clear sense of direction that the game is feasible. It helps to save time, effort and money, and paves the path for a quality X-warz game application.

GAME PLAY

X-warz is a tactical second-person shooter computer game. X-warz pits the wits of a valiant soldier against a group of enemies in an unfamiliar territory. His mission is to find a sword in the terrain.

Since this is a second-person shooter game, both the player and the opposing force can fire shots at the other party. X-warz is programmed with two types of enemies, and they are placed randomly to maneuver around the territory on each run of the game

There is no floor plan provided to the player in his course of finding the sword. In addition, he has to locate health packs and ammunition to sustain his combat. This adds to the complexity, which makes the playing more exciting for the player. Over layering the background music, is the impending sound and special lighting effects of danger approaching, this helps to heighten the suspense.

Some twists to the game are, for instance, the sword is hidden from our normal vision and the player has to figure out the location of the sword in an unfamiliar territory. After which, the player has to work out how to pick up the sword without getting killed by the enemies.

ART ASSETS

3D GEOMETRY

There are three agents in X-warz, namely the first-person player, enemy A and enemy B. Each of these agents is constructed using primitive-modeling in MED Model Design. X-warz uses MED to create, texturize and animate simple models as its agents.

The Lite-C game engine is able to render several types of 3D entities. These entities can be created by script, or placed in the level by WED World Editor Program. The following types of entities are supported in X-warz:

Models

The entities for the first-person player, enemy A and enemy B (see below) in X-warz are known as models. A model is an animated 3-D object, stored in an external MDL file, made with a 3D mesh with an underlying skeleton (bones) and a soft 'stretching' skin. These model entities are able to cast dynamic shadows and are used for animated objects. Each of the character models are created with a Model Editor Program MED and the model mesh can have different textures, material properties, and shaders assigned to different parts of this model.

To elaborate further, each of these models consists of a wireframe mesh, made of connected triangles. The corners of the triangles are known as vertices. Hence, the positions of the vertices define the shape of the mesh. For each triangle face in the mesh, a corresponding triangle is cut from the model texture image skin. The UV mapping technique then maps each three-dimensional vertex of a triangle that describes a XYZ position in space to a corresponding two-dimensional vertex that describes a UV position on the skin picture.



Sprites

X-warz also uses many sprites to build its environment and create special effects. A static sprite is a 2-D object that can be placed in the landscape or it can behave pseudo-3D by always facing the camera. It could also be positioned on a wall, floor or ceiling as a decal, for example, the lamp, door, or the posters on the wall shown below.



In addition to static sprites, X-warz also uses many one-shot animation sprites (these are animation frames which are played once and then stops). To create animated sprites, a series of single animation frames are positioned beneath each other in the image, and simulated as a number of consecutive frames in a horizontal film strip. The following shows some examples of animated sprites in X-warz:

1. The fading in and fading out effects of the introduction screen (the pictures are increasing in clarity)



2. The different firing effects (see the three diagrams below namely, the red beam ray, the small explosion with silver pieces of bullet, the big explosion with golden bullets) when the bullet is fired from the different models of gun using the left mouse click



3. The animation starting from how the enemy is shot, to the gradual dying process with the blood oozing out, followed by the body disappearing, as in the screenshots below



4. The red light circumference when the first-person player is shot by the enemy, and there is a red beam ray indicating the line-of-shot



5. The gradual darkening of the lighting when the first-person player dies (the picture on the right is darker than the picture on the left)



Sprites are stored as PCX, BMP, TGA or DDS files and can be created using a standard paint program. While TGA or DDS files can contain an alpha channel which gives a transparency value for each single pixel, only PCX, BMP, or TGA files can be animated. Typically, sprite entities are rendered faster than MIP maps, and can be used for fire

shots, lights, etc. MIP maps are pre-calculated, optimized collections of bitmap images that accompany a main texture. DDS files use the technique of mipmapping to produce better quality models and faster rendering.

Terrain

The warehouse landscape is composed of several textures mapped onto a rectangular grid. It is stored in an external HMP file. In order to maintain good resolution of the large warehouse terrain that is implemented in X-warz, a small detail texture has been used to blend a secondary high-resolution brick structure into the primary stone texture. An extension for the X-warz game is to use multiple detail textures for mapping patches of wood, stone, bricks, etc. onto different parts of the landscape and then using a shader to blend these three textures plus a shadow map for a more realistic terrain.

SKELETAL ANIMATION

The skeletal animation of the agents consists mainly of their walking behavior and the collapsing behavior after they have been shot. The first-person player can move around using key press “w” (forward), “a” (left), “d” (right), “s” (backward) and the spacebar (jump).

To simulate these skeletal animations, there are several possible options:

- 1) With a skeletal animation system, the bones can be defined using matrices and the vertices are influenced by a number of bones. When drawing this type of object, each vertex is transformed by its associated bones. The result is scaled accordingly and summed together.
- 2) Another method uses an invisible hierarchy of animating bones to move vertices of the mesh either directly or in weighted combinations.
- 3) Alternatively, key-frames can be generated using transformation routines and the resultant vertices are placed into a vertex buffer, for rendering.
- 4) Lastly, to linearly interpolate the vertices from one key frame to another.

X-warz simulates the walking behavior of the agents using bones animation, that is to rotate a bone of a underlying skeleton that the vertices are attached to.

ENVIRONMENTS

X-warz relies heavily on textures to establish the features, intricacy and realism of the combat scene. For texturing, a compromise exists between resolution and reusability. As X-warz requires real-time rendering, it is necessary to limit the amount of geometry involved. Therefore, texturing is used to compensate for the details that cannot be represented with polygons.

For instance, adding bloodstains to the wall (see the first texture below) can enhance the realism of the battlefield.



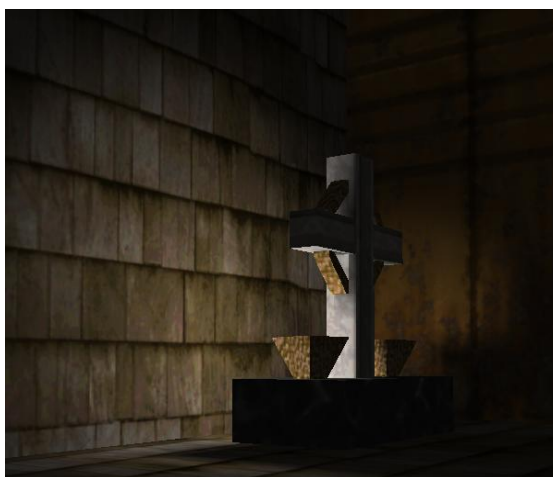
There are three other types of textures used on the walls of the different rooms in X-warz. Using different types of wall textures gives the impression of a larger landscape like the corridor below:



Although a box entity is constructed using 12 polygons and flat on the sides, X-warz adds variation by applying textures to it, to form complex objects like health packs, ammunition, as seen in the diagrams below.



Likewise, pseudo-complex entities can be created easily using textures, for instance the diagrams of an age-worn spiral staircase and the cross monument shown below.



With the memory limitations of real-time rendering in X-warz, there is a bottom line on the pixel counts for each object. For larger objects like the door (leftmost picture below), a higher pixel count is dedicated to it. However, for smaller objects that are relatively distant from the first-person player, lower pixel counts is needed to save memory. Instances of lower pixel count are the lamps and lights hanging on the ceiling or walls, as seen in the middle and rightmost pictures below.



LAYERING

X-warz exploits the use of multiple layers to composite elements in the scene. Each element is separated from the background and kept on its own layer. The allocation of entities to the different layers is as follow:

the health status bar and ammunition status bar belong to one layer;

the aiming circle and the first-person's gun form another layer;

the multiple explosions are from a subsequent layer;

the characters and environment on a different layer.

These elements, being superimposed on each other, it is easy to control individual elements and not affect the entire scene. Furthermore, each element can be color corrected, transparency adjusted, or transformed without affecting the rest of the image.

SOFTWARE AND TECHNICAL

3D ENGINE

X-warz is designed as to be a PC game. It is developed using Lite-C game engine. Lite-C contains:

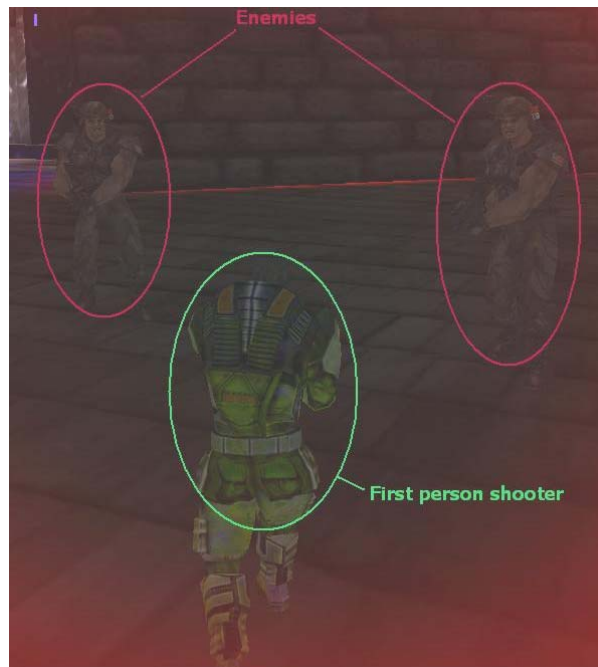
- MED, the 3D model and terrain editor
- SED, the source code editor, compiler and debugger
- A7, the virtual reality engine

The Lite-C engine requires an updated 3D accelerator and DirectX 9.0c or better.

BEHAVIOURS

Enemy A and B are designed to maneuver around the territory, and are placed randomly in the territory on each run of the game. For the first-person player, the agent is designed to maneuver around the territory and the enemies can fire shots at the other party. In X-warz, there are two kinds of enemies with diverse intelligence levels. This means they are configured with different parameters for their path-finding algorithm, wired to move at different speeds and have variant reactions to different ammunition. They are programmed to defend the sword from intruders, and to pursue and fire shots at the first-person character.

The following picture shows the first kind of enemy agents (circled in red) with the lower capabilities, and the first-person player agent is circled in green.



The second type of enemy agent is represented as the soldier in the black armour. They possess higher capabilities than the enemy agents above. Their shots cause a bigger reduction in health points than the first type of enemy agents.



Pathfinding

Path finding is the process of moving an artificially intelligent agent from one point to another. This protocol allows characters in X-warz to move around the landscape in a realistic manner while avoiding obstacles. The basic rule is that as long as the agent is not at the goal, they must choose a direction that is nearer to the goal and move towards it.

Different search algorithms and different techniques of determining intermediate waypoints or sub-goals are used to determine the agent's resultant action. For instance, for an enemy agent which is pursuing the first-person player around an obstacle, it might be best if they sneaked around the obstacle and try to snipe at the first-person, rather than directly attacking the first-person player within line-of-sight, since the first-person player can also counter-attack the enemy, or even worse charging into the obstacle in straight lines.

X-warz implements a basic path-finding algorithm in a pre-determined terrain. The program places a starting point for the first-person player, and a target destination for the sword. In addition, the program plants the enemy agents randomly on each round. The enemy agents try to move towards the first-person player goal, if it has a straight line of sight to it. If it does not have a direct line of sight, then waypoints are required for the agent to move around obstacles e.g. walls, monuments, etc.

The waypoints are used to set some points for a visible path-finding system where the game world is subdivided into smaller sections using specific points, so as to reduce the complexity of the world. The basic path-finding engine for the enemy agents in X-warz uses the A* search algorithm to find a path through the waypoints, whereby the points chosen as the next sub-goal are chosen based on a distance heuristic, for example, the distance between start and waypoint and waypoint and first-person player goal.

The waypoints have been added automatically into the X-warz program using the quick-hull algorithm. The quick-hull algorithm creates a convex hull around a set of points, where the outer most points of the set are used to form edges that contain all other points in the set within them. The algorithm positions the waypoints as follows: First, a line is calculated between the start position of the enemy and the first-person player target. After which, a line orthogonal to this line is placed at the midpoint of the first line. From the midpoint, it scans outwards in both directions away from the midpoint along the orthogonal line. When it finds a passable point, it moves out a bit further, and places a waypoint there. It then repeats this procedure from the start to each of these waypoints, and from each of these waypoints to the first-person player target. Finally, it repeats this procedure once again for every waypoint to every other waypoint. The result is a set of waypoints that tend to wrap around the perimeter of obstacles.

As the quick-hull algorithm produces a large set of waypoints, the search algorithm involving so many waypoints would take far too long to be done in real-time. The technique of spatial partitioning is used to reduce the number of possible waypoints to look at. In X-warz, a 2D mesh is used to represent the landscape; and waypoints to represent points for a visibility system, whereby the enemy agent can only maneuver from one point to the next if the agent has a line of sight from its current position to the waypoint. Hence the actual path finding algorithm is much simpler compared to the spatial subdivision required to make the search algorithm feasible.

Collision detection

X-warz's collision detection system uses the default OBB (oriented bounding boxes) collision system built into the Lite-C engine. Moving entities, like the first-person player or the enemies, use a simple ellipsoid hull; and non-moving obstacles like terrain or map entities use a polygon hull in their own shape. The size and shape of the bounding ellipsoids is essential for a proper OBB collision detection. By default, entities get an X/Y symmetrical hull in two standard sizes. This design helps to ensure the agents in X-warz fit through doors regardless of their real shape.

As X-warz requires its agents to perform collision gliding efficiently along walls and slopes, special attention is devoted to the relative forward motion and the absolute downward velocity for gravity. For example, when the agent approaches the spiral stairway and the "spacebar" is pressed, the agent is programmed to jump up. This is achieved by increasing the agent's forward velocity and reducing the gravity.

EXECUTABLE

The best example of the operation of the program can be seen by actually running it. An executable file (new_level2.exe) has been submitted.

X-WARZ GAME COMMANDS

Command	Action
W	Move Forward
S	Move Backward
A	Move Left
D	Move Right
Spacebar	Jump
Mouse Left Click	Fire
F7	Toggle between first-person perspective and third-person view
Q	Toggle between Gun Models
Esc	Displays Toolbar