Guardians of Neverwood

An action -adventure game for one or two cooperative players

Brought to you by StormForge Entertainment Group

TODO: STORMFORGE LOGO HERE

Rolf Hendriks * Matt Holden * Chris Winings

TODO: CLASS LOGO/NAME HERE

a division of 404 Name Not Found Productions

Table of Contents

Table of Contents

Section VI: Engine and Technical Design

Team Information and Division of Labor

Rolf Hendriks — Technical Lead

(407) 681-3500 rolfhendriks@earthlink.net

- **Rendering**
  - **Space partitioning**
- **Artificial Intelligence**
  - **Finite State Machine**
  - **Enemy and Falcon AI**
- **Collision Detection & Reaction**
- **Sound**
- **In-game Level Editor**

Matt Holden — Project Lead

(407) 405-1835 matt@mattholden.com

- **Input Management**
- **Debugging Tools**
  - **Memory Manager/Monitor**
  - **Debug Heads-Up Display**
- **Interface**
  - **Front-End Menu**
  - **In-game Menu**
  - **Heads-Up Display**
- **High-Resolution Timer**
- **Particle System and Editing Tool**
- **Story Implementation**

Chris Winings — Outsourcing Lead

(301) 996-3295 silverdrgn@juno.com

- **Animation**
- **Model Loading**
  - **Skin and Bones Animation**
  - **Combat Motions**
- **Gameplay Architecture**
  - **Combat Systems**
  - **Players and Enemies**
  - **Weapons and Projectiles**
  - **Powerups**

Outsourced

- **Original Content Creation (Models, Sound, etc.)**
- **Additional Content Gathering**

Executive Summary

Mission

StormForge Entertainment Group and its members are committed to creating a game that is fast-paced, professional in appearance and quality, and most of all, fun. We will ensure that *Guardians of Neverwood* is free from bugs, video glitches, and low frame rates that are characteristic of many amateur projects, and establish ourselves as professional game designers and developers as we close out our school careers. It is a secondary goal to enter *Guardians of Neverwood* in the IGF competition at GDC 2005.

Simultaneously, we hope to achieve a higher level of competence and confidence in the field of game development. As we have never created a 3D game, or a project of this scope in the past, we anticipate and welcome the opportunity to learn new and necessary skills that the industry will demand. We will do this in a team environment, relying heavily on each other for technical and motivational support as we work toward the goal of bringing *Guardians of Neverwood* into being in such a short time.

High Concept

For twenty years, the brutal hordes of Orkdursk have destroyed everything in their path. They've just chosen the wrong forest.

Setting

Guardians of Neverwood takes place in the forest Neverwood, a normally peaceful, if deep, forest that has been invaded by the orcish hordes of Orkdursk.

Goal

As Tempest and Aurexis, the player's goal is to rid Neverwood of the Hordes of Orkdursk. In a more complete version with additional development time, the pair would leave Neverwood in an attempt to dismantle the entire Horde.

Genre

Guardians of Neverwood is an action/adventure platforming title for one or two cooperating players.

Executive Summary

Primary Target Audience:	Fans of third-person adventure games Fans of platform games
Secondary Target Audiences:	Couples who play games together Team members' potential employers

Key Features

- Designed with women in mind, a rare distinction in action/adventure games
- Traverse a beautiful forest
- Two styles of play with Tempest, focusing on melee combat or stealth/ranged
- Two-player cooperation in split screen
- The ability to play as or to command a falcon
- Unlockables and easter eggs add replay value
- First-person mode allows for precision sniping with the bow
- Multiple types of arrows add damage and satisfaction to every shot
- Dexterity-based combat provides a new alternative to hack-and-slash games
- Refined combat motion ensures there is no unavoidable attack

Main Character Overview

Tempest Silvreleaf: a human female ranger. She is extremely agile and uses this dexterity in combat. Tempest prefers to dodge rather than block. Her primary weapons are a short sabre, with which she can both slash and thrust, and a longbow. No matter what she wears, she always dons a large leather gauntlet on her left hand. This serves as a landing perch for Aurexis, the hunting falcon with whom Tempest has traveled for the last two years. Tempest is controlled by Player 1 in *Guardians of Neverwood.*

Aurexis: a sleek, mottled brown hunting falcon that fights with Tempest. He serves as scout and support, attacking enemies from above with his talons and wings. He watches for enemies, traps, switches and important items, alerting Tempest to their presence. Aurexis can also retrieve items in his talons, either bringing them to Tempest or dropping them onto enemies from below. He also can snatch projectile weapons out of the air, protecting Tempest from rear attacks. In one-player mode, Aurexis is controlled by various AI modes (Aggressive, Defensive, etc.). In two-player mode, Aurexis is controlled by Player 2.

Executive Summary

Main Character Overview (continued)

Kara'tok: a field general in the Hordes of Orkdursk. He is a brutal, savage warrior who shows no mercy to friend or foe alike. He is a towering orc, standing 6'2" tall, and extremely muscular. He usually dresses in filth-crusted black hide pants and a bandoleer over his otherwise bare olive skin. Kara'tok fights with a massive double-bladed battle axe and a long, black steel claymore, able to wield each one-handed due to his colossal strength.

Sul'talan: a tamed raven owned by Kara'tok. Like his owner, he has no compassion for the living. Sul'talan feeds on decomposing flesh, and considers killing merely an opportunity to procure his next meal.

User Experience & Gameplay Example (Level One)

Tempest walks through the dense forest she has called home these last six years, carefully taking each step to avoid alerting the orcish guards. Sighting one on a hill beyond, she nocks an arrow in her longbow. Carefully aiming for the orc's throat, she releases the arrow and waits as it strikes home and drops the guard. Moving a bit faster now, she continues toward the center of the forest.

From above, Aurexis signals that there are guards approaching. He snatches an axe from the ground in his talons, dropping it on its unsuspecting owner. As the guards scramble to deal with the new threat in their midst, another of Tempest's arrows streaks through the air and downs another warrior. Charging from the wood, Tempest shoulders her bow in favor of her long sabre. She engages in a fierce battle with three of the orcs, dodging and rolling side to side with her amazing agility, striking with quick slashes and thrusts where she can, and blocking blades that get too close with her falconer's gauntlet. After dispatching the guards with Aurexis' help, she hops onto a rock in the river and continues on.

Target Platform

Guardians of Neverwood is being developed for the PC, however, its action feel and joypad control emphasis lends it well to console ports at a later date.

Guardians of Neverwood is intended for use on PC compatible computers satisfying the following:

Minimum system specifications:

- 1 GHz Pentium-compatible processor or greater
- 128 MB system RAM or greater
- Windows 98 or higher
- OpenGL-compatible video card with 64 MB video memory or greater
- Sound card
- 100MB hard disk space

Recommended system specifications:

- 1.5 GHz Pentium-compatible processor or greater
- 256 MB system RAM or greater
- Windows XP
- OpenGL-compatible video card with 128 MB video memory or greater
- Sound card
- 100MB hard disk space
- Two force feedback game controllers

Charter

Meetings

Team members will meet on Thursdays from 5:00 PM until 8:30 PM at Matt Holden's residence. If the team feels additional meetings are necessary, they shall be held from 3PM to 5PM on Mondays in Full Sail 4. Meetings can only be can celled unanimously. In addition, time in each class is devoted to the project wherever it is permissible by the course director and the workload. During individual work time, which will be at the discretion of each member with regards to the tasks scheduled them, we will remain in contact via email, telephone, and instant messages.

Decision-Making Process

The team writes down all acceptable solutions and eliminates them until a solution is found. The potential solutions are discussed until a consensus is reached. If a consensus cannot be reached, the deciding vote rests with the team member responsible for the module in question. For shared modules, the decision rests with the project lead.

Team Member Conduct

- Team members will make every effort to get the appropriate amount of sleep, food, oxygen, and all other physiological and psychological needs. Violation of this rule will result in the calling of the offending party's mother and/or wife.
- Beer and whine are not permitted at team meetings.
- No juggling fire in the house.
- Team members will use Rolf's cellular phone as sparingly as possible.
- Team members will arrive on time to team meetings and bring all necessary equipment to work, including documentation, computer and A/C adapter.
- No wearing shoes in Chris's apartment.
- No smoking at team meetings.
- The contents of Matt's fridge are public domain during meetings. Asking for permission to retrieve something from said fridge will cause said privilege to be revoked for a period of one (1) hour. This rule also applies to the restroom.
- Video games, television, and other avoidable distractions are forbidden during team meetings while other members are working. If a break is necessary, the entire team will take one together.
- Team members will make complete backups of their work each time a milestone or task has been made. This is in addition to copies stored on the CVS server and will be located on another PC or removable media.

Charter

Code Review Process

At the start of each Thursday meeting, each team member will give a brief overview of the week's work. This may include white board diagrams, printed or digitally distributed code, test applications, or any other appropriate means of demonstration available. Questions will be fielded after each team member's briefing.

Code issues, including noncompliance with code/comment standard, will be discussed by the entire group at the end of these presentations, if applicable.

Damage Control

If a situation is deemed to be a problem by the team as a whole, the following measures will be taken to resolve the problem as quickly as possible:

1. Don't panic.
2. Inform the team immediately if you're falling behind or feel you are in danger of falling behind.
3. Team members will make every effort to pick up the slack for team members who are falling behind or falling ill, unless the current project of the able team member is of a higher listed priority than the project of the ill/unable member.
4. Everyone works on one bug at one time during debugging phase.

Tempest Silvreleaf

Overview

Tempest Silvreleaf is a human female ranger. She is extremely agile and uses this dexterity in combat. Tempest prefers to dodge than block. Her primary weapons are a short sabre, with which she can both slash and thrust, and a longbow. No matter what she wears, she always dons a large leather gauntlet on her left hand. This serves as a landing perch for Aurexis, the hunting falcon with whom Tempest has traveled for the last two years. Tempest is controlled by Player 1 in *Guardians of Neverwood.*

Physical Description

Polygon Count: **2,500 to 4,000**

Tempest stands 5'5" tall. She has dusty blonde hair, which she wears in a braid that ends between her shoulderblades, and piercing blue eyes. She typically adventures in a dark green tunic and brown hide pants with her sabre at her side and a quiver over her shoulder, though she does change clothes often. Tempest is 22 years old. She is well-toned and well-proportioned throughout, especially her muscular legs, built for dodging and evading.

Gameplay Mechanics

Tempest has two main modes of play, which are entirely at the discretion of the player. The first mode is more melee-based, involving the player charging at enemies with her sword and bowshots. This is the more straight-forward approach and is geared toward fans of first-person shooters.

However, there is also a stealth element present in the game, to appeal to fans of such games as the *Thief* and *Metal Gear* franchises. In this mode, Tempest will nock an arrow and switch to first-person view. She will be able to more carefully aim her bow, enabling her to strike targets more precisely, as well as striking targets that are out of the range of her automatic targeting.

Statistics

Damage:	**Movement rates:**	**Maximum HP: 100**
sword thrust: 5 HP	**3 m/s walk**	**Hit recovery: .25 sec**
sword slash: 3 HP	**2 m/s swim *(optional)***	
regular arrow: 3 HP	**2 m/s climbing *(optional)***	
bow	**5 m/s running**	
charged arrow: up to 9 HP	**3.5 m/s jumping**	

Tempest Silvreleaf

Behaviors

Combat:

- **Slash with sabre**
- **Thrust with sabre**
- **Fire bow** (charge for more damage)
- **Pick up herbs to recover health**

Defense:

- **Dodge left and right**
- **Parry with sabre**
- **Block attacks with gauntlet**

Interaction with Aurexis: (1 Player)

- **Call Aurexis to her wrist**
- **Set Aurexis' AI tendencies**

Movement:

- **Jumping** (vertical and directional)
- **Walking and running**

Optional Movements:

- **Grab ledges**
- **Climb trees**
- **Wall/tree jumping**
- **Swim**

View:

- **Move camera position**
- **Switch between 1st/3rd person**

Default Controls

Action	Keyboard & Mouse	Joystick
Walk/Run	**W, A, S , D**	**Left analog stick**
Control Camera	**Mouse movement**	**Right analog stick**
Camera Default Position	**Page Up**	**R3 (depress right stick)**
First-Person View (hold)	**Control**	**Left shoulder 1**
Fire Arrow	**Right mouse button**	**Right shoulder 1**
Sword Slash **Activate Switch**	**Left mouse button**	**Button 3 (PSX X)**
Sword Thrust	**Middle mouse button**	**Button 1 (PSX Square)**
Jump	**E**	**Button 2 (PSX Triangle)**
Block	**Space bar**	**Button 4 (PSX Circle)** **Right shoulder 2**
Dodge	**Arrow keys**	**'Block' + left analog**
Pause	**Enter**	**Start**
In-Game Menu	**Escape**	**Select**
Toggle Aurexis' AI states	**Shift**	**Left shoulder 2**
Taunt / Comment	**T**	**L3 (depress left stick)**

Tempest Silvreleaf

Note: Tempest will be textured differently than in this photo.

Aurexis

Overview

Aurexis is a sleek, mottled brown hunting falcon that fights with Tempest. He serves as scout and support, attacking enemies from above with his talons and wings. He watches for enemies, traps, switches and important items, alerting Tempest to their presence. Aurexis can also retrieve items in his talons, either bringing them to Tempest or dropping them onto enemies from below. He also can snatch projectile weapons out of the air, protecting Tempest from attacks.

Physical Description

Polygon Count: **1,200 to 2,000**

Aurexis is a brown, sleek falcon. His beak is short and hooked like a buzzard's. His keen eyes are brown and never still. He has mostly short feathers, brown with splotches of a darker brown, and a few with black tips. Aurexis' razor-sharp talons are knobbly and brownish-gray. He wears no harness or bell to indicate domestication.

Gameplay Mechanics

In one-player mode, Aurexis is controlled by switching between his AI states. In two-player mode, Aurexis is controlled by the second player, and sees his field of vision in the top pane of the split-screen window. Whereas one-player mode locks Aurexis into one of the roles listed above, the second player can perform actions individually as desired.

AI States (1 player mode only)

Aggressive	**Attack nearest enemy whenever possible**
Assist	**Deflect attacks on Tempest, aid in jumping**
Search	**Seek out items to retrieve and switches**
Rest	**Perch on Tempest's shoulder and regain HP**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 75**
talon slash: 3 HP	**6 m/s flight**	**Hit recovery: .25 sec**
wing slap: 2 HP	**8 m/s diving**	
charged attack: up to 8 HP		
dropped item: Varies		

Aurexis

Behaviors

- **Fly**
- **Retrieve items and return them to Tempest ***
- **Retrieve items and drop them onto enemies from above ***
- **Swooping attack with talons/beak**
- **Charge attack to dive forward with burst of speed**
- **Activate switches and levers**
- **Screech to signal important items, areas, or enemies**
- **Snatch arrows and other projectiles out of the air**
- **Grasp Tempest's shoulders and help her glide on longer jumps**
- **Rest on Tempest's shoulder to regain life (2 HP per second)**
- **Move camera position**
- **Switch camera between 1st/3rd person**

* Aurexis' movement rate drops to 2.75 m/s when carrying an object in his talons.

Default Controls

Action	Keyboard & Mouse	Joystick
Bank left/right in air*	**A, D**	**Left analog stick**
Pitch up/down in air*	**W, S**	**Left analog stick**
Control Camera	**Mouse movement**	**Right analog stick**
Camera Default Position	**Page Up**	**R3 (depress right stick)**
First-Person View (toggle)	**Control**	**Left shoulder 1**
Dive	**Left mouse button**	**Button 3 (PSX X)**
Charged attack	**Space Bar**	**Button 1 (PSX Square)**
Pick up / drop item	**Right mouse button**	**Button 4 (PSX Circle)**
Talon scratch	**Middle mouse button**	**Button 2 (PSX Triangle)**
Screech (alert Tempest)	**Q**	**Left shoulder 1**
Grab Tempest (rest or aid in jumping)	**E**	**Right shoulder 1**
In-Game Menu	**Escape**	**Select**
Pause	**Enter**	**Start**
*** Aurexis can not hover in mid-air; he is always moving forward.**		

Aurexis

Base Orc Concept Art

Orcish Scout

Overview & Description

Polygon Count: **2,000 to 3,500**

The Orcish Scout is the most dexterous of the Orcish hordes, but that's not saying much. They wear the normal Orkdursk uniform, which consists of a ratty beige vest and black hide pants. They usually go barefoot and do not ride mounts. The Scout is never without his blackwood longbow, which is his only style of combat.

AI States

Aggressive	**Attack nearest player target**
Assist	**Aid another Scout that is calling for help**
Search	**Seek out players from ground or towers**
Retreat	**Flee from the players toward reinforcements**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 45**
arrow shot: 3 HP	**2.5 m/s walk** **1.25 m/s climb (optional)** **4 m/s run**	**Hit recovery: 0 sec**

Behaviors

- **Fire arrow (standard arrow range)**
- **Swing bow at target (melee attack – standard melee range)**
- **Yell for help (enemies within 32 meters can hear)**
- **Walk**
- **Run/Retreat**
- **Scout from watch towers**

Default Controls

Orcish Scout is a non-playable common enemy.

Orcish Woodcutter

Overview & Description

Polygon Count: **2,000 to 3,500**

The Orcish Woodcutter is a larger, stronger enemy than the Scout. He wears the standard uniform pants but no vest. He wields a large, single-bladed axe with which he can hew down trees and players alike. The Woodcutter is barefoot and lumbers about (no pun intended) on foot at all times.

AI States

Aggressive	**Attack nearest player target**
Idle	**Chop at nearest tree with axe**
Retreat	**Flee from the players toward reinforcements**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 75**
Axe strike: 9 HP	**1.75 m/s walk** **3.5 m/s run**	**Hit recovery: 0 sec**

Behaviors

- **Swing axe at player (melee attack – standard melee range)**
- **Parry melee attacks with axe haft**
- **Chop at closest tree with axe (idle animation)**
- **Walk**
- **Run/Retreat**

Default Controls

Orcish Woodcutter is a non-playable common enemy.

Orcish Lieutenant

Overview & Description

Polygon Count: **2,000 to 3,500**

The Orcish Lieutenant is physically more svelte than the Woodcutter but stronger than the Scout. He wears the Orkdursk uniform pants, but can afford to protect himself with a jerkin of studded brown hide. The Lieutenant wields a short, double-bladed hand axe in his right hand and a wooden buckler in his left.

AI States

Aggressive	**Attack nearest player target**
Command	**Order nearest Woodcutter or Scout to help**
Idle	**Sharpening his axe**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 125**
Axe strike: 7 HP	**2.25 m/s walk** **3.75 m/s run**	**Hit recovery: 0 sec**

Behaviors

- **Swing axe at player (melee attack – standard melee range)**
- **Parry melee attacks with buckler**
- **Sharpen axe (idle animation)**
- **Command nearest Woodcutter or Scout to assist him (32 meter range)**
- **Walk**
- **Run (Orcish Lieutenants never retreat)**

Default Controls

Orcish Lieutenant is a non-playable miniboss-level enemy.

Buzzard

Overview & Description

Polygon Count: **1,500 – 2,500**

The Buzzard is a large black bird that feasts on carrion. It will aggressively attack any creature that it senses might be close to death. It will also strike violently at any aerial threat. The Buzzard is devoid of feathers on the top of his head.

AI States

Aggressive	**Attack nearest player target**
Circling	**Stalk an enemy before attacking**
Defensive	**Attack only enemies that are attacking it**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 40**
Wing strike: 2 HP **Talon scratch: 3 HP**	**4 m/s fly**	**Hit recovery: 0 sec**

Behaviors

- **Fly**
- **Swooping attack with talons/beak**
- **Mid-air wing attack**
- **Circle around players**

Default Controls

Buzzard is a non-playable common enemy.

Kara'tok

Overview

Kara'tok is a field general in the Hordes of Orkdursk. He is a brutal, savage warrior who shows no mercy to friend or foe alike. Kara'tok fights with a massive double-bladed battle axe and a long, black steel claymore, able to wield each one-handed due to his colossal strength. He offers no respect for any living creature except his trained raven, Sul'talan.

Physical Description

Polygon Count: **2,000 to 3,500**

Kara'tok is a towering orc, standing 6'2" tall, and extremely muscular. He usually dresses in filth-crusted black hide pants and a blood-red double bandoleer over his otherwise bare olive skin.

AI States

Aggressive	**Attack nearest player target**
Command	**Order nearest Orc to help**
Defensive	**Focus on blocking all attacks**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 500**
Axe strike: 9 HP	**2.5 m/s walk**	**Hit recovery: 0 sec**
Claymore strike: 12 HP	**4.5 m/s run**	

Behaviors

- **Swing axe at player (melee attack – standard melee range)**
- **Parry melee attacks with claymore**
- **Swing claymore at player (1.5 times melee attack range)**
- **Command nearest living Orc to assist him (32 meter range)**
- **Walk**
- **Jump**
- **Command Sul'talan**
- **Run (Kara'tok never retreats)**

Kara'tok

Default Controls

Kara'tok is normally a non-playable boss enemy. However, by entering a code or unlocking the secret ability, Kara'tok can become playable. When this is done, his controls are as follows:

Action	Keyboard & Mouse	Joystick
Walk/Run	**W, A, S , D**	**Left analog stick**
Control Camera	**Mouse movement**	**Right analog stick**
Camera Default Position	**Page Up**	**R3 (depress right stick)**
Claymore Slash Activate Switch	**Left mouse button**	**Button 3 (PSX X)**
Axe Slash	**Middle mouse button**	**Button 1 (PSX Square)**
Jump	**E**	**Button 2 (PSX Triangle)**
Block	**Space bar**	**Button 4 (PSX Circle) Right shoulder 2**
Pause	**Enter**	**Start**
In-Game Menu	**Escape**	**Select**
Toggle Sul'Talan's AI state	**Shift**	**Left shoulder 2**
Taunt / Comment	**T**	**L3 (depress left stick)**
*** Commanding orcs is not available when playing as Kara'tok.**		

Sul'talan

Overview

Sul'talan is a tamed raven owned by Kara'tok. Like his owner, he has no compassion for the living. Sul'talan feeds on decomposing flesh, and considers killing merely an opportunity to procure his next meal.

Physical Description

Polygon Count: **1,500 to 2,000**

Sul'talan is a large, jet black raven. His eyes are blood-red. He wears a leather harness around his back and wings, with which Kara'tok can train or restrain him. It is not unusual to find bits of ichor or flesh nettled into his oily black feathers.

AI States

Aggressive	**Attack nearest enemy whenever possible**
Assist	**Deflect attacks on Kara'tok, aid in jumping**
Search	**Seek out items to retrieve and switches**
Rest	**Perch on Kara'tok's shoulder and regain HP**

Statistics

Damage:	**Movement rates:**	**Maximum HP: 300**
talon slash: 4 HP	**4 m/s flight**	**Hit recovery: 0 sec**
wing slap: 3 HP	**8 m/s diving**	
charge attack: 10 HP		
dropped item: Varies		

Behaviors

Sul'talan has the same behavior set as Aurexis. See Aurexis' character detail sheet for more information.

Default Controls

By default, Sul'talan is a non-playable boss enemy. However, if the player enters a cheat code from the menu, Kara'tok becomes a playable character. When playing as Kara'tok, Sul'talan becomes an AI-controlled character just as Aurexis is in the normal one-player mode. In two-player cooperative mode when player one is controlling Kara'tok, player two controls Sul'talan. In this case, Sul'talan has the same control set as Aurexis. See Aurexis' character detail sheet for more information.

Models & Animations

Tempest Silvreleaf

Polygon Count : **2,500 – 4,000**

Tempest stands 5'5" tall. She has dusty blonde hair, which she wears in a braid that ends between her shoulderblades, and piercing blue eyes. She typically adventures in a dark green tunic and brown hide pants with her sabre at her side and a quiver over her shoulder, though she does change clothes often. Tempest is 22 years old. She is well-toned and well-proportioned throughout, especially her muscular legs, built for dodging and evading. **Two versions of the Tempest model are required, one which is wearing an ankle-length skirt, and one which is wearing pants.**

Animations:

- Raising her wrist as a perch for Aurexis
- Fire an arrow, including reaching back to quiver to draw it out
- Slash her sabre diagonally
- Thrust her sabre forward, preferably with both hands
- General somersault
- Walking
- Running
- Raise her gauntleted arm to block (can be the same as perching)
- Parry with her sabre
- Jump
- Climb a ladder
- Crouch down
- Pick up an item while crouching
- Flinch on being hit
- Dropping to knees, slumping over (death)
- Kick off a wall to double-jump

Textures:

- Standard ranger gear; brown leather quiver, gauntlet and boots, green tunic and tanned brown pants. Blonde hair, fair skin, blue eyes.
- Amazon outfit pictured in Tempest's character description, but keeping the facial features and skin tone of the base Tempest.
- 2-4 sets of alternate outfits to be discussed with artists for purposes of feasibility. At least two should use the "skirted" model, though alpha-blended textures can be used to make the skirt appear shorter. (Preferably, one will use the full-length skirt while the other would use the shorter version via texture blending)

Models & Animations

Aurexis & Sul'talan

Polygon Count: **1,200 – 2,000**

Aurexis is a brown, sleek falcon. His beak is short and hooked like a buzzard's. His keen eyes are brown and never still. He has mostly short feathers, brown with splotches of a darker brown, and a few with black tips. Aurexis' talons are knobbly and brownish-gray. He wears no harness to indicate domestication.

Sul'talan is a large, jet black raven. His eyes are blood-red. He wears a leather harness around his back and wings, with which Kara'tok can train or restrain him. It is not unusual to find bits of ichor or flesh nettled into his oily black feathers.

Note: Aurexis and Sul'talan will share this model, with differences lying in scale (handled programatically) and texture.

Animations:

- Wing flap
- Sleek forward dive with all feathers back (charge attack)
- Tilt back and slash forward with talons
- Tilt back and strike forward with wings
- Perch on shoulder or wrist
- Grab and lift with both talons
- Glide in a circular pattern without flapping wings
- Flinch on being hit

Textures:

- **Aurexis:** see above description.
- **Sul'talan:** see above description.
- **Alternate Sul'talan:** Above description, but without the harness.

Buzzard

Polygon Count: **1,500 – 2,500**

The Buzzard is a large black bird that feasts on carrion. It will aggressively attack any creature that it senses might be close to death. It will also strike violently at any aerial threat. The Buzzard is devoid of feathers on the top of his head.

Animations:

Same as Aurexis & Sul'talan, less charge attack and perch.

Textures:

See description above.

Models & Animations

Orc

Polygon Count: **2,000 – 3,500**

The base Orc is large and muscular everywhere, in a grotesque manner. It is bipedal but hunches over, looking almost like an enormous rotting gorilla in stature. Differences in various orc types will be handled via texture and scale.

Note: Animations can be shared with Tempest wherever appropriate.
Note: All four orc types will share this model, with differences lying in scale (handled programatically) and texture.

Animations:

- Raising wrist as a perch for a bird
- Fire an arrow, including reaching back to quiver to draw it out
- Slash weapon diagonally (with each hand – for dual-wielding boss)
- Double-handed horizontal chop with large axe
- Walking
- Running
- Raise left arm to block (can be the same as perching)
- Parry with right-hand weapon
- Jump
- Climb a ladder
- Crouch down
- Pick up an item while crouching
- Flinch on being hit

Textures:

- **General:** Olive-gray skin. Shoulder-length, greasy black hair. Large, exaggerated grotesque facial features. Separate faces would be preferable for Orcish Scout, Woodcutter and Lieutenant, but this is not imperative if time does not permit.
- **Orcish Scout:** They wear the normal Orkdursk uniform, which consists of a ratty beige vest and black hide pants. Barefoot.
- **Orcish Woodcutter:** Same pants as Scout, no shirt (show muscular chest). Barefoot.
- **Orcish Lieutenant:** Dark brown, ichor-crusted studded hide armor, same black pants as scout. Hide boots.
- **Kara'tok:** Same pants as Scout, same boots as Lieutenant. No shirt, bandoleers across chest with black metal shoulder guards. Looks especially menacing.
- **Kara'tok Alternate:** Same outfit as Kara'tok, but with dark gray pants and barefoot. Should be visually distinctive from Kara'tok at combat pace. This is used for the boss orc when playing as Kara'tok.

Models & Animations

Weapons

Polygon Count: **500 – 1,500 each**

- **Longbow** (textures: carved wood for Tempest, black wood for Orcish Scout)
- **Sabre** (texture: polished silver blade, ornate green and brown inlay hilt)
- **Claymore** (texture: black metal, stained with dark blood)
- **Woodcutter's axe** (texture: rotten driftwood haft, rusty metal blade)
- **Short double-bladed hand axe** (texture: blackened blade, cherry wood haft)
- **Arrow** (textures: black rot matching Orcish bow, white oak for Tempest's)
- **Small round shield** (texture: rotten driftwood with iron stud through center)

General Loose Items

Polygon Count: **500 – 1,500 each**

- **Wooden barrel** (any unfinished wood is OK)
- **Wooden crate** (any unfinished wood is OK)
- **Branch** that can be chopped off a tree (small)
- **Large log** consistent with trees in the level
- **Rock** (sized to be appropriate to be carried by Aurexis)
- **Torch** (sized to be appropriate to be carried by Aurexis)
- **Herb plant** (for health regeneration – should be distinguishable from standard foliage)
- **Small bushes and shrubs**

Model File Format: .mdl

This is a custom file format related to the model exporter and loader we will be using. This exporter is a plug-in for Maya and will be made available to artists.

Sound Effects

Tempest Silvreleaf

- *Voice:* Female pain yelp
- *Voice: "Aurexis!"*
- *Voice: "Thank you, Aurexis."*
- *Voice: "They'll pay for this..."*
- *Voice: "This place resonates with anger..."*
- *Voice: "Eugh! Disgusting!"*
- *Voice: "Arrow, aim true."*
- *Voice: "Hold still..."*
- *Voice:* Battle yell while swinging sword - *"Yaa!"*
- Falcon-calling whistle
- Sword being drawn from sheath
- Arrow drawn from quiver
- Bow creaking as it's drawn back
- Arrow shot
- General exertion noise (for jump, dodge, etc.)
- Sword-on-sword clang
- Sword whoosh

Aurexis, Sul'Talan & Buzzard

- Falcon scream while diving: *"Skreeeeeeee!"*
- Evil-sounding raven cackle: *"Ca-CAW!"*
- Buzzard attack noise
- Wing flap, gentle
- Wing flap, frantic and rapid (takeoff)
- Pained bird noise for hit

Orcs

- *Voice: Guttural* growl
- *Voice: Guttural* laughter
- *Voice:* Scout: *"She's here!"*
- *Voice:* Lieutenant: *"Over here, maggot!"*
- *Voice:* Woodcutter: *"I'll hack you to pieces!"*
- *Voice:* Kara'tok: *"Sul'taran, tear them apart."*
- *Voice:* Kara'tok: *"This forest is mine now."*
- *Voice:* Kara'tok: *"I'll break your neck like a chicken's."*
- Sword-on-wood contact
- Chopping wood noise
- Axe whoosh (deeper than sword woos to reflect larger size)

Sound Effects

Ambient

- Bird chirping in distance
- Leaves crunching underfoot
- Fire, torch whoosh
- Fire, standing flame crackle
- Breaking wooden container (chest/barrel)
- Scurrying small woodland creature feet
- Running brook
- Creaking wooden door / drawbridge
- Small explosion
- Howling wind
- Gentle breeze through trees
- Loud war horn
- Additional sounds at the artist's discretion

Music

Level 1 – The Depths of Neverwood

The music should be based in a Celtic theme. It should convey the emotion of a peaceful place in crisis. Deeper drums are appropriate here, and harps and bells are also preferred instruments if the artist feels they are appropriate. This music should loop and be approximately 2-4 minutes in length.

Level 2 – The Field of the Fallen

The music should be based in a Celtic theme as well, but sadder. It should have an air of stealth/mystery to it, almost as a less industrial version of something out of *Metal Gear*. This music should loop and be approximately 2-4 minutes in length.

Level 3 – The Heart of the Horde

This music should be primarily played with deep, heavy drums and bells. Chanting would be appropriate as well. We seek to create a *Duel of the Fates*-style feeling of epic combat. This music should loop and be approximately 1-3 minutes in length.

Front-end Menu Loop

If the Level 1 music conveys a peaceful place in crisis, the front-end music should be essentially the same peaceful place, without the crisis. The music here should be lighter and happier, dealing with brighter instruments and a lighter beat. This should also loop, and should be approximately 1-3 minutes in length.

Sound File Formats

Sound Effects:	**.WAV**
Music:	**.MP3**

Force Feedback Effects

Force Feedback Effects

- Bow shot draw and release
- Small explosion
- Break an object
- Block impact
- Sword whoosh
- Damage impact
- Swaying effect (suspension bridge)
- Pulling effect (opening a door)
- Footsteps
- Wing flaps
- Sustained tension (charging bow shot)

Force Feedback Effect File Format

DirectInput Force Feedback Effect: **.FFE**

Melee Mechanics

Tempest's sword is used for two melee attacks: slashing and thrusting. Each will require its own animation. Successful thrusts and slashes both cause a knockback of the target equal to 0.2 meters per hit point of damage assessed.

Thrusting is a longer-range strike that seeks to impale the opponent. This attack is used as a closing strike. It is a more deliberate attack, dealing significantly more damage than the slash attack. However, the attack is also slower; and since it leaves Tempest more extended, it leaves her more open to attack (for a period of .5 seconds) when she misses with the thrust. Thrusting has a narrow strike radius of 10 degrees, centered on the “forward” vector of the player. Its range is 1.5 meters.

Slashing encompasses a shorter range from the character (1 meter), however, it provides a broader range across the character's body. The slash attack will successfully hit anything within the 1.5 meter range and within a strike radius of 45 degrees centered on the “forward” vector of the player. This strike does less damage, but is much faster and can be executed in rapid succession. After three slashes, the input will force the player to wait a moment, to prevent the player from simply mashing the slash button as he/she runs through the level.

Pressing the Block button with no direction selected will cause the character to parry an attack with her gauntlet. This is a question of precise timing as to whether or not the block succeeds. Failed blocks will result in the player taking full damage, whereas a successful block will prevent all damage to the player from that attack.

Dodges are also timing-based and are executed by pressing the Block button and a direction simultaneously. Dodging will display a somersaulting animation as Tempest whirls out of harm's way. If the dodge is not properly timed, the character will be knocked back twice as far as she normally would as she is off-balance at the time the strike connects.

Bow Mechanics

The bow is controlled with the Fire Bow button. Holding the button down allows for the bow to “charge” up to a certain point, causing much greater damage (up to 300% of normal bow damage) to the enemy on a successful hit. This extra damage will be granted at the rate of 100% for each second the button is held, to a maximum of two seconds for 300% total damage. Fractions of sections will be converted into percentages and added; thus, holding the button for 1.5 seconds would confer a 150% damage bonus and 250% damage overall.

In third-person firing mode, the player can still walk as normal and fire the bow. Holding the button for increased damage remains viable. The arrow is released when the Fire Bow button is. Since there is no first-person aiming available in this mode, the bow will auto-aim to whichever enemy within Tempest's view frustum is the shortest distance from her firing hand. Simple relative line-to-sphere checks will resolve this in an inexpensive fashion.

First-person mode is activated by holding down the First Person Button. This will cost Tempest the ability to walk while in this mode, but she will be able to aim her bow precisely to achieve sneak attacks on un-alerted enemies and enemies at a greater distance. This also provides her with a better opportunity to achieve more critical damage by drawing the bow back to its maximum force.

Arrows will have a yellow and white particle trail when fired, to intimate light or a “pixie dust” effect coming from the back of the arrow. This is not a 'magical' effect so much as it is an aid to the player in tracking the flight path of arrows visually. Tempest's bow has unlimited arrows. The “fast” shot will be executable three times per second. It will take approximately .1 second to draw the new weapon when switching from sword to bow and back.

Arrows that collide with their target will assess damage and will remain stuck in the model until it is destroyed. The enemies will contain a list of arrows, and when an enemy is struck, the projectile will move from the “active projectile” list to the enemy's own list so that it is transformed as a part of the enemy's geometry. Special effects to arrows (fire, etc.) will also resolve at this time. They will also alter the particle effect associated with the arrow's trail, either replacing or adding to the existing “light” effect mentioned.

Standard bow shot range: 40 meters
Charged bow shot range: 48 meters

Aerial Combat Mechanics

The fight mechanics for aerial combat are an interesting challenge, perhaps the greatest of many in this development cycle. The combat must be fluid and fast, and the controls kept responsive. All research done on this topic has indicated that aerial motion and combat is largely done by trial and error. However, guidelines for beginning are as follows.

Birds can not hover, however, they can slow down and speed up. This reduces aerial motion to a problem of tilt, speed, and turning radius. Aurexis is by far the fastest of the three birds in straight flight, at 6 m/s. Sul'talan can only achieve 4 m/s, while the Buzzard is restricted to 3 m/s. Sul'talan, however, is more maneuverable than is Aurexis, as his turning rate is 120 degrees per second to Aurexis' 90. The lowly, disposable Buzzard can achieve a modest 60, preferring to glide in wide circles around its prey.

All attack damage will cause “knockback”, except in the case of aerial combat, the knockback will be applied in both the direction of the attack, and in the negative Y. An aerial combatant who is knocked to the ground suffers an additional 3 points of damage, and is significantly defenseless until it can get back in the air. Arrow shots from the ground will translate the avian backward the full amount of the knockback, and half the knockback distance in the positive Y. However, the bird will then fall 60% of the knockback value in accordance with the force of gravity.

Diving attacks double the motion speed of the birds, with the exception of Aurexis who only gains 2 meters per second due to his already incredible flight rate. A successful charged dive attack deals 8 points of damage, but can only be executed when the charge meter is full. It takes 5 seconds to charge the meter from empty to full, and a charged attack drains the meter completely. Charge may not be gathered while the bird is at a rest state with its owner.

Resting on the shoulder of the bird's owner earns back 2 hit points per second. No healing is awarded in a second during which the owner has taken damage from an attack.

If a player bird dies, it respawns on the shoulder of its owner 8 seconds later. When Sul'talan dies as the NPC boss avian, he does not respawn. If the player owner dies, the bird may continue play, but as it cannot gather Inju plants and has no shoulder on which to rest, it is not likely to succeed in completing the level.

Combat Quick Reference

Attack	Used by	Range (m)	Time (s)	Damage (HP)
Sword Thrust	Tempest	1.5	0.75	5
Sword Slash	Tempest	1	0.25	3
Bow Shot	Tempest Orcish Scout	40	0.33	3
Charged Shot	Tempest	48	Up to 2	Up to 9
Talon Slash	Aurexis Sul'talan Buzzard	0.2	0.33	3
Wing slap	Aurexis Sul'talan Buzzard	0.4	0.25	2
Charged Dive	Aurexis Sul'talan	20	Up to 2	Up to 9
Axe Chop	Orcish Woodcutter Kara'tok	2	1.25	9
Axe Slash	Orcish Lieutenant	1.5	0.85	7
Claymore Bash	Kara'tok	2.5	1.1	12

Hit Point Chart

Tempest	100
Aurexis	75
Orcish Scout	40
Orcish Woodcutter	75
Orcish Lieutenant	125
Buzzard	40
Kara'tok	500
Sul'talan	300

Heads-Up Display

Heads-Up Display Concept

Display Interface Elements

- The red bar indicate the player's health as a percentage. The bar will empty to black as the player loses life.

- The yellow bar represents the “charge” ability of arrows. This bar begins empty, and will fill when the Fire Bow button is held down. When the bar fills to its maximum level, it will glow slightly with a particle effect to indicate this. The bar will empty when the arrow is released, and can be recharged immediately.

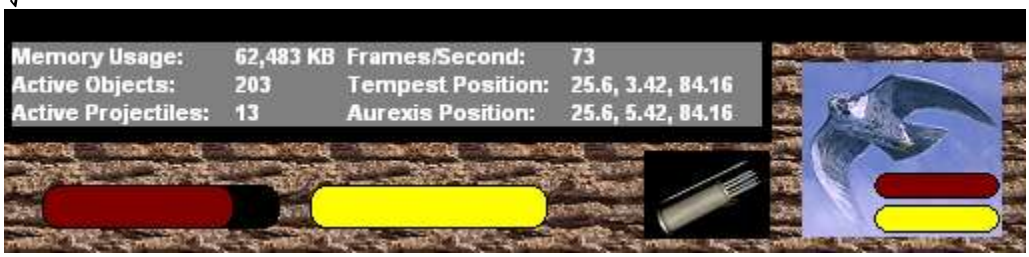
- The arrow picture will show either a bow and arrow or a sword to indicate which weapon is currently being used. If special arrows (such as exploding arrows) are equipped, the picture will indicate this via a particle effect. This element will display a static picture of Kara'tok's claymore and axe crossed when playing as Kara'tok.

- The bird picture will be replaced with a 2D rendering of Aurexis (or Sul'talan, if playing as Kara'tok).The red and yellow bars inset into the bird picture represent Aurexis' (or Sul'talan's) life and “charge” for dive attacks, respectively.

- The background of the display should appear very woodsy, as if it is all inlaid into a log. The display overall should be extremely slim and unobtrusive, and will hug the bottom-right of the window (or pane, in split-screen mode). The display will be as small horizontally as possible while still being large enough to convey the information clearly to the player(s). Optimally, it will take up no more than 50% of the screen width.

Debug Heads-Up Display

Debug Heads-Up Display Concept



Display Interface Elements

The Debug HUD maintains all the functionality of the regular HUD, but adds realtime statistics on the game's status for debugging purposes. This will be output as simply white text on a quad. It will contain the following information:

- The X, Y, Z position of both Tempest and Aurexis
- The number of base objects and projectiles currently active in the system
- Total dynamic memory usage
- Frames per second counter
- Any other information programmers feel is necessary to track in-game.

Power-Ups

Special arrows can be picked up throughout the level(s). These arrows appear in pick-up form as a single spinning arrow with a particle effect on it, hovering about a meter off the ground. Upon collecting the arrow, Tempest's regular shot will be replaced with the new arrow type, and the display on the HUD will be updated as well.

Exploding Arrow

This arrow has a reddish-orange particle effect behind it as it flies. Upon collision with an enemy or static geometry, the arrow explodes, creating a sphere of flame around the collision point. Any enemy within the radius of this sphere suffers 4 HP worth of damage. If the arrow collides with an enemy directly, that enemy suffers 2 HP of damage, plus the standard bow damage, plus the explosion damage.

Freeze Arrow

This arrow has a blue and purple particle effect behind it as it flies. Upon collision with an enemy, the enemy has a 50% chance to become partially frozen, reducing his movement rate to 1/3 of its normal rate. This includes speeds for movement, attacking, and defending. To better display this effect visually, a successful “freeze” will change the model's base color from white to a blush-gray. This will preserve all texturing information, while giving the model a “chilled” or “frozen” appearance.

Inju Plant

This small, aloe-like herb promotes rapid healing and stops blood loss. Simply break the leaves and the plant oozes a salve that will cure whatever ails Tempest, or at least 20 HP worth of it.

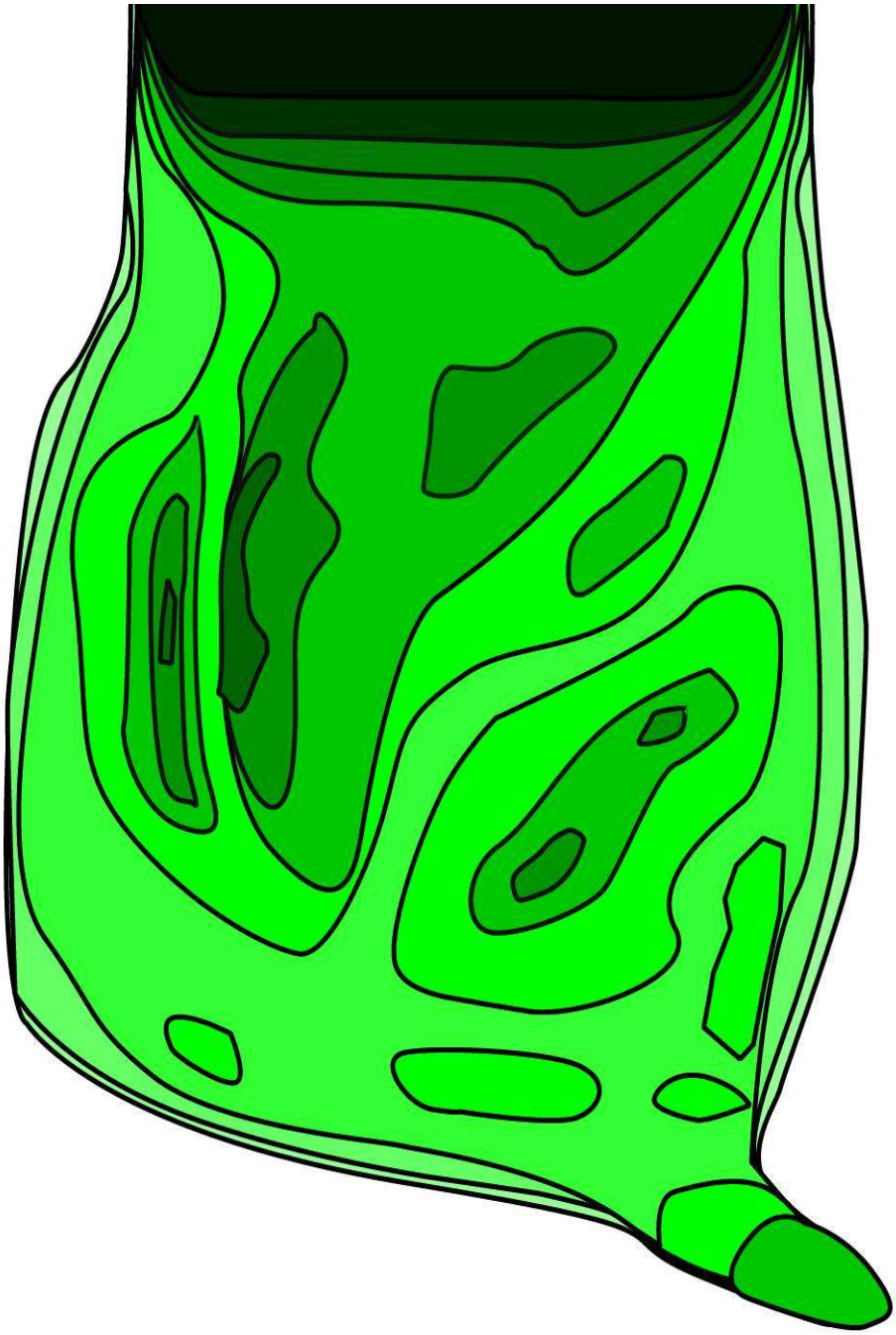
Power Statue

This falcon statue (simply use a static, scaled-down version of the Aurexis model) will increase the strength of any bird of prey, allowing Aurexis (or Sul'talan) move at his normal movement rate while carrying items.

Relief of Cocidius

Modeled in the image of the Celtic god of war and the forests, it allows Aurexis to sustain his Charged Dash for a full seven seconds.

Level 1 — The Depths of Neverwood



Level 1 – The Depths of Neverwood

Level Setting

Tempest and Aurexis were wandering in the wood one day when they heard a great deal of stomping and commotion in the distance. They investigated, and they found that the forest was full of orcs! Tempest decides to dispatch the monsters that are destroying her glen, and seek out their leader.

The forest is dense in this area, and a thick canopy of tree branches above blocks out the sun in most areas, though small beams of light manage to wiggle their way through the leaves in places. The tree trunks are as thick as two men, and the ground is thick with underbrush revitalized by the string. Overall, the forest has the feeling of a tranquil place, an untouched sanctuary, and the player should get the feeling that this is the type of place that is worth protecting.

Level Goal

Tempest and Aurexis must navigate through the woods, dispatching any orcs they find, until they reach the exit, which is defined as having passed a certain Z coordinate as marked by the map. The actual numeric value will be determined based on the scale of the model/terrain.

Expected Completion Time 5 - 10 Minutes

Enemies

65% - Orcish Scout
25% - Buzzard
10% - Orcish Woodcutter

Powerups Available (dropped by enemies)

Chance of enemy dropping a powerup on death: 10%

If powerup is dropped, probabilities are as follows:
65% - Inju Plant
10% - Freeze Arrow
10% - Power Statue
10% - Relief of Cocidius
05% - Exploding Arrow

Level 1 — The Depths of Neverwood

Interactions

As Tempest

- Engage in melee combat with Orcish Scouts and Woodcutters
- Engage in ranged bow combat with all enemy types
- Acquire powerups
- Jump, roll, and wall-jump (optional) around a picturesque forest
- Disturb shrubs and underbrush as she passes against them
- Kick up dust with her feet as she runs (particle effect)

As Aurexis

- Engage in aerial combat with Buzzards
- Engage in aerial vs. ground combat with Orcish Scouts and Woodcutters
- Acquire powerups
- Soar through the canopy of the majestic oaks
- Pick up loose rocks and objects and drop them onto enemies
- Rest on Tempest's shoulder to regain health
- Snatch arrows out of mid-air

Flow of the Level

1. Swing in on a rope or some such to get to the Start Point

2. Begin working upward through the level. At first, orcs will be un-alerted to your presence and can be killed easily with arrows in First Person Mode.

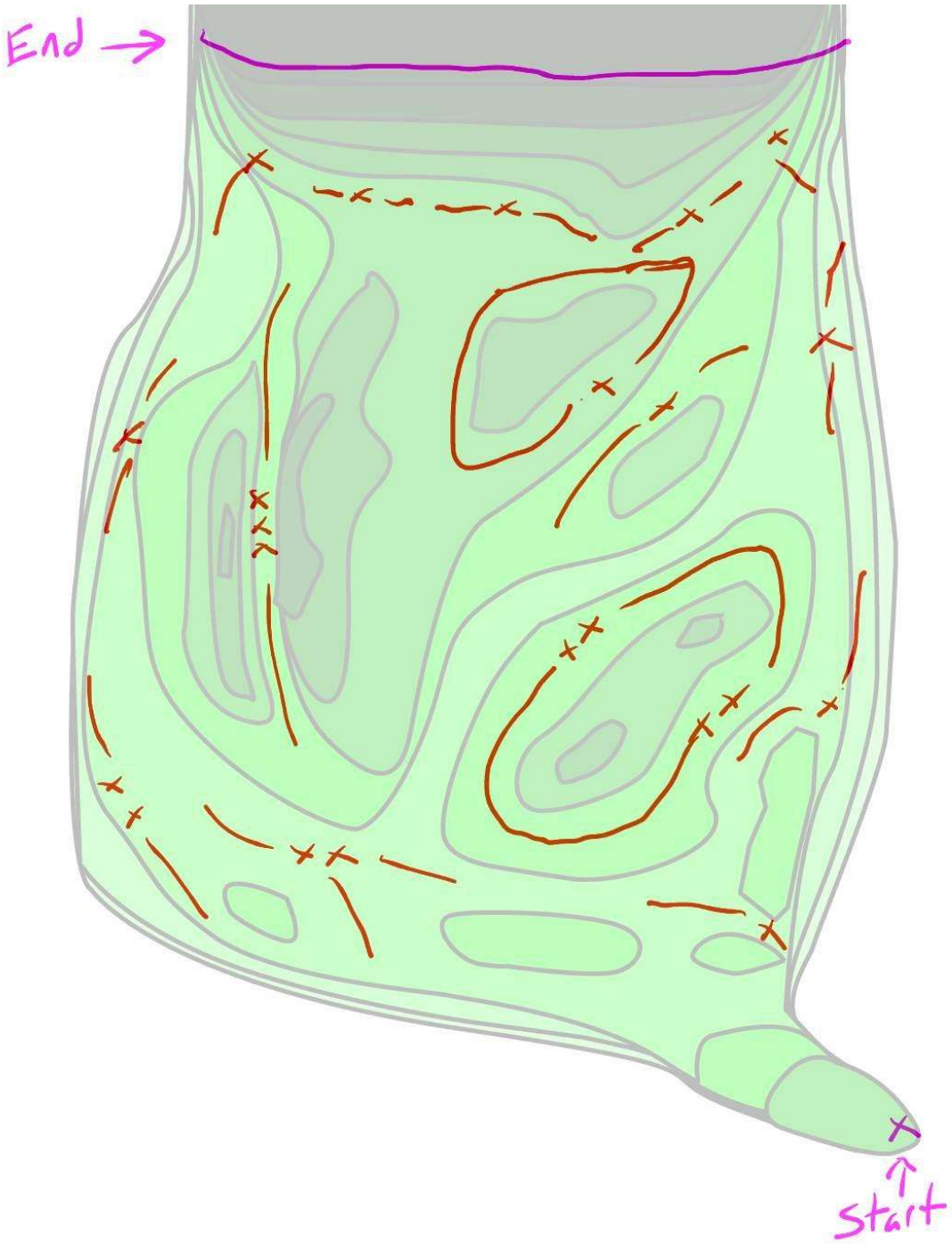
3. As the pair continues up the pathways, Buzzards begin attacking from above. Aurexis should take to the skies to eliminate them while Tempest pushes forward, dealing with the orcs who now have a 50% chance to be alerted to her approach.

4. As Tempest reaches the last 20% of the level, the enemies will have a 90% chance to be alerted to Tempest's approach, and the number appearing will increase. Tempest will need to be ready with her sword to handle them, now!

5. Tempest exits the western grove of Neverwood onto a plain of hewn-down logs and stumps. Orcs are building watch towers in this region.

Level 1 – The Depths of Neverwood

Patrol Paths for Enemy AI



Level 2 – The Field of the Fallen

Level 2 — The Field of the Fallen

Level Setting

Tempest and Aurexis have left the deepest part of the forest, and are devastated by what they encounter. The entire meadow, once dotted with the most ancient and proud spruce, is laid bare of its greenery. Instead, the plain is littered with great, wide stumps and logs strewn about carelessly like casualties on a battlefield.

Around the logs, watch towers have been erected, on which Orcish Scouts keep vigilance against any spying rangers. They are equipped with post-mounted horns that the Scouts use to signal any approaching danger. At the opposite end of the field from the Depths of Neverwood lies the beginnings of a small rocky outcrop with the River Brynnan slicing through it. A suspension bridge of wood and rope connects it to cross the river onto Kealah's Point, a larger, rocky hilltop near the center of the forest.

Level Goal

Tempest and Aurexis must sneak carefully through the maze of logs and bramble, being careful not to raise suspicion from the Orcish Scouts atop the watchtowers. If they sound their mighty horns, the whole of the Fourth Horde will be atop them, and they are as good as dead.

In order to achieve this goal, Aurexis will be called into service. He must fly up to the towers, scratching at the Scouts, dropping items onto them, and generally causing ruckus while Tempest slips through the tower's coverage area unseen. Even then, however, she must be careful, because the Woodcutters that are working on the logs will surely take offense to your presence.

Expected Completion Time — 10 - 15 Minutes

Enemies

40% - Orcish Scout
10% - Buzzard
45% - Orcish Woodcutter
05% - Orcish Lieutenant (Three, guarding the near side of the bridge)

Level 2 — The Field of the Fallen

Powerups Available (dropped by enemies)

Chance of enemy dropping a powerup on death: 5%
Chance of enemy dropping a powerup on successful distraction: 10%

If powerup is dropped, probabilities are as follows:
30% - Inju Plant
10% - Freeze Arrow
20% - Power Statue
20% - Relief of Cocidius
20% - Exploding Arrow

Interactions

As Tempest

- Engage in melee combat with Orcish Woodcutters
- Emphasis in ranged bow combat with all enemy types, to kill enemies quickly and silently
- Acquire powerups
- Hide behind objects and sneak deftly around towers to dodge the enemies' sight
- Kick up dust with her feet as she runs (particle effect)
- Activate a switch on the wall in order to open the gate to the rocky outcropping
- Climb over large logs, dropping down into melee range with an unsuspecting Orcish Woodcutter

As Aurexis

- Engage in aerial combat with Buzzards
- Engage in aerial vs. ground combat with Orcish Woodcutters
- Distract tower guards with attacks and item drops
- Acquire powerups
- Pick up loose rocks and objects and drop them onto enemies
- Rest on Tempest's shoulder to regain health
- Snatch arrows out of mid-air

Level 2 – The Field of the Fallen

Flow of the Level

1. Walk into the level, show a camera pan of the devastation.

2. After Tempest takes ten steps, a horn shatters the calm. Ten orcs dart to the sound, axes at the ready. It was a false alarm and they all disperse, but Tempest is now aware of what she is up against.

3. Tempest makes her way through the maze of logs. As she encounters Woodcutters, she dispatches them with her First Person Mode shooting, taking full advantage of her charge ability to deal the most damage possible and drop the Woodcutters before they give her presence away. The arrow from the hollow is silent and swift, but its justice is sure.

4. Tempest waits behind a large stump while Aurexis soars to the top of the nearest tower, flapping his wings and scratching at the Scout manning it. The Scout flails about the tower, trying to shield his eyes from the crazed bird that has just entered his nine-foot square dominion. So concerned is he with his falcon foe that he fails to notice the deft lady of the woods hurrying under the base of the tower and behind a log on the opposite side.

5. Sprinting past the last watch tower, Tempest prays that Aurexis can keep the guard busy, as there is no cover between her position and the wooden stockade the orcs have erected. She reaches the gate, and her haste is redoubled when she discovers that she cannot open it. Aurexis, growing quite tired from his frenzied flapping and fighting, engages the tower Scout one last time while Tempest finds the large, sturdy wheel on the outer wall. Turning it with all her strength, which admittedly is not comparable to even the weakest orc, she manages to open the gate just enough to slip her slender body through. Aurexis, thrilled for a chance at a rest, soars over the wall to join his companion.

6. The gate was apparently partitioning off a lumberyard. The Woodcutters are hacking logs to bits, shaving them into boards for use in constructing Orkdursk weapons of war. With no guard towers in this area, Tempest is free to engage the Woodcutters at close range, if she is willing to brave the long reach of the Woodcutters' axes. On second thought, she reaches back to her quiver.

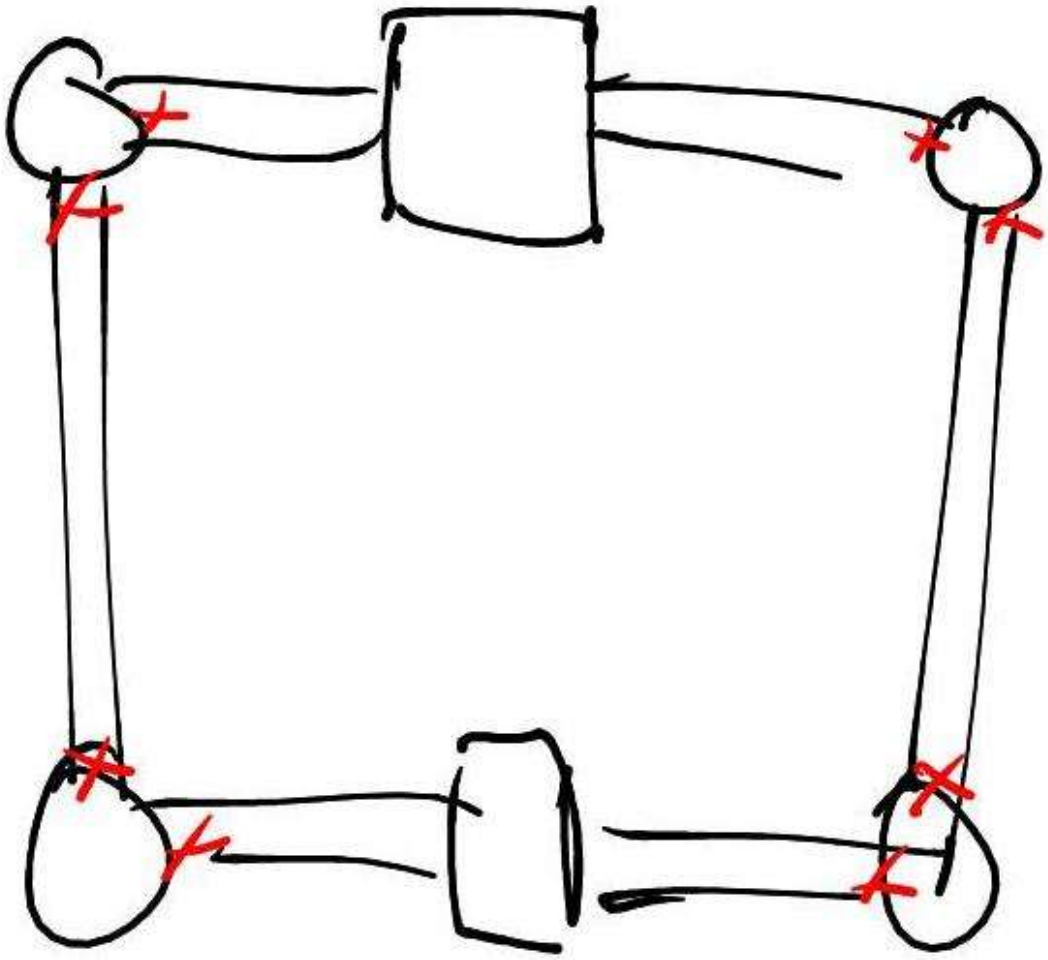
Level 2 — The Field of the Fallen

Flow of the Level (continued)

7. As Tempest nears the bridge, she realizes why the Woodcutters have been working so hard. Three Orcish Lieutenants are standing about, keeping watch on the Woodcutters' activities. Each eager to be the one to bring the ranger girl's head to Kara'tok, the Lieutenants practically fall over themselves to be the first to draw Tempest's blood. The woodsmistress draws her silver saber and sets about decapitating the leadership, literally. The Lieutenants summon a few Woodcutters to their aid, but Aurexis' swooping Charged Attacks make the life-or-death difference.

8. Tempest crosses the suspension bridge, Aurexis perched on her wrist. Kaelah's Point lies ahead, and the massive stockade surrounding it can only mean one thing; there are more orcs and they are up to no good.

Level 3 – The Heart of the Horde



Level 3 – The Heart of the Horde

Level Setting

Tempest and Aurexis clamor up the rocky hilltop to reach Kaelah's Point. No longer is it the peaceful place of sweeping vistas and the sound of the river, however. The Horde has constructed a massive stockade fort on the hilltop, its imposing gates heavily guarded by Orcish Lieutenants and Scouts. Not only is it being used as fortification, but the orcs have also begun digging into the hillside, delving for the iron deposits within.

Once inside, Tempest and Aurexis are discovered, and a thundering horn blast shatters the silence. Barreling forth from the tents and mine shafts, a seemingly endless fray of Woodcutters (who are less than pleased at being asked to play at mining) and Lieutenants charge at Tempest, while the few Scouts left at the fortress take aim from the walls. Vastly outnumbered, there is no time for stealth now. Tempest draws her blade and charges at the thundering horde, while a shadow watches from the command hut.

Level Goal

With all the remaining Fourth Horde upon them, Tempest's primary goal is to keep herself and her avian companion alive. As she fiercely battles the advancing throng of brutes, Aurexis' keen eyes spy out some loose parts in the rock piles above the shafts. If only he could shift them just enough, the rocks would fall, sealing off the mine shafts and the reinforcements within.

When all four mine shafts have been sealed off and the remaining orcs within the stockade dispatched, Kara'tok will emerge from the command hut. With a guttural curse, he raises his arm to the heavens, and a sleek, jet-black raven perches upon his fist. Aurexis, his blood boiling at the thought of a fellow avian aiding in the decimation of Neverwood, takes wing and charges Sul'talan. Meanwhile, saber held firm and legs weakened with exertion, Tempest must battle Kara'tok to the death.

Expected Completion Time 15 - 25 Minutes

Enemies

10% - Orcish Scout
38% - Orcish Lieutenant
50% - Orcish Woodcutter
01% - Kara'Tok (Will attack when all the generators are disabled)
01% - Sul'talan (Will attack when all the generators are disabled)

Level 3 – The Heart of the Horde

Powerups Available (dropped by enemies)

Chance of enemy dropping a powerup on death: 10%

If powerup is dropped, probabilities are as follows:
50% - Inju Plant
10% - Freeze Arrow
10% - Power Statue
15% - Relief of Cocidius
15% - Exploding Arrow

Interactions

As Tempest

- Engage in melee combat with the orcish hordes
- Emphasis in ranged bow combat with all enemy types, if she can get a shot off before she is overrun
- Acquire powerups
- Direct Aurexis to seal off the mine shafts
- Kick up dust with her feet as she runs (particle effect)
- Use powered-up arrows to destroy multiple death-dealing orcs in a single hellish blast

As Aurexis

- Engage in an aerial vs. ground free-for-all war with Orcish fighters
- Experience a mid-air duel to the death with the flesh-rending raven, Sul'talan
- Acquire powerups
- Pick up loose rocks and objects and drop them onto enemies
- Rest on Tempest's shoulder to regain health
- Snatch arrows out of mid-air
- Loosen rocks to seal off the mine shafts and the endless flow of orcs

Level 3 – The Heart of the Horde

Flow of the Level

1. Enter the stockade. The doors slam shut behind Tempest and arrows begin to fly from the walls. A loud horn blast shakes the fortress. Orcs of every type begin pouring out of the mine shafts.

2. Tempest frantically hacks and slashes at the orcs, retreating to a corner to use her bow whenever the opportunity presents itself. Meanwhile, Aurexis circles the inner walls of the fortress, picking at the mine shaft supports where he can and defending Tempest from arrows whenever the opportunity arises.

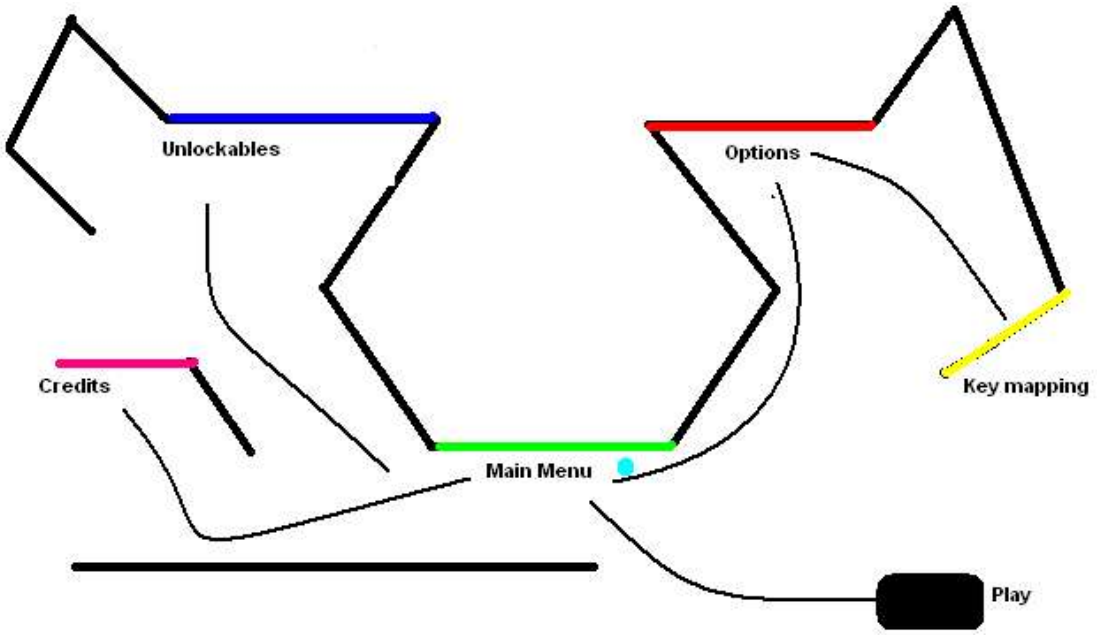
3. The onslaught of orcs does not cease until all four mine shafts have been sealed. The orcs spawn at a rate of one every four seconds, per shaft. In Easy mode, only one shaft at a time will spawn, with one more added for each difficulty level after Easy.

4. When all the shafts are sealed, Tempest gets to work cleaning up the orcs that remain in the fortress. Aurexis soars to the guard platforms, dealing with the Scouts who have bravely hid in the huts and fired down into the crowd.

5. Just when Tempest and Aurexis think the battle is won, and all the orcs are destroyed, a monstrous bellow erupts from the command hut, and out steps Kara'tok. With a spine-rattling cackle, Sul'talan descends from the heavens to perch on Kara'tok's fist. After a moment of stare-downs, the birds throw themselves at each other in mid-air, while Tempest nocks an arrow and begins the duel for the fate of her forest.

6. When Tempest and Aurexis manage to defeat the Orkdursk commander and his carrion-feasting minion, they can set about putting the forest to rights. At least, until General Hr'arak sends another troop to Neverwood...

Front-End Menu System



Layout

The drawing above represents a basic 3D model. Each thick black line represents a tessellated quad. The user will maneuver through the menus by making selections with the mouse. If the selection calls for moving to another menu, the camera will move from the quad currently showing the menu to the new quad, following the paths demonstrated by the curved lines. No free reign of motion is permitted in this system; the camera is moved “on a rope” to the new destination. Selecting “Play” will follow the path depicted, but as the camera moves forward, the 'forest' will fade to black and the load screen will display.

The model will appear to be a deep forest, using the same “sky box” textures from Level 1. As the camera is clamped to follow the line and face forward, any geometry that the camera will never be able to turn to see can and should be omitted in this model. The actual “menu” quads should appear to be carved into the sides of giant trees.

The cyan dot near the main menu represents a small rose bush, the only mandatory piece of detail geometry in the “level”. The bush should have several yellow roses in bloom, and in its branches is perched a 'redbird' or cardinal. This bird does not need to animate, move, or make noise, and can be modeled directly on the bush if this is more convenient to the artist.

Front-End Menu System

Main Menu

Play	**Loads Level 1**
Options	**Move to Options screen**
Credits	**Move to Credits screen**
Secrets	**Move to Unlockables**
Quit	**Fade to black, exit to Windows**

Options

Controls	**Move to Key Binding screen, which simply shows a list of actions to bind controls to**
Main	**Move to Main Menu**
Video	**Set screen resolution, any other needed video properties**
Sound	**Set SFX, music volumes**
Difficulty	**Easy, Medium, Hard, Ridiculously Hard**

Unlockables

Character Content	**Show texture map, T-pose model, and background story for each character, enemy**
Choose Character	**Pick a character to play as**
Character Outfit	**Select alternate textures**
Sound Test	**Play all sounds/music in the game**
Main	**Move to Main Menu**

Credits

A simple texture printed with the names and contributions would be sufficient here.

Coding Standards

Naming Conventions

As a general practice, all non-constant names will start with a capital letter. Each additional word in the name will be separated by another capital letter. Names will intuitively identify the purpose of the object/function being named.

Variables will be named using a loose form of Hungarian Notation. All built-in types, including pointers, are expected to follow this standard. In object names, Hungarian notation is not mandatory, with the exception of Vector ('v') and Matrix ('m'). After the Hungarian notation characters, variable names will start with a capital letter. Each additional word in the name will be separated by another capital letter. Member variables do not require the 'm_' prefix, nor do globals require '_g'. Padding variables should be named glaringly obviously what they are and be prefaced by three underscores, i.e.: char ___PaddingDoNotUse;

Constants and enumerations will be named in all capital letters. They should be named verbosely enough to ensure uniquity throughout the project. Declaring a type for enumerations is mandatory, but using it when declaring variables to hold those enumerations is not. Enumerations do not need to be prefixed with an 'e'.

Object names will be in accordance to the general practice. No Hungarian notation is to be used when declaring the name of a structure or class.

Banned Practices

The following programming techniques, etc. are forbidden in the main game build for purposes of speed and/or debugging transparency:

- The **double** data type
- The entire **Standard Template Library**
- The entire **Microsoft Foundation Class** Library
- Use of **dynamic_cast** (use type flags in base objects instead)
- Avoidable use of **new** and **delete** within the game loop

Coding Standards

Minimum Commenting Standards

While there is no mandatory header format for files or functions, the following practices will be observed:

- A comment at the top of each file, only so long as is necessary to convey the purpose of the file
- A comment at the top of each function declaration, explaining its purpose. This should include explanations of any non-obvious parameters, as well as any tips/warnings regarding the use of the function.
- A comment associated with any code block that is not immediately intuitive as to its purpose or workings.
- The following tags will be used for their associated situations. All tags will be followed by the programmer's name and the time/date. They will not be removed until the situation has been resolved. They are:

TODO:	Place this above or in place of any code that is not complete and requires further attention.
DEBUG:	Place this above any code that has not yet been thoroughly tested. If there is a known issue with the code block, mention that in the comment as well. If you debug someone else's code, add a **CHANGED:** tag.
CHANGED:	Place this above any block of *someone else's* code that you have changed. Include a detailed description of what you changed and why. This does not apply to the addition of new, independent functions or variables to existing classes that do not affect any existing code.
OPTIMIZE:	Place this above any code block that requires optimization. If you optimize someone else's code, add a **CHANGED:** tag.

System Architecture Overview

Guardians of Neverwood uses an object oriented C++ architecture. The following is a brief overview of the core components of Neverwood's game engine (see the flowchart that follows for a visual depiction):

Common Technology

Common Technology is the collection of all general-purpose modules that are used throughout the rest of the engine. Common technology includes, but is not limited to:

- **Memory Manager**

The memory manager gets used whenever memory is allocated dynamically in debug mode, exporting to a text file the total dynamic memory usage at any given point in the game's execution. At the end of the game's execution, any objects in the file will represent leaked memory.

- **Dynamic Arrays / Linked Lists**

It is our position that the Standard Template Library is cumbersome and inefficient for purposes of speed and debugging transparency. Instead, we have employed dynamically-declared arrays or custom-made (non-templated) linked lists on any time-critical processes.

- **Vector Library / Matrix Library / Math Utilities**

To facilitate the implementation of mathematical algorithms, Guardians of Neverwood's engine defines its own matrix and vector math libraries using overloaded operators. The math utilities contain other useful helper functions such as floating-point equality testing.

- **High-Resolution Timer**

The high-resolution timer is a simple singleton class that the main game loop uses to keep track of the system time at which frames and events occur. It counts frames per second and provides one central repository for all time-based events in the game, so that all events occur in synchronization.

- **Utilities**

Utilities are general-purpose constants, macros, or global functions used by several different modules. These include unit conversions, conversions between strings and integers, trigonometric tables, floating point comparison, and various different macros.

System Architecture Overview

Managers

The engine uses singleton manager classes generously in order to facilitate sharing data between modules. The following modules are represented as singleton classes

- **Game class**

The game class is the most important module in Guardians of Neverwood, as it contains the game's main loop. The main purpose of the game class is to communicate between the engine's managers (input, sound, rendering, etc.) and the game's entities. It is also responsible for handling game state and transitions between the menu system and the game.

- **Input Manager**

The input manager uses DirectInput 8.0 to provide basic support for keyboard, mouse, and gamepad input, as well as more advanced features such as dynamic action mapping/key binding and force feedback support.

- **Sound Manager**

The sound manager uses FMOD to provide basic playback ability for WAV, RAW, MP3, OGG, MIDI, and MOD files. Supports volume control and tracking / looping of sound effects and music.

- **Texture Manager**

The texture manager is responsible for loading and releasing textures based on a file name, registering them with OpenGL, and assigning them unique integer indices that can be used to reference the textures.

- **Model Loader:**

The model loader is used to retrieve geometry information from a proprietary file format created through the use of a Maya exporter. It supports raw vertex data for simple objects and static geometry, as well as models using a skin-and-bones architecture for purposes of animation.

- **Animation Manager**

Maintains and updates relevant data used to animate models using a skin-and-bones architecture. Able to transition an object between its animations and present positioned geometry to the renderer.

- **Front-End Menu System**

The front end menu system consists of the title screen, options (including key binds, video, and sound options), and access to unlockable items.

System Architecture Overview

- **Heads-Up Display**

The HUD consists of a small, semitransparent quad sprited flat against the camera's near clipping plane, placed in the lower right hand corner of a player's window. This display will contain information related to the player's currently selected weapon and the player's health and charge status.

- **Renderer**

Unlike most modules that communicate with the game class, the renderer is not a singleton. This is because the game supports split-screen mode, which will be implemented by using two renderers. The renderer's main purpose is to provide a wrapper class for OpenGL rendering functionality, sorting objects according to their rendering context and presenting them to the screen. All culling of geometry is performed in the world class, so the renderer can simply draw any geometry that it receives.

Abstract Modules

These modules are more complex than just a simple class. They represent a hierarchy of classes, or their functionality is embedded in other modules.

- **Object Hierarchy**

Objects are any classes that derive from the “Base” class. A partial list includes: player characters, enemies, power-ups, switches, containers, cameras, particle emitters, projectiles, and the world geometry itself. Many operations are provided in the Base class to give all game objects a common interface and functionality set.

- **Bounding Shapes**

Bounding shapes are the classes that derive from BoundingShape.h: spheres, cylinders, axis aligned bounding boxes, and frustums. Bounding shapes contain all functionality necessary for collision detection against other bounding shapes and world geometry.

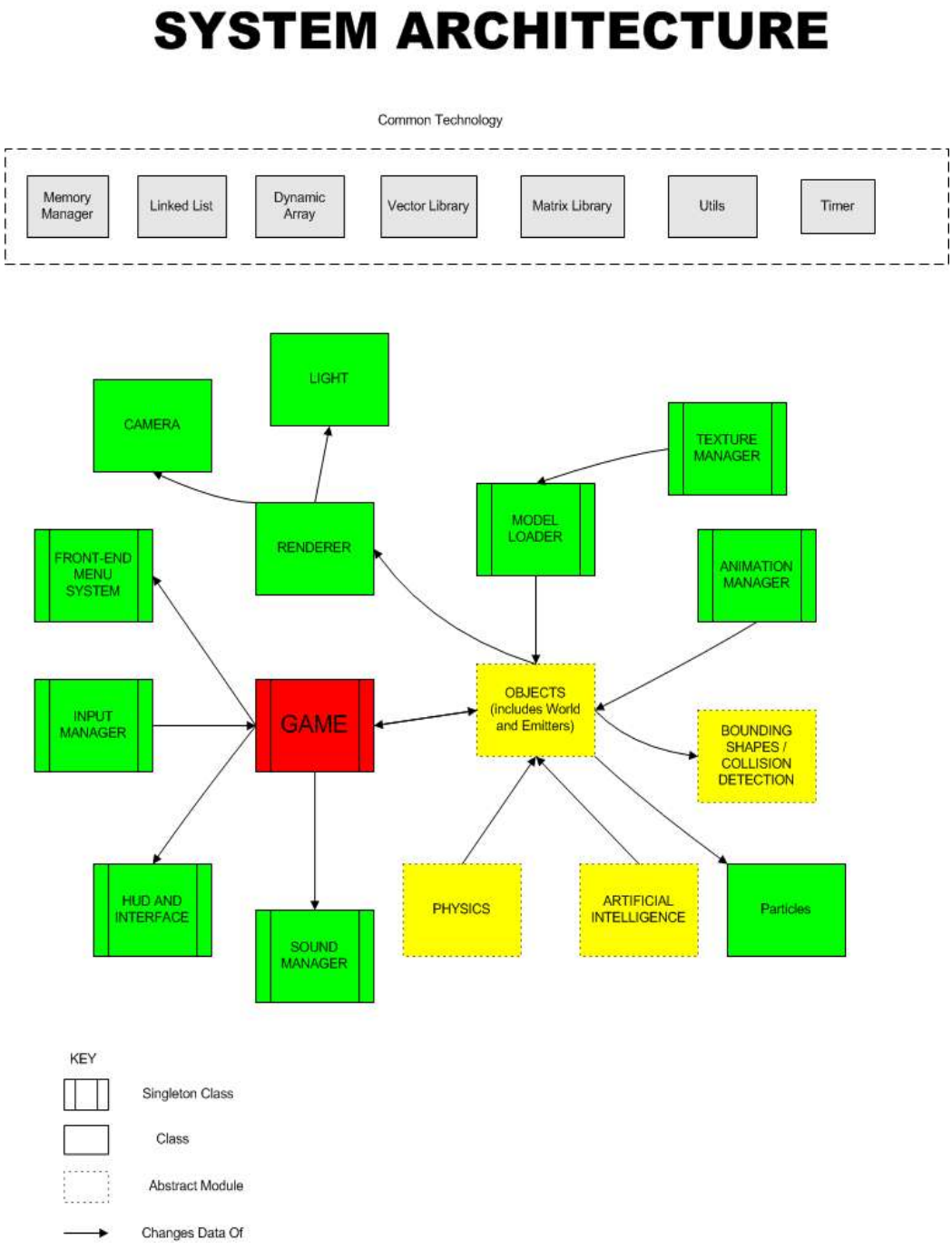
- **Physics**

The engine does not contain an explicit physics system. Rather, physics functionality is incorporated into the base class and used by the game class.

- **Artificial Intelligence**

As with the physics system, most AI is embedded into the base class. Some AI, like waypoints and patrol points, is stored inside the world.

System Architecture Overview



Base Entity Class

Uses	Used By
Vector, Matrix RenderInfo Vertex BoundingShape	Emitter (for particles) Camera Ranger Enemy (OrcScout, OrcCutter, OrcLt, KaraTok) Bird (Falcon, Raven, Buzzard) Container (chest, barrel, crate) World (the geometry of the world itself) Powerup (arrow upgrades, herbs, falcon power ups)

Key Features

- Provides wide range of functionality to increase the common interface and decrease fragmentation of code base.
- Contains necessary information to position, move, rotate and render an object.
- Encapsulates all matrix math related to the object
- Contains state and flags related to how the object should behave.

Key Member Variables

Matrix mGlobal	Defines the object's matrix in global space, which is the heart and soul of all our transformations.
INT iObjectType	Keep track of what type of item we are using flags in Identifier.h
Vertex *pGeometry	The object's loaded model for OpenGL.
UNSIGNED CHAR ucTextureID	Store an index to the object's texture map.
RenderInfo RenderContext;	Contains several flags that the renderer will use to properly present objects to the screen. For more information regarding this structure, please see the Renderer section.

Notes

The base class communicates to other modules through flags rather than states. Whereas a state represents a high-level description of what the object is doing (dying, resurrecting, etc.), a flag translates directly into code. The base class' state variable is entirely optional and is intended to be used as a more intuitive way to represent a collection of flags. The 'dying' state, for example, might include: disable input, disable collision, disable motion. Alternatively, the object's state and time in state can be used to transition between behaviors based on time.

Base Entity Class

Inventory of Key Functionality

RotateX, Y, RotateLocalX, Y, Z Rotate	FLOAT Degrees (Vector vAxis)	Rotates the object clockwise by a number of degrees around its global matrices. A seperate function is provided for each local axis. A fourth function, Rotate(), takes in a vector to serve as the axis of rotation. This function is slower, however, so should be used sparingly. RotateLocal functions work just as RotateX... do, but around *local* matrices.
LookAt	A point (store as Vector)	Apply the LookAt matrix manipulation, rotating the object to face the given point. This is useful for such tasks as camera following, targeting, and so forth.
TurnTo	A point (store as Vector)	Apply the TurnTo matrix manipulation, rotating the object to face the given point. This is useful for such tasks as camera following, targeting, and so forth.
FaceDirection	Vector vDirection	Turn so that the object's Z axis (forward vector) matches the direction of vDirection.
MoveTo	A point (store as Vector)	Moves the object to the given point in global space. (in meters)
MoveBy	Vector	Translates the object from its current position in a direction and distance given by the vector parameter. (in meters)
MoveForward MoveRight MoveUp	FLOAT Amount	Moves the object along its local X, Y, or Z axis by a number of meters defined by Amount.
Operator new ([]) Operator delete ([])	None	Overloaded in debug mode only to inform the memory manager of allocations/deallocations.
CollidesWith	Base *OtherBase Vector *vIntersect	Determines if the bounding volumes of the two base objects collide. If they do, return a boolean “true” and fill vIntersect with the intersection point.
CollisionClamp	Base *Other	Determines if two objects have collided, and if they have, position them so that they are just touching.
CollisionKnockback	Base *Other	Determines if two objects have collided, and if they have, knock them each back relative to their masses and velocities.
GroundClamp	None	Place the object on the ground of the world.

Base Entity Class

Key Virtual Functions

Initialize	None	This will perform all initialization necessary for the object, including nulling out values and declaring dynamic memory. Must be implemented for each derived class.
Shutdown	None	This will perform all shutdown necessary for the object, including nulling out values for safety and deallocating dynamic memory. Must be implemented for each derived class.
Update	None	This function will be called from within the game loop once for each frame that the object should update. The object will be updated with regards to: motion, collision, animation, AI, state changes, etc. Must be implemented for each derived class.
Draw	None	This draws the object to the screen. Should be called from the renderer once during each frame that the object should be visible. This function performs no culling or space partitioning; this should be done elsewhere. Must be implemented for each derived class.

Base Entity Class

```
#ifndef StormForge_Shared_BaseClass
#define StormForge_Shared_BaseClass

#include "Vertex.h"
#include “Model.h”
#include "Matrix.h"
#include "MemoryMgr.h"
#include "Vector.h"
#include "BoundingShape.h"

class Base
{
public:
    Base(int iObjType = INVALID);   // Generic constructor for making arrays of Base objects.

    // GEOMETRY
    Matrix mGlobal;                 // Our matrix as it relates to the global space
    Model *pModel;                  // our animated model
    float fWidth, fHeight, fDepth;  // dimensions of the object
    RenderInfo Rcontext;            // rendering context

    // COLLISION & MOTION
    Vector vVelocity;               // Where are we going and how fast? (scaled by time)
    BoundingShape *pShape;          // A bounding shape to use for collision
    float fMass;                    // For conservation of momentum, gravity, etc.

    // DATA SORTING AND STORAGE
    Base *pNext;              // For making linked-lists of Base-derived objects.
    int iObjectID;            // A unique object ID for this item.
    int iObjectType;          // Classification of what this object is - camera, powerup, etc...
    static int siObjectCount; // How many Base objects are active right now?
                              // Used for generating unique IDs.


    // FLAGS
    // Motion and gameplay
    int bCollidable : 1;        // Should this object be tested for collision?
    int bMovable : 1;           // Can this object move?
    int bControllable : 1;      // Does this object need to contain a Control structure? (AI or input)
    int bTakesDamage : 1;   // Can it be damaged or destroyed?
    int bGroundClamped : 1;// Should this object clamp to the ground as it moves?
    int bFollowsTerrain : 1;   // Should this object adjust to the terrain as it moves?
    int bGravityAffected : 1;  // Should this object accelerate downward due to gravity?
    int bCanChangeState : 1;// Can this object change states?
    int bIsLiving : 1;          // Is it a living object? (can it be "alive"?)
    int bUpdate : 1;            // Should this object receive updates?
```

Base Entity Class

```
// Extra bits used for data alignment
private:
        int ____PADDING_DO_NOT_USE_ : 13;
public:

// MEMORY MANAGER
#if _DEBUG
        ManageMemory(Base, BASETYPE_BASE);
#endif

// MOVEMENT FUNCTIONS
// puts the object to the given position
void MoveTo (Vector& vPoint) { memcpy (mGlobal.fMatrix + 12, vPoint.fComp, 12); }

// translates the object globally by a given amount
void MoveBy     (Vector& vPoint);

// moves the vector along its local axes
void MoveForward (float amount);
void MoveRight (float amount);
void MoveUp (float amount);

// rotate around the world's axes
virtual void RotateX (float fDegrees);
virtual void RotateY (float fDegrees);
virtual void RotateZ (float fDegrees);
virtual void Rotate (Vector& vAxis, float fDegrees);

// rotate around the object's local axes
virtual void RotateLocalX (float fDegrees);
virtual void RotateLocalY (float fDegrees);
virtual void RotateLocalZ (float fDegrees);

// more sophisticated movement algorithms
virtual void LookAt (Vector& vPoint);
virtual void FaceDirection (Vector& vDirection);
virtual void TurnTo (Vector& vPoint, float fAngle);

// MATRIX ACCESSORS (No modifiers needed since we're returning pointers) //
Vector GetPos(void)     { return mGlobal.GetPos(); }
Vector GetZ(void)       { return mGlobal.GetZ(); }
Vector GetY(void)       { return mGlobal.GetY(); }
Vector GetX(void)       { return mGlobal.GetX(); }
float GetXPos(void)     { return mGlobal.fMatrix[12]; }
float GetYPos(void)     { return mGlobal.fMatrix[13]; }
float GetZPos(void)     { return mGlobal.fMatrix[14]; }
```

Base Entity Class

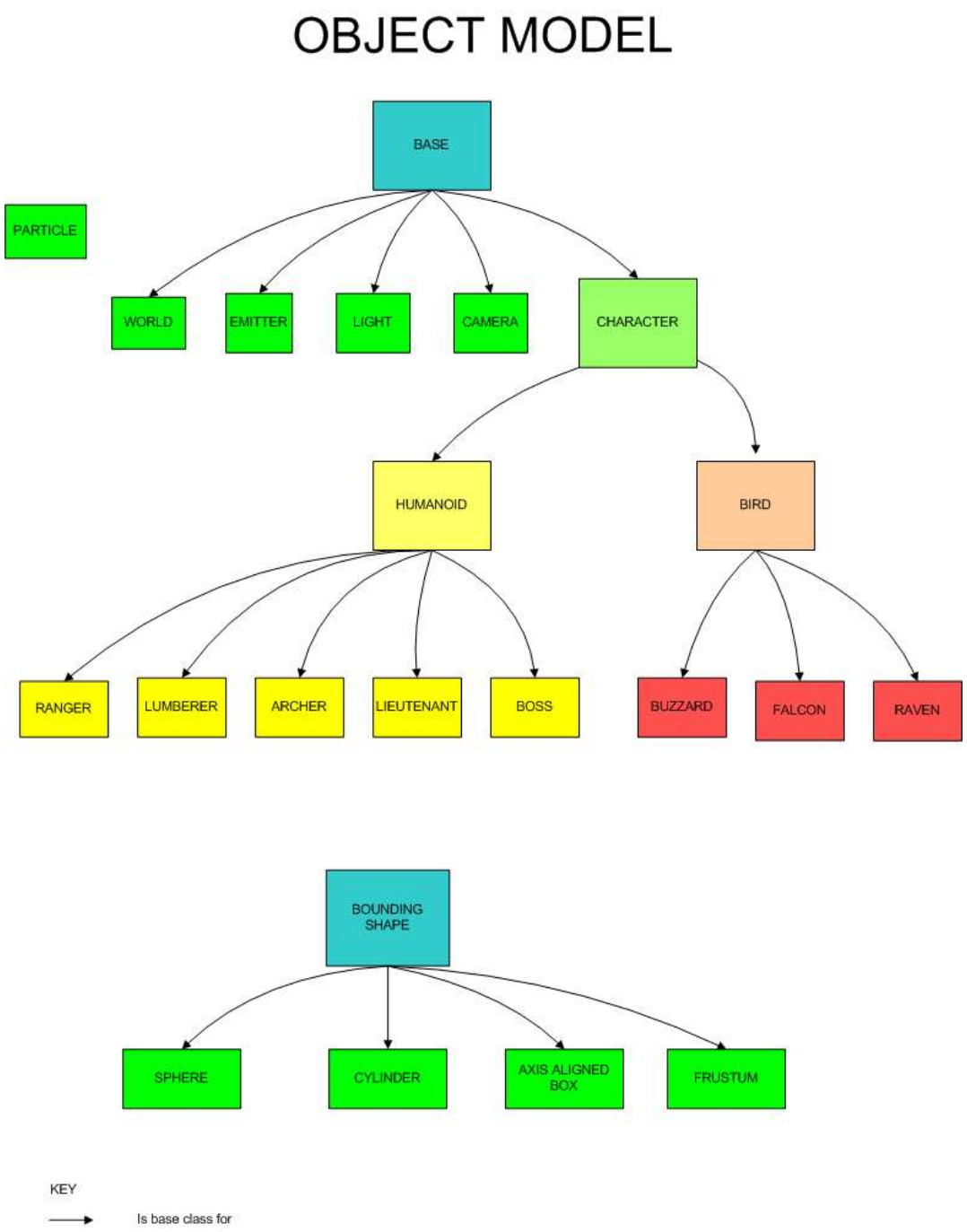
```
    // COLLISION DETECTION
    virtual bool CollidesWith (Base* other, Vector* vIntersection);
    virtual bool CollisionClamp(Base *bOther);
    virtual bool CollisionKnockback(Base *bOther);
    virtual bool CollisionPushback(Base *bOther);

    // GENERAL UPDATING
    virtual void Initialize(void);
    virtual void Shutdown(void);
    virtual void Draw(void);
    virtual void Update(void);

    // STATE MACHINE
    int iCurrentState;          // What state are we in now?
    float fStateTime;           // How long have we been in this state?

};
#endif
```

Object Hierarchy



Character Entity Class

Uses	Used By
Base (inherited) ActionMap	Humanoids (Ranger, OrcScout, OrcCutter, OrcLt, OrcBoss) Birds (Falcon, Raven, Buzzard)

Key Features

- Contains an ActionMap to control the character's actions
- Encapsulates AI states and core AI functionality
- Adds support to the Base class for variables related to living characters.

Key Member Variables

INT iHP, iMaxHP	Player's hit points and maximum hit points
INT iCharge, iMaxCharge	Player's charge state and maximum charge state
ActionMap *pActionMap	Stores all the potential actions of the character and the key bindings associated with them.
BOOL bEvil	If this is 'true', the creature is defined to be a monster
INT iAnimationIDs[]	Stores all of the player's animations for their actions
INT iSoundIDs[]	Stores all of the player's sound ID's for various actions

Inventory of Key Functionality

FindPathTo	Vector vPoint	Perform a pathfinding algorithm to determine an optimal path to get to vPoint from its current position.

Key Virtual Functions

OnHit	Character *pAttacker	React to having been hit. Includes taking damage, flinch animation, knockback, sounds, checking for death
ApplyAI	None	Update the character's AI mode. See the Artificial Intelligence section for more information
Die	None	Shuffle off this mortal coil while playing animations and sounds, and potentially kicking the player back to the menu, respawning the falcon, etc.
Attack	Base *pTarget INT iAttackType	Begin attack animation, play sound, set Attack state. Parameter is a Base in case we want to attack inanimate stuff like crates, walls, etc.

Vector Library

Uses	Used By
Math Utilities	Pretty much everything. Vectors are the backbone of 3D graphics math. They are used for every point, position, direction, motion, and color in the game.

Key Features

- Overloaded operators provide less tedious, more legible code.
- Provides support for 3D and 4D vectors
- Uses some inline X86 assembly for optimization where appropriate
- Adds some functionality not normally covered by math libraries

Key Member Variables

FLOAT fComp[3] (Vector) **FLOAT fComp[4] (Vector4D)**	Stores the components of the vector in the order X, Y, Z (,W).

Notes

Neverwood's vector library contains two classes, **Vector** (3-dimensional vectors) and **Vector4D** (4-dimensional vectors). Both rely heavily on overloaded operators to make operations easy to use.

Many of the function calls are prefaced with the Microsoft-specific compiler directive __fastcall. This directive causes parameters to be passed in registers as opposed to the call stack, and is said to increase efficiency. This will need to be profiled to determine if any real resource savings are present.

Vector Library

Inventory of Key Functionality

Operator + **Operator +=**	Vector v	+ : Adds two vectors and returns the result. += : Adds two vectors and stores the result.
Operator - **Operator -=**	Vector v	- : Subtracts two vectors and returns the result. -= : Subtracts two vectors and adds the result.
Operator - (Unary)	None	Negates a vector; returns <-Ax, -Ay, -Az>
Operator * **Operator *=**	FLOAT f	* : Scales a vector by the float and returns it. *= : Scales a vector by the float and stores it.
Operator *	Vector v	Returns a float containing the dot product of the two vectors : (Ax * Bx) + (Ay * By) + (Az * Bz)
Operator / **Operator /=**	FLOAT f	/ : Normalizes and scales the vector by the float, returns /= : Normalizes and scales the vector by the float, stores.
Operator & **Operator &=**	Vector v	& : Performs the cross product of the 2 vectors, returns. &= : Performs the cross product of the 2 vectors, stores. Cross: (AyBz – AzBy)i, (AzBx – AxBz)j, (AxBy – AyBx)k
Operator ! (Unary)	None	Normalizes and returns a vector.
Operator ~ (Unary)	None	Returns the magnitude of the vector as a float.
Operator % **Operator %=**	Vector v	% : Projects v onto this vector, returns new vector. %= : Projects v onto this vector, stores.
GetRandomVector	Vector v Vector w	Returns a vector of random float values, where each component is a float between v.component and w.component.
IsNormalized	None	Performs a quick length check and returns true if the length is within an epsilon value of 1.
IsOrthogonalTo	Vector v	Performs dot product, if value is within an epsilon value of 0, the two vectors are orthogonal. Return boolean true.
AngleTo	Vector v	Returns a float representing the angle, in degrees, between two vectors. Expensive; use sparingly.

Matrix Library

Uses	Used By
Vector, Vector4D	Pretty much everything. Matrices are responsible for all things that have position and orientation, including all objects derived from Base and lights. Matrices also can define translations, rotations and scales, so they are used heavily in animation and in general math algorithms.

Key Features

- Overloaded operators provide less tedious, more legible code.
- Uses some inline X86 assembly for optimization where appropriate
- Contains common matrix manipulations as member functions for easily maintaining objects

Notes

The Matrix class stores a 4x4 matrix, but many operations ignore the bottom row as an optimization.

Key Member Variables

FLOAT fMatrix[16]	Stores the components of the matrix, ***column major.***

Matrix Library

Inventory of Key Functionality

GetX, GetY, GetZ, GetPos	None	Return the contents of a row/column (seperate functions for each)
SetX, SetY, SetZ, SetPos	Vector &v FLOAT fX, fY, fZ	Replace the contents of a row/column (seperate functions for each)
GetRow, GetCol	INT n	Returns the requested row/column
Operator * and *=	Matrix &	Returns (*) or stores (*=) the result of a matrix multiply.
SetToIdentity	None	Fills the matrix with a 4x4 identity.
Invert	None	Inverts the matrix, stores. Returns 'false' if it is an un-invertible matrix.
Transpose	None	Transposes the matrix about its major axis
Translate **TranslateLocal**	FLOAT x, y, z or Vector v	Translates the matrix in global space (Translate) or local space (TranslateLocal)
RotateX, Y, Z **RotateLocalX, Y, Z** **Rotate**	FLOAT Degrees (Vector vAxis)	Rotates the object clockwise by a number of degrees around its global matrices. A seperate function is provided for each local axis. A fourth function, Rotate(), takes in a vector to serve as the axis of rotation. This function is slower, however, so should be used sparingly. RotateLocal functions work just as RotateX, Y, Z do, but around *local* matrices.
RotateTowards	A point (Vector) FLOAT fAngle	Rotates toward the point by fAngle (in degrees).
LookAt	A point (store as Vector)	Apply the LookAt matrix manipulation, rotating the object to face the given point. This is useful for such tasks as camera following, targeting, and so forth.
TurnTo	A point (store as Vector)	Apply the TurnTo matrix manipulation, rotating the object to face the given point. This is useful for such tasks as camera following, targeting, and so forth.
FaceDirection	Vector vDirection	Turn so that the object's Z axis (forward vector) matches the direction of vDirection.
Scale	FLOAT Scale	Scales the matrix by a factor of Scale.
CheckOrthoAxes	None	Returns 'true' if the X, Y, and Z axis are orthonormal (perpendicular to each other)
NormalizeIfNeeded	None	Performs a quick check (IsNormalized()) on each axis, and calls Vector::Normalize() on any that are not unit length.

Camera

Uses	Used By
Base (inherited) Vector, Matrix BoundingShape World (to transform)	Renderer World Emitter (particles)

Key Features

- Encapsulates the world transform to display world in camera's perspective.
- Encapsulates and orients the view frustum.
- Provides manipulations for various camera effects.

Key Member Variables

INT iCameraMode	Stores which camera mode (Hard Attach, Soft Attach, etc.) the camera is currently running in.
Frustum instance (Dynamic; using BoundingShape *pShape inherited from Base)	This is the actual view frustum for this camera. Objects will be collision checked against it to determine visibility.
INT iAttachedID	The unique ID of the object the camera is attached to.
Matrix mQueue[10]	A backlog of matrices used to create padding in Soft Attach mode.

Inventory of Key Functionality

HardAttachTo	Base *AttachTo, Vector vOffset	Performs the Hard Attach algorithm to link the camera to an object.
SoftAttachTo	Base *AttachTo, Vector vOffset	Performs the Soft Attach algorithm to link the camera to an object.
MouseLook	None	This will use the DirectInput mouse position to determine which direction to use. This camera mode is popular with fans of first-person shooters.
FaceDirection LookAt	Vector v	These algorithms cause the camera to look at a certain position. See the Matrix class for details.
TransformWorld	none	This will position the world appropriately so as to appear that the camera is moving around within it.

Animation & Model Manager

Uses	Used By
Model structure Animation structure Vector & Matrix Timer	World Base objects Renderer

Key Features

- Reads a formatted model file and extracts model data
- Packages model into game-ready data structure
- Applies skin-and-bones animation to the model and presents the deformed geometry to the renderer to be drawn
- Assigns integer ID's to animations and models, allowing them to be referenced by index alone in other modules
- Contains all local-to-global translation math, so that the only matrix each object needs to store permanently is its global matrix.

Key Member Variables

Animation pAnims[]	Array of all the animations currently loaded
Model pModels[]	Array of all models currently loaded

Inventory of Key Functionality

Initialize	None	Sets up the manager for initial use.
Shutdown	None	Frees all memory allocated and exits gracefully.
LoadModel	CHAR *strFile	Opens the given file and parses the model into memory. Returns an integer ID to its data.
LoadAnimation	CHAR *strFile	Opens the given file and parses the animation into memory. Returns an integer ID to its data.
Animate	Base *pBase	Queries the Base object for its current animation and frame, generates the local matrices and performs all vertex transforms. Uses spherical linear interpolation (SLIRP) to achieve this. Fills pBase's model information with the transformed copy of the model, which will then be used when pBase's Draw() function is called by the renderer.

Model Structure

Uses	Used By
None Vector	Model loader Base objects Renderer

Key Features

- Stores a model's data so that it can be used in a call to OpenGL's indexed vertex array capability.

Key Member Variables

Vector pvPosition[]	Array of vertices' positions
Vector pvNormal[]	Array of vertices' normals
Vector4D pvColor[]	Array of vertices' colors
FLOAT pfTexCoord[][2]	Array of vertices' texture coordinates
INT piIndices[]	Array of vertex indices that OpenGL will use to draw
INT iNumVerts	Number of vertices in the above arrays
INT iNumIndices	Number of indices in piIndices

AnimationStructure

Uses	Used By
None Vector & Matrix Model	Model loader Base objects Renderer

Key Features

- Stores a model's animation frames so we can move the model around.

Key Member Variables

Line pKeyframes[K][B]	An 2D array of lines. Since a Line is nothing but two points, it is an adequate way to represent a bone. This array stores a number of keyframes K, each containing positions for a number of bones B.
INT iKeyframes, iBones	Number of keyframes and bones, respectively

Light Manager

Uses	Used By
Base (face objects) OpenGL API Vector, Matrix Lights	Renderer Emitter (particles)

Key Features

- Provides a global, singleton-type interface for lights.

Key Member Variables (all are Static)

Light ** pLights	Stores a dynamic array of light pointers. Using array of pointers so unused lights only waste 4 bytes of memory.
Vector4D vEmission	This determines the “glow” of objects. It is a material property that should be turned on for items that are intended to glow, and shut off by default.
BOOL bColorMaterial	Reflects whether or not OpenGL is in Color Material mode. This is 'true' by default.
CHAR icShininess	Reflects the Shininess material property value.
Vector4D vSceneAmbient	This is the “background ambient” light in the scene.
BOOL bLighting	This is 'true' if GL_LIGHTING is enabled.

Inventory of Key Functionality (all are Static)

Enable **Disable**	BOOL bEnable	Enables/Disables OpenGL lighting.
SetEmission **SetSceneAmbient**	Vector4D v	Sets the associated light / material vector and establishes the change with OpenGL.
Initialize	None	Sets up general lighting parameters in OpenGL
ColorMaterial	BOOL bEnable	Toggles Color Material mode.
SetShininess	CHAR icShiny	Sets the shininess value.

Light

Uses	Used By
Base (face objects) OpenGL API Vector, Matrix	Renderer Emitter (particles) Light Manager

Key Features

- Wraps the OpenGL light and its associated values.
- Allows for context checks rather than blindly setting values in OpenGL, which saves many redundant state settings per frame by verifying that the context needs to be changed before we waste OpenGL's time to do so.
- Allows for manipulation of lights to match other objects.

Key Member Variables

Vector4D vAmbient **Vector4D vDiffuse** **Vector4D vSpecular**	Store the ambient, diffuse, and specular components of the light.
CHAR icLightID	The hardware light ID, i.e. GL_LIGHT0.
BOOL bEnabled	Reflects the enabled/disabled state of this hardware light.
FLOAT fSpotAngle	This is the angle representing the cone of light coming from a spotlight, in degrees.
FLOAT fSpotExponent	This is the exponent used for spotlights.
BOOL bSpotlight	This is 'true' if the light is acting as a spotlight. Note: A spotlight's direction is set as the Forward vector.
Matrix mLight	Stores the light's matrix, though we really only are interested in the forward vector and the position
Vector vAttenuation	Stores the three attenuation values in the following order: CONSTANT, LINEAR, QUADRATIC.

Light

Inventory of Key Functionality

Enable / Disable	BOOL bEnable	Enables/Disables the light.
SetSpotAngle **SetSpotExponent**	FLOAT fValue	Sets the associated spotlight value and establishes the change with OpenGL. Will turn on Spotlight mode if it's not on already.
SetAttenuation	Vector vAtten	Sets the attenuation values in this order: CONSTANT, LINEAR, QUADRATIC
Spotlight	BOOL bEnable	Enables/Disables spotlight mode.
MakePositional	BOOL bPosition	Set whether or not this is a positional light.
Initialize	None	Sets up the light with OpenGL.
FaceDirection	Vector v	Cause the light to shine at a certain position. See the Matrix class for details. This is only appropriate for spotlights.
LookAt	Base *pObject	Face an object. Great for spotlights.
MoveTo **MoveBy**	Vector v	Move to a location (MoveTo) or move a certain amount (MoveBy) and update the light position.

Emitter

Uses	Used By
Base (inherited) OpenGL API Vector, Matrix Particle Timer Camera World (get gravity) ParticleUpdateInfo	Particles Projectiles (Arrow...) World (Visual effects) Menu (Mouse cursor) HUD (Show special effects on weapons) Swords (Whoosh effect)

Key Features

- Contains and manages particles in a system
- Simple interface of velocities and forces can create almost any effect
- Reads effects from binary files

Key Member Variables

Vector4D vColorRangeMin **Vector4 vColorRangeMax**	These vectors specify the minimum and maximum values for each component of a new particle's color vector.
Vector vVelocityRangeMin **Vector vVelocityRangeMax**	These vectors specify the minimum and maximum values for each component of a new particle's velocity vector.
Vector vPositionRangeMin **Vector vPositionRangeMax**	These vectors specify the minimum and maximum values for each component of a new particle's position.
Vector4D vTextureCoordX **Vector4D vTextureCoordY**	Specifies the texture coordinates of the quads we will generate. Each index in the X, Y represents a pair of coordinates, beginning at the upper left and going counter-clockwise.
FLOAT fQuadHalfHeight **FLOAT fQuadHalfWidth**	Half the dimensions of the quads we will build. Used in generating the quads.
Particle *pParticles	Our inventory of Particles.
UNSIGNED INT iuParticles	Number of particles we have
FLOAT fLifeMin, fLifeMax	The minimum and maximum values for a new particle's life span.
UNSIGNED INT iuLastUpdated	The last particle that got an update. Used when we are scaling effects.
FLOAT fEmitRate, fInverseRate	Rate to emit particles; fInverseRate = 1/fEmitRate.
BOOL bEmitting	'True' if the emitter is currently spawning.
STATIC FLOAT sfRateScan	The percentage of the total number of particles we want to update each frame. I.e., if this is 1.0, every particle will be updated every frame.

Emitter

Notes

Each time the emitter's Update() function is called, it retrieves the camera's matrix. From this, it extracts the Up (Y axis) and Right (X axis) vectors, scaling them by half the desired size of the quad. By adding these values (and their negations) to the center position, four coordinates can be generated. These coordinates, along with the negated Forward (Z axis) of the camera representing the polygon's normal (for lighting), can be used to generate a quad on the fly. This quad is stored in four vertices in the vertex buffer. When all particles have updated, a single call to glDrawArrays() renders the information, dumping all the particles across the AGP bus in one chunk.

Inventory of Key Functionality

ClearGeometry	UNSIGNED INT i	Zeros out the vertex buffer members associated with the given particle
PRIVATE ReadEffect	CHAR *strFileName	Loads an effect file into the emitter.
Initialize	CHAR *strFileName	Initializes the emitter. Calls private function ReadEffect to load the effect file.
Update	Camera *pCamera	Updates the values of the particles. Called once per frame.
Draw	None	Draws the particles in one batch using glDrawArrays
Shutdown	None	Stops the emitter and frees its memory
STATIC ScaleEffects	FLOAT fScale	Scales the percentage of particles to update.
ApplyForce SetForce	Vector vForce	Adds a force to the system (ApplyForce) or sets the system's force vector directly to a value (SetForce).

ParticleUpdateInfo

Uses	Used By
Vector, Vector4D Vertex	Particle Emitter

Key Features

- Simple structure for passing information about particle updating to the Particle class without taking up an enormous amount of call stack to do it.

Key Member Variables

Vector4D vColorDelta	Specifies how much to change the color value of the particles, per second
Vector4D vTextureDeltaX, Y	Specifies how much to change the texture coordinates of the particles, per second
Vector vUp, vRight, vNormal	See notes in the Emitter section for how these are determined. They are used to build the quads that make up the particles.
Vector vGravity, vForce	External forces that are scaled by time and added to the existing velocity.
FLOAT fTime	The time that has passed since last frame.
Vertex *pGeometry	This is the same pointer as the Emitter's pGeometry, and is used to store the vertex buffer into which our particles are drawn
BOOL bSpiral **BOOL bGravity** **BOOL bApplyForce**	These boolean values represent the states of special effects that can be turned on or off.

Particle

Uses	Used By
Vector, Vector4D Vertex ParticleUpdateInfo	Emitter

Key Features

- Generates a quad each frame based on the camera's matrix
- Communicates with an OpenGL vertex buffer to get quads drawn

Key Member Variables

Vector4D vColor	Contains the particle's current color
Vector vVelocity	Contains the particle's current velocity vector
Vector vPosition	Contains the particle's current position (at the center)
FLOAT fLife	How much longer the particle has to exist.

Inventory of Key Functionality

Clear	None	Zeros out the particle's memory to recycle it.
Update	ParticleUpdateInfo *pInfo UNSIGNED INT i	Updates the particle at index 'i' in the Emitter's array using the data contained in pInfo.

DirectInput Manager

Uses	Used By
DirectInput 8.0 API CInputKeyboard CInputMouse CInputJoystick	CInputKeyboard CInputMouse CInputJoystick ActionMap *Any arbitrary need of system input throughout the game.*

Key Features

- Encapsulates initialization, polling, and shutdown of all input devices

Key Member Variables (all are Static)

CInputKeyboard idKeyboard **CInputMouse idMouse** **CInputJoystick idJoysticks[2]**	These objects, all derived from CInputDevice, will hold the wrapper objects for the keyboard, mouse, and joystick(s) respectively.
LPDIRECTINPUT8 *pDInputObject	The DirectInput object instance.
HWND hWnd **HINSTANCE hInstance**	Information about our main window; used primarily for initialization

Notes

The class is fairly basic, with the exception of initialization and shutdown. Most of the functionality is encapsulated within the devices themselves. However, the user will rarely need to use the full interface over the ActionMap interface.

Inventory of Key Functionality (all are Static)

Initialize	HWND HWnd, HINSTANCE HInstance	Initializes the DirectInput object and calls the initialization for all devices. Returns 'false' if anything goes wrong.
Shutdown	None	Shutdown the DirectInput object and calls the initialization for all devices. Returns 'false' if anything goes wrong.
Poll	None	This will handle all re-acquisition and filling of all buffers for every active device.
CALLBACK, FRIEND EnumerateJoysticks	DIDEVICEINSTANCE *pDIInstance, void *pVoid	Gathers information about the joysticks connected to the system and helps to initialize them.

CInputDevice

Uses	Used By
DirectInput 8.0 API	CInputKeyboard CInputMouse CInputJoystick

Key Features

- Encapsulates initialization, polling, & shutdown of individual input devices
- Encapsulates reading immediate and buffered input from the device
- Handles re-acquisition if lost

Key Member Variables

LPDIRECTINPUTDEVICE8 pDevice	The pointer to the DirectInput device COM object
DeviceType (enum) eDeviceType	Flag used to identify which type of device this instance is
DIDEVICEOBJECTDATA InputBuffer[20]	The buffer to store buffered input in.
DWORD dwBufferSize	The number of elements in InputBuffer.
BOOL bExclusive, bBackground	Flags concerning the cooperation level

Inventory of Key Functionality

FillBuffer	None	Pulls the data from the buffer and stores it in InputBuffer. Called internally by Poll().
CooperativeLevel	BOOL BExclusive BOOL BBackground	Sets the cooperative level of the device. Called internally by the child class' Initialize().

CInputDevice

Virtual Functions

Initialize	HWND hWnd, LPDIRECTINPUT8 pDInput	Perform all initialization necessary for the device.
Shutdown	None	Perform all shutdown necessary for the device.
Poll	None	Checks for acquisition and fills device's data buffer.
CheckImmediateInput	DWORD dwInput	Returns nonzero if the input indicated in dwInput (example: DIK_SPACE) was found in the immediate input stream.
CheckBufferedInput	DWORD dwInput	Returns nonzero if the input indicated in dwInput (example: DIK_SPACE) was found in the input buffer.

CInputKeyboard

Uses	Used By
DirectInput 8.0 API CInputDevice	DirectInput Manager *Any function needing direct access to the keyboard.*

Key Features

- Encapsulates all functionality related to the keyboard

Key Member Variables

CHAR cKeyBuffer[256]	Stores the immediate-mode keyboard input.
BOOL bKillWindowsKey	If this is 'true', Windows won't respond to the Windows logo key.

Inventory of Key Functionality

DisableWindowsKey	HWND hWnd, BOOL bWinKey	Toggle the functionality of the Windows logo key.
Additionally, all virtual functions from CInputDevice are implemented.		

CInputMouse

Uses	Used By
DirectInput 8.0 API CInputDevice	DirectInput Manager *Any function needing direct access to the mouse*

Key Features

- Encapsulates all functionality related to the mouse

Key Member Variables

DIMOUSESTATE2 MouseState	Stores the immediate-mode mouse input.
BOOL bShowWindowsCursor	If this is 'true', Windows will draw the mouse cursor
INT nFakeX, nFakeY, nFakeZ	These values represent where DirectInput's mouse cursor is.

Inventory of Key Functionality

ClickInRect	DWORD dwInput RECT &rClickZone	Returns 'true' if the button identified by 'dwInput' was clicked inside 'rClickZone'.
GetCursorPosition	None	Returns a POINT containing the mouse's X and Y position, according to DirectInput
GetScroll	None	Returns the scroll value (nFakeZ)
ShowWindowsCursor	BOOL bCursor	Toggles the Windows cursor display.
Additionally, all virtual functions from CInputDevice are implemented.		

CInputJoystick

Uses	Used By
DirectInput 8.0 API CInputJoystick	DirectInput Manager *Any function needing direct access to the mouse*

Key Features

- Encapsulates all functionality related to the joystick
- Encapsulates and manages all force feedback-related information

Notes

Buffered input of the joystick axes is not available at this time. Buttons, POV hats, and directional pads are fully supported.

Key Member Variables

DIJOYSTATE2 JoystickState	Stores the immediate-mode joystick input.
ForceFeedbackEffect *pFeedbackEffects	Linked-list of force feedback effects we've loaded.
INT nPointerX, nPointerY **FLOAT fPointerSensitivity**	These values represent where a joystick's cursor would be, when we use the joystick like a mouse.
GUID JoystickGUID	The DirectInput GUID for this joystick device.
INT nMaxX, nMaxY, nMaxRX, nMaxRY	The ranges for all four of the analog axes.
INT nDeadZone	The size of the dead zone around stick center.
BOOL bForceFeedback	Stores 'true' if force feedback is enabled.
BOOL bSupportsForceFeedback	Stores 'true' if force feedback is supported.

Inventory of Key Functionality

SimplifyPOV	DWORD dwPOV	Simplifies POV input from hundreths of a degree to one of nine enumerated values.
GetCursorPosition	None	Returns a POINT containing the joystick's X and Y position, according to DirectInput
FeedbackRunEffect	INT iEffectID BOOL bSoloEffect	Plays effect with ID 'iEffectID'. If 'bSoloEffect' is true, stop all other effects first.
FeedbackLoadEffects	CHAR *strFileName	Loads all feedback effects in the given file.
FeedbackStop	None	Stops all feedback effects currently playing
Additionally, all virtual functions from CInputDevice are implemented.		

CAction

Uses	Used By
None	ActionMap

Key Features

- Encapsulates a single action and its key, mouse, and joystick bindings
- Allows for easy key remapping, identification, and input checking

Key Member Variables

CHAR czActionName[21]	The name of this action as it will display on the options screen.
DWORD dwCurrent[3]	Stores the current key binding for the keyboard, mouse and joystick, respectively.
DWORD dwDefault[3]	Stores the default key binding for the keyboard, mouse and joystick, respectively.
INT iIndex	The index of this action in its parent ActionMap.

Inventory of Key Functionality

GetCurrent **GetDefault**	DeviceType eDevice	Returns the current or default key binding for the device given (keyboard, mouse or joystick)
SetCurrent **SetDefault**	DeviceType eDevice	Sets the current or default key binding for the device given (keyboard, mouse or joystick)

ActionMap

Uses	Used By
CInputJoystick CInputMouse CInputKeyboard CAction	Ranger Falcon OrcBoss (when playing as Kara'Tok) Raven (when playing as Sul'talan)

Key Features

- Encapsulates all inputtable actions for a character or situation
- Allows for one-line checking to determine if an action has occurred
- Allows for simple remapping of all input devices

Key Member Variables

CAction *pActions	A linked-list of possible Actions within this action map
INT iActionCount	The number of Actions in pActions.
INT iJoystickIndex	Which joystick to poll for input toward this Action Map.

Inventory of Key Functionality

AddAction	CHAR *strActionName DWORD dwKeyboard DWORD dwMouse DWORD dwJoystick	Adds an action to the map with the given name and default bindings for each device as given. Sets the current bindings to the defaults.
CheckAction	INT iIndex DeviceType eDevice BOOL bBuffered	See if the action at pActions[iIndex] happened on the given device (default: check them all). Uses buffered by default.
ActionName	INT iIndex CHAR *strStoreHere	Stores the “display” name of the action at pActions[iIndex] in strStoreHere.

Timer

Uses	Used By
QueryPerformanceCounter() LARGE_INTEGER	All time-based activities in the game, including particle and AI updating, frame timing, and animation

Key Features

- Provides a singleton interface to the timer so all events are synchronized
- Handles time difference internally, returning the user a simple difference as a float

Key Member Variables (all are Static)

LARGE_INTEGER liCount	The actual high-resolution time to compare against.
FLOAT fFrameTime	The amount of time the last frame took, in seconds
FLOAT fTimeOfLastFrame	The time at which the last frame happened
SHORT isFrames	The number of frames executed in the last second.
FLOAT fFrequency	1 divided by the PerformanceCounter's frequency. This is stored as an optimization over performing the divide every frame.
FLOAT fTimeScale	Scale factor for time. 1.0f by default. By lowering this number, we fool the computer into giving us a lower frame rate, for slow-mo.

Inventory of Key Functionality (all are Static)

Initialize	None	Starts the timer.
AppTime	None	Returns the time the game has been running
Update	None	Called once per frame to update the timer's values
GetFPSCount	None	Returns the frames executed in the last second.
GetElapsedTime	None	Returns the time since the last frame ran
CompareTime	FLOAT fCompareTime	Returns the time between now and fCompareTime.
SetMotionSpeed	FLOAT fSpeed	Set the speed scale. Clamped to 0 <= fSpeed <= 1.

Memory Manager

Uses	Used By
MemoryNode structure	All game entities and managers that can be declared dynamically register with the Memory Manager to

Key Features

- Tracks all dynamic memory allocation in debug mode
- Uses overloaded new and delete operators for ease of use
- Provides on-demand logging of active memory and leaks

Key Member Variables (all are Static)

UNSIGNED INT iuTotalMem	The total amount of dynamic memory allocated at the moment
UNSIGNED INT pItemCount	This array keeps count of how many of each type of entity (as declared by enumeration) are allocated at present
UNSIGNED INT iuNodeCount	The number of MemoryNodes in the system.
MemoryNode *pNodes	The array of MemoryNodes.
MemoryNode *pLastNode	The last node that was allocated, for fast access to add more.

Inventory of Key Functionality (all are Static)

Initialize	None	Starts the manager.
New	VOID *pAddress, size_t szSize, INT iTypeSize, UNSIGNED CHAR isType, CHAR *strFile INT iLine	Registers a memory allocation with the memory manager. Creates a MemoryNode, storing the type of the entity, a pointer to it, its size, and the file and line number on which it was allocated. Also adds to the entity counts in pItemCount[]. Called internally by the overloaded *operator new* and *new [ ]*.
Delete	VOID *pAddress	Removes the MemoryNode associated with this memory address from the manager.
Shutdown	None	Deallocate and shut down the memory manager
Dump	OSTREAM &oStream	Writes a summary of the current memory state to the given output stream.
DumpToFile	None	Writes a summary of the current memory state to MemoryNode.txt.

MemoryNode

Uses	Used By
None	Memory Manager

Key Features

- Provides a data structure to store information about memory allocations

Key Member Variables

MemoryNode *pNext	Provides linked-list functionality
VOID *pAddress	A pointer to the memory that was allocated
CHAR *strFile	The file name in which the memory was allocated.
INT iLine	The code line on which the memory was allocated
BOOL bActive	Indicates whether or not this node represents currently allocated memory or is available for use.
UNSIGNED SHORT isType	The type of the entity, corresponding to the enumerations in Identifiers.h
UNSIGNED INT iuSize	Stores sizeof(Entity)
UNSIGNED INT iuCount	The number of Entities represented in this node. Used when memory is declared with operator new [ ].

Texture Manager

Uses	Used By
OpenGL API	Renderer

Key Features

- Assigns loaded textures an index to which to bind
- Checks the current texture to prevent redundant binding
- Provides file loading for bitmaps and Targa files.

Key Member Variables (all are Static)

UNSIGNED INT *textures	This array stores the texture ID's that OpenGL gives us.
UNSIGNED INT unCurrentTexture	The ID of the currently bound texture
UNSIGNED INT unTextureCount	The number of loaded textures.
UNSIGNED INT unTexturesAllocated	The number of textures allocated at initialization.

Inventory of Key Functionality (all are Static)

Initialize	None	Starts the manager.
LoadTexture	CHAR *strFile	Dependent on which file type is passed, calls either LoadTextureBmp or LoadTextureTga to load the file and assign it an ID. Returns the ID.
Bind	UNSIGNED INT uTexture	Binds to the texture with this ID, if it is not already the actively bound texture.
Shutdown	None	Deallocate and shut down the manager
LoadTextureTga	OSTREAM &oStream	Loads a Targa file and assigns it an ID.
LoadTextureBmp	CHAR *szFileName	Loads a bitmap and assigns it an ID.

Quadric Manager

Uses	Used By
OpenGL API	Renderer Abstract, geometry-less entities, such as Emitter, Light, and Camera

Key Features

- Encapsulates OpenGL utility quadric objects
- Provides easy drawing of primitive shapes.
- Provides a convenient way to indicate the position of invisible objects.

Key Member Variables (all are Static)

GLUquadricObj *pObject	Pointer to an OpenGL Quadric Object.
BOOL bWireframe	If this is 'true', always draw in wireframe mode.

Inventory of Key Functionality (all are Static)

Initialize	None	Starts the manager.
Shutdown	None	Deallocate and shut down the manager
AutoTexture	BOOL bFlag	Toggles automatic texture coordinate generation. By default, this value is 'true'.
Disk	FLOAT innerRadius FLOAT outerRadius INT slices INT loops	Draws a disk with the specified values.
Sphere	FLOAT radius FLOAT slices FLOAT stacks	Draws a sphere with the specified values.
Cylinder	FLOAT baseradius FLOAT topradius FLOAT height FLOAT slices FLOAT stacks	Draws a cylinder with the specified values.
Wireframe	BOOL bWire	If 'true', sets the quadric manager to always draw in wireframe mode.

Sound Manager

Uses	Used By
fmod API soundData	Objects that need to play sounds Game class for looping music Menu system

Key Features

- Supports .wav, .raw, .mp3, .ogg, .mp2, .midi, and .mod files
- Supports global and per-sound volume control
- Assigns integer indices for loaded sounds
- Looped playback
- Tracking (skipping to a specified offset of a sound file or looping to a point other than the start of the file)

Key Member Variables (all are Static)

soundData *pSounds[]	Index of soundData structures containing sound information.
INT iSoundCount	The number of loaded sounds.

Notes

The sound manager uses fmod to support three different types of file playback:

1. Raw playback
With raw playback, the entire sound file is loaded into and played from memory. Raw playback provides high speed at the cost of high memory usage, and is thus best suited for short, frequently used sounds. By default, the sound manager uses raw playback for the .wav and .raw file formats

2. Streamed playback
Streamed playback uses a buffer to load only small sections of a sound at a time. The result is that the sound plays while using close to zero memory, but doing so uses up CPU cycles. Streamed playback is best suited for large files such as mp3's or other non-sequenced music files. By default, the sound manager uses streamed playback for the .mp3, .ogg, and .mp2 file formats

Sound Manager

Notes (continued)

3. Sequenced playback
Sequenced playback is used for highly compressed music files that store information in terms of instruments and sequences of notes. It provides low memory usage as well as low hard disk usage, but often at the cost of low sound quality. Sequenced playback is used for .midi and .mod files.

Since fmod uses different data structures to keep track of these different forms of playback, the main purpose of the sound manager is to abstract away from these details as much as possible. In most cases, it should suffice to call LoadSound and PlaySound, and the sound manager will make the correct decisions based on the file's extension. Another purpose of the sound manager is to provide global controls for sound and music.

Inventory of Key Functionality (all are Static)

Initialize	None	Starts the manager.
Shutdown	None	Deallocate and shut down the manager
LoadSound	CHAR *filename INT volume BOOL looped	Loads a sound file and returns its index. Determines which type of sound it is and calls the appropriate load function: .wav**/.raw:** Uses raw playback and marks the sound as a sound effect **.mp3/.mp2/.ogg:** Uses streamed playback, marks the sound as a music file and loops. **.midi/.mod:** Uses sequenced playback, marks the sound as a music file, and loops.
Play	INT soundID	Plays the sound at the given index.
StopMusic	None	Stops all music currently playing
SkipTo	INT soundID INT time	Plays the sound at the given index, starting at 'time' milliseconds into the sound.
GetVolume	INT soundID	Returns the given sound's volume setting.
SetVolume	INT soundID INT volume	Set the given sound's volume.

Renderer

Uses	Used By
OpenGL API LightManager & Light TextureManager RenderInfo Base & World Camera	Base-derived objects that need to be drawn Heads-Up Display Front-End Menu World

Key Features

- Provides a wrapper class for OpenGL's geometry rendering functions
- Sorts objects into queues to accelerate rendering
- Enables outputting information for debugging
- Supports split-screen view for two-player mode

Key Member Variables

RenderInfo RContext	The current contexts that the renderer is in. This should match the current OpenGL settings at all times. Ideally, no other module would perform OpenGL context changes.
HDC hDC **HGLRC hRC**	The device context and OpenGL rendering context for this window.
RECT rWindow	Store the dimensions of our portion of the window
CHAR cPlayerMode	Will determine if this renderer owns the top or bottom half of the screen, or the whole screen.
RenderInfo *pMain **RenderInfo *pTransparent** **RenderInfo *pWireframe**	Doubly-linked lists acting as priority queues for regular objects (pMain), transparent/alpha blended objects (pTransparent) and objects to be drawn in wire frame (pWireframe)
Camera *pCamera	The camera whose perspective we are rendering.

Notes

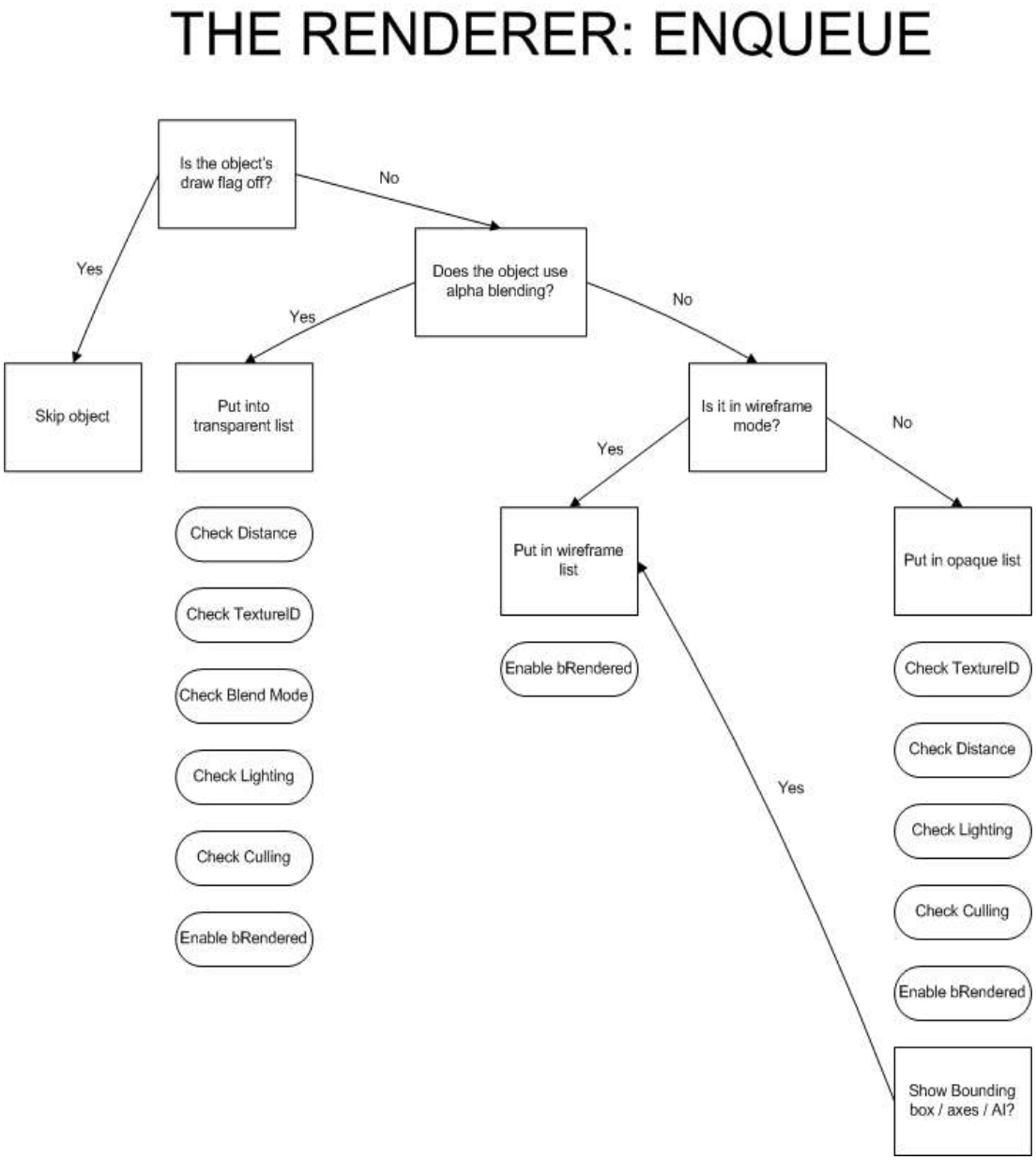
The renderer aims speeds up rendering by minimizing context changes. It accomplishes this by maintaining priority queues of objects sorted by their rendering context. As the renderer receives a request to draw an object, it computes the priority of the object and places it in the appropriate order in the appropriate queue. When all objects that the renderer should draw at the current frame are placed in a priority queue, the renderer will write objects to the backbuffer in descending priority order and present it to the screen.

Renderer

Inventory of Key Functionality

Initialize	HDC hrc, HGLRC hrc INT iWindowWidth, INT iWindowHeight, INT iPlayer	Starts the Renderer, passing in all the information needed in order to set up the window for rendering.
Shutdown	None	Deallocate and shut down the renderer
^ QueueInsert	RenderInfo &Rinfo RenderInfo *pList	Inserts RInfo into the appropriate place in the list whose first element is pList.
^ DrawList	RenderInfo *pFirstNode	Draws everything in list starting at pFirstNode.
^ CleanList	RenderInfo *pFirstNode	Remove everything that hasn't been drawn in a few frames from the queue.
^ Clear	None	Clears the screen.
^ CameraTransform	None	Applies the camera's transform to OpenGL.
^ MatchContext	RenderInfo *pContext	Set the renderer's RenderInfo context to match the one passed in.
^ Present	None	Swap buffers; present buffer to the screen
Enqueue	Base *pBase	Add this object's rendering info to the queue.
SetCamera	Camera *pNewCamera	Set the current camera to render from.
Render	None	Encapsulates the every-frame process of clearing the screen, drawing what should be drawn, presenting, and clearing the queues.
^ = Private function.		

Renderer


THE RENDERER: ENQUEUE
Is the object's draw flag off?
No
Yes
Does the object use alpha blending?
Yes
No
Skip object
Put into transparent list
Is it in wireframe mode?
Yes
No
Check Distance
Check TextureID
Check Blend Mode
Check Lighting
Check Culling
Enable bRendered
Put in wireframe list
Enable bRendered
Put in opaque list
Check TextureID
Check Distance
Check Lighting
Check Culling
Enable bRendered
Yes
Show Bounding box / axes / AI?

RenderInfo

Uses	Used By
Model	Base-derived objects that need to be drawn Heads-Up Display Front-End Menu World Renderer

Key Features

- Provides a data structure for all the information the renderer needs about an object in order to sort and draw it correctly

Key Member Variables

Model *pModel	The model in its already-animated state
Base *pOwner	Point to the object whose context this is
RenderInfo *pPrev, pNext	For linked-list functionality
INT iBlendSrc, iBlendDest	Parameters for glBlendFunc()
SHORT isTexture	The texture ID of this object
BOOL bTexture	True if the object is textured
BOOL bLight	True if the object is lit
BOOL bAlpha	True if the object is transparent
BOOL bRendered	True if we should draw it at all
BOOL bRenderedThisFrame	Provides the renderer with a way to mark drawn objects
BOOL bDepthTest, bDepthWrite	True if we should test against the depth buffer or write to it
BOOL bWireFrame **BOOL bDrawBoundingShape** **BOOL bDrawAxes** **BOOL bDrawAI** **BOOL bDrawPosition**	Extra things that can be turned on to draw more information about an object during debugging.

Inventory of Key Functionality

GetPriority	None	Calculates an integer priority based on the structure's context

Game Class

Uses	Used By
Objects Input Manager Sound Manager	Objects (Level Editor)

Key Features

- Handles general game logic
- Communicates between the managers and objects
- Implements the level editor / debug mode
- Implements the HUD / menu system

Key Member Variables (all are Static)

Enum eGameStates	A list of all available game states
Int iState	The current game state
Float fTimeInState	The time, in seconds, remaining in the current game state, just in case you want to implement time-based transitions.
Int iGameMode	Single player or two player split-screen
Int iLevel	The number of the level that the player is currently playing
World* world	The level itself
Base* ranger, falcon	The entities that player 1 and player 2 control.
Living* enemies	A linked list of enemies
Base* powerups	A linked list of powerups

Purpose

The game class contains the main game loop, and is the central module for Guardians of Neverwood. Its main purpose is to communicate between the engine's modules. The game class receives data from the input manager, objects, and world. It then uses this data to update its objects, change its game state, and play sounds when appropriate.

Game Class

Inventory of Key Functionality (all are Static)

HandleInput	None	Reads user input from the keyboard, mouse, and joystick, and updates the players and game state accordingly.
Update	None	Updates all objects
Initialize	None	Sets up all variables and parameters for the game. Calls all its members' Initializes.
Shutdown	None	Frees its own memory and any members' Shutdowns.
GameLoop	None	Applies physics, collision detection / response, and AI
Render	None	Uses the world to clip geometry. Draws all entities to the screen. Draws the HUD.

Game Loop

The main game loop is a single function that accomplishes a multitude of tasks in the following order:

- **Handle User Input**

At the beginning of the game loop, the game class uses the input manager to poll the keyboard, mouse, and controller(s) if applicable. Through the input manager's action mapping the game communicates actions to player 1 and, if in coop mode, to player 2. See the "Action Map" section for details.

- **Apply AI**

The game class applies AI only to classes that derive from Character.h, since they are the only ones that have AI functionality. It does so by calling the living object's aptly named ApplyAI() function, which applies any AI related computation and state changes that have not been covered by Update or OnHit.

- **Update Objects**

The game class iterates through all lists of all entities (any class that derives from Base) and calls their Update() function. In the very least, an entity's Update function guarantees that all the timing variables that the entity uses are modified appropriately. Where appropriate, an entity may also implement time-based behavior and state transitions. See the "Base Class" section for details.

Game Class

Game Loop (continued)

- **Apply Physics**

This is incorporated as part of the Update() function, and gets applied after an entity makes its time-based transitions. Physics include updating the object's velocity, updating its position, and applying ground clamping. The base class uses flags to determine whether to apply movement, gravity, or ground clamping.

- **Collision Detection / Response**

For each entity, the game class checks collision against all other appropriate entities and the world. The algorithmic details for this are implemented in BoundingShape.h. The game class is responsible for choosing the appropriate collision response based on the types of the colliding objects. Objects that collide with the world get clamped, entities colliding with each other respond by pushing each other slightly, and entities get knocked back if a weapon hits them. An entity's OnHit() function handles all the successive status and state changes. See collision detection section for details.

- **Render**

This is the phase where all the geometry gets processed for output to the screen, and is easily the most computationally expensive. Rendering proceeds in two phases: first, the Game class uses the World class to filter away invisible geometry. It then uses the renderer to draw the geometry. See the "World" and "Renderer" sections for details.

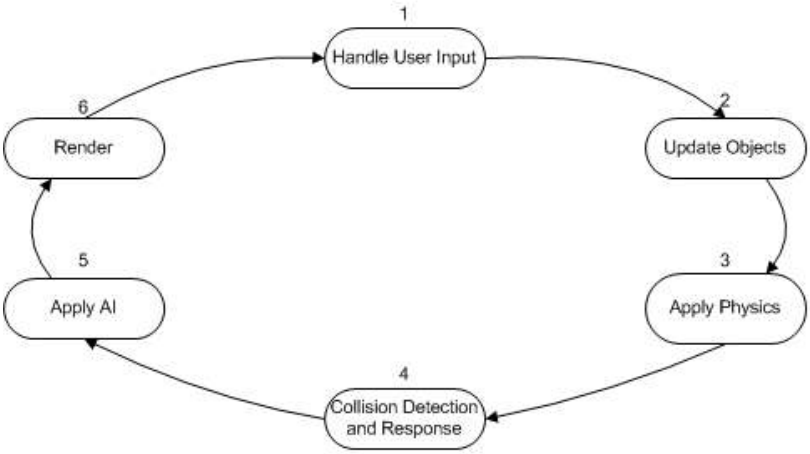
Notes

The main game loop also incorporates the level editor. See the "Level Editor" section for details, and for pseudo-code on the main game loop.

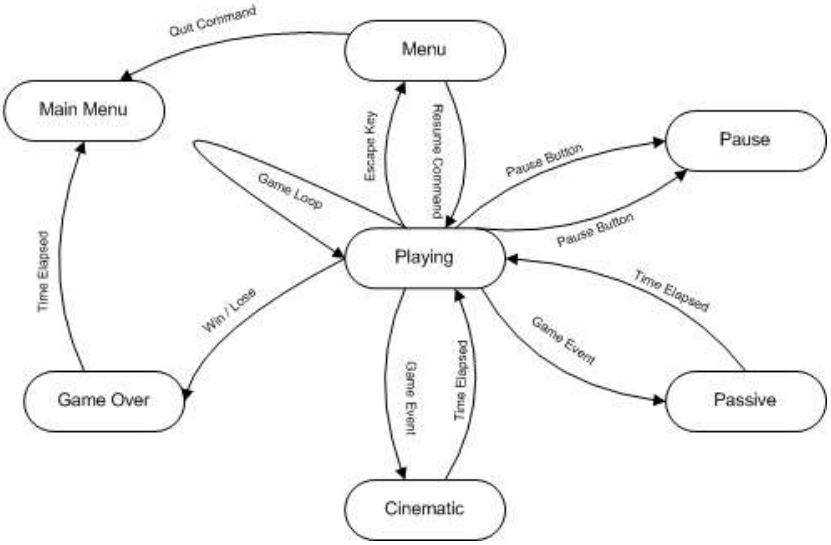
The main game loop uses a high-precision timer to keep track of the time elapsed since the last iteration of the game loop. To facilitate debugging, the game loop will clamp the elapsed time to a maximum of 0.1 seconds. This is because an excessive amount of time between frames may lead to bugs such as objects moving inside of each other.

Game Class

MAIN GAME LOOP



GAME STATES



The World Class

Uses	Used By
Base Camera	Renderer Objects

Key Member Variables

RenderInfo* RContext	The world's rendering context
Int iPartitionSize	The width (and depth and height) of the grid into which the world gets partitioned
RenderInfo* pWorldPartitions	A dynamic array of geometry storing one chunk of geometry for each world partition. The array represents a 3D grid indexed by the column, then row, then plane.
Int iCols, iRows, iPlanes	The width, height, and depth of the world, measured in number of grid cubes.
AINode* meshNodes	A dynamic array of AI nodes that provide a basic means for moving around in the world. The array represents a 3D grid indexed by the column, then row, then plane.
Int iMeshGranularity	An integer representing how far apart the cubes of the AI node grid are.

Inventory of Key Functionality

Partition	INT iPartitionSize	Splits up the world geometry into evenly sized cubes.
CreateMeshNodes	INT iGranularity	Creates a grid of AI nodes and edges for the entire level.
GetGeometry	INT iRow, iCol, iPlane	Returns the world geometry (RenderInfo*) contained in the cube with the given 3D indices
GetIndex	Base* pObject	Based on the position of the object, GetIndex finds the object's 1D index into the array of world partitions. Essentially, this function finds the section of the world that the object is in.
Load	CHAR *strFile	Loads geometry from a Maya file using the model loader, or loads an existing level in binary format.
Save	CHAR *strFile	Saves the current level (geometry, partitions, mesh nodes, and entities) to a .lev binary file

The World Class

Purpose

The world is the largest object in the game. It contains all the static geometry of the current level, the current level's node mesh, the partitioned data of the world, and possibly all the players, enemies, powerups, etc. Deriving the world from base lets us inherit functionality for storing and drawing geometry. Also, having the ability to move the world and alter its rendering context may lead to interesting special effects.

Algorithms

Since the functions in the world class are very involved, the following pseudo-code is supplied as a guideline:

Partition (size)

- Set the partition size and array dimensions
- Calculate the size of the dynamic array of geometry and allocate it
- Initialize the world data in each partition: set its data equal to the entire world's, and initialize the indexed arrays to 64 elements
- Calculate the center point of each triangle of the world's geometry
- Add each triangle's indexed data to the partition containing the center point
- After all the data has been stored, reallocate the dynamic arrays for an exact fit

CreateMeshNodes (granularity)

- Set the granularity and array dimensions
- Calculate the size of the dynamic array of nodes and allocate it
- Initialize all nodes to aerial nodes with no children
- For each node, create edges to neighbors only if the edge does not intersect any world geometry. Take a nap while this is being computed!
- For each node, mark it as a ground node if none of its children are lower than it

Physics

Uses	Used By
Vector / matrix math	Objects Game

Purpose

Guardians of Neverwood is not a physics-intensive game. As such, game physics are not an explicit module of Guardians of Neverwood. Instead, the game's physics are implemented within the functionality of the base class, and are implemented as part of the Update() function. Guardians of Neverwood implements the following physical concepts:

- **Movement**

The movement model of Guardians of Neverwood is very simple: each object maintains a velocity to denote its speed and direction of movement. The velocity is clamped to the entity's maximum speed every frame. To prevent choppy/unrealistic movement, especially by the AI, each entity has a maximum turning rate measured in degrees per second. Most likely, entities will have an acceleration rate as well to prevent choppy movement. The base class knows whether it is an air or ground entity, and automatically clamps rotations to the ground plane for ground entities.

- **Gravity**

Gravity runs at a standard 10 m/s^2. Since Guardians of Neverwood has static geometry and flying enemies, it is important that not all objects be affected by gravity. For this purpose, the base class contains a flag denoting whether gravity affects it.

- **Ground Clamping**

The base class ensures that ground units follow the surface of the world. Since the terrain is simple and there are no vehicles, it will suffice to use a single line emanating from approximately the model's knees to just below the feet. The point to clamp to will be the intersection of that line with the world.

Physics

- **Impact**

As an essential part of the combat system, entities get knocked back significantly from being hit by a weapon. Obviously, the amount of knockback depends greatly on the entity being hit, the weapon that hit it, and the attack that hit it. Realistic physics would be overkill and possibly hinder fun, so the idea is to simply base the reaction on entity mass, weapon mass, and attack damage. The following is a recommended function for knockback, although it is very difficult to predict what a good function is without experimenting:

Knockback = (weapon.mass / entity.mass) * attack damage

Collision Detection & Response

Uses	Used By
BoundingShape CollisionUtils Line Plane Triangle Quad	Objects Game

Purpose

Collision detection is handled by various different classes. On the highest level, each of the game's entities has a function to determine whether it has collided with another entity. On a slightly lower level, each entity has collision boundaries, which may be one of several kinds of shapes: spheres, cylinders, axis aligned bounding boxes, and frustums. Each of these shapes is a class with functionality similar to the base class. On the lowest level, the bounding boxes use general purpose classes representing lines, planes, quads, and triangles.

Bounding Shapes

Each object has at least one bounding shape used for collision detection. A bounding shape is abstract class that may be either a sphere, cylinder, axis aligned bounding box, or view frustum. Like the base class, bounding shapes have a position, an orientation, and various different functions for moving around. That way, bounding shapes may move together with the objects they contain.

- **Spheres**

Spheres are used as the default bounding box. They are quick and accurate, especially with respect to collision with cylinders. They also provide the most accurate means of collision response

- **Cylinders**

Cylinders are used as collision boxes for weapons, since weapons may have arbitrary orientation and do not fit the shape of a sphere well. Cylinders may also be used to represent the bounding box of an entity if a sphere will not suffice. Collision detection is much more expensive for cylinders, however.

Collision Detection & Response

Bounding Shapes

- **Axis Aligned Bounding Boxes**

Axis aligned bounding boxes are used to represent segments of the world. They are not used on the game's entities, as determining a point of intersection with other shapes is very inaccurate.

- **Frustums**

Frustums are used to determine areas of visibility. They are used mainly by the camera. Enemy AI also uses frustums to determine which objects are visible to it.

Key Member Variables

Matrix mGlobal	Maintains the global position and orientation of the collision boundary

Inventory of Key Functionality

CollidesWith	BoundingShape* other, Vector* intersection=0	Checks whether two bounding shapes collide. If they do, the function optionally outputs their point of intersection, usually at a slight additional CPU cost

General Classes

Collision detection routines all use the same library of 3D math algorithms. For the sake of reusability, these general purpose algorithms are encapsulated in their own classes:

- **Line**

The line class represents a simple line segment. It supports 3D half space tests, distance to a point, distance to another line, and computing the closest point to another line.

- **Plane**

The plane class represents an infinite 2D plane in 3D space. It supports half space tests, distance tests, and intersection with line segments. Even if a plane does not intersect a line segment, the plane to line segment test returns the intersection of the plane with the infinite line containing the line segment

Collision Detection & Response

General Classes (continued)

- **Triangle**

The triangle tests collision against other triangles and lines by using plane to line tests and 3D half space tests.

- **Quad**

Similar to the triangle, except that it has four vertices and edges. Quads must be convex and planar.

Collision Response

Guardians of Neverwood implements three different types of collision response:

- **Collision Clamping**

Collision clamping an object simply puts it just outside the collision box that it collided with. Collision clamping is used for entity-to-world collisions.

- **Collision Knockback**

Collision knockback is very similar to elastic collisions. An object that uses collision knockback pushes the object it collided with back. The magnitude of the knockback depends on the two object's relative masses. The direction of the knockback is the same as the colliding object. Collision knockback is used for weapon-to-entity and projectile-to-entity collisions.

- **Collision Pushback**

Using collision pushback on an object is almost the same as collision clamping, except that the object it collided with gets pushed back slightly. This is used for entity-to-entity collision. Its purpose is to prevent entities from blocking each other's paths for an extended period of time.

Artificial Intelligence

Uses	Used By
Vector	Objects

Purpose

The AI of Guardians of Neverwood is not implemented directly as a module. Rather, artificial intelligence is stored in several different modules incorporated into the world and entities: mesh nodes, paths, patrol points, and a finite state machine inside the living class. Ultimately, though, all the AI lies in an entity's ApplyAI() function.

Mesh Nodes

Mesh nodes pervade the entire world in a grid-like manner. Their density is one mesh node for every $8m^3$ (8 mesh nodes per world partition). Mesh nodes are precomputed and never altered. Each mesh node is connected to some of its 8 neighboring nodes. Two mesh nodes are connected only if the line joining them does not intersect any world geometry. In essence, mesh nodes provide a means for objects to move around the world without bumping into static geometry. See the 'world' section for additional details.

Paths & Pathfinding

A path is a sequence of adjacent mesh nodes, and is used to represent how to reach a certain destination from a certain point. Each AI-controlled entity maintains its own path that tells it how to reach whatever destination it might be trying to reach. For efficiency, paths are computed using best-first search. Since the worlds in Guardians of Neverwood are generally sparse, anything more than best-first search seems like overkill. Also, since there is no need for pathfinding over long distances, paths are recomputed every frame as part of the object's ApplyAI() function. Pathfinding always gets computed on a relatively large precomputed node mesh that pervades the entire world. If the entity is close enough to the target, it will create its own miniature node mesh to apply pathfinding on the fly. This miniature node mesh is necessary to avoid situations such as an enemy trying to run through a tree to get to the player on the other side of it.

Artificial Intelligence

Patrol Points

Enemies that do not perceive any danger generally pace around between a predefined set of points called patrol points. Patrol points are stored as an array of nodes within the enemy. The amount and location of patrol points is determined within the level editor. Care must be taken when placing patrol points into the world: the AI assumes that no line connecting two nodes intersects any world geometry. Also, there is a variety of different ways in which an enemy may use the patrol points:

- ***Pacing***

The enemy traverses the list of patrol points from beginning to end. After reaching the end, it traverses the list in reverse order, etc. Essentially, this simulates pacing back and forth along a certain path.

- **Cycling**

The enemy traverses the list of patrol points in cyclic order, going from the last element in the list directly back to the first

- **Roaming**

Whenever the enemy reaches a patrol point, it picks its next patrol point at random.

Finite State Machine

Each entity has an implicit finite state machine defining its behavior. Although the finite state machine of each entity can and should vary from other entities, a basic model for an enemy FSM is given in the enemy behavior diagram. Note that many of the transitions in the FSM have a probability associated with them. The purpose of this is to prevent enemies from becoming predictable and easily manipulated. All enemies support the following states, which are defined in Character.h:

- **Patrol**

The enemy traverses its patrol points in one of the ways described above.

- **Confused**

The enemy walks around in random directions. This is used for two purposes. One is because some attacks have the ability to physically disorient opponents. The other is as an emergency routine to fall back to if the AI gets stuck.

Artificial Intelligence

Finite State Machine

- **Frozen**

The enemy cannot move while in this state. Happens after they get hit by Tempest's Freeze Arrow.

- **Alerted**

This is a temporary state in which the enemy prepares to follow a target. The amount of time spent in this state is random, but usually short. This is to avoid the artificial looking behavior of everyone reacting to an event instantaneously. An important aspect of alertness is that enemies alert each other if they discover a threat. If an enemy gets attacked, it will automatically notify any enemy within 15 meters of the attack. Also, most enemies may choose to call surrounding enemies to their aid at any time.

- **Approach**

The enemy has chosen a target and is **running** after it. This involves computing a path to the target at each iteration. Keep in mind that the target is not necessarily something the enemy is trying to attack.

- **Change Target**

This is not an explicit state, but has been added to the diagram for clarity. While approaching a target, some events may cause an enemy to switch targets.

- **Attack**

If the enemy is in the attack state, it is taking a swing at its target. Since enemies are generally big, sluggish orcs, they may not move or change states in mid-attack.

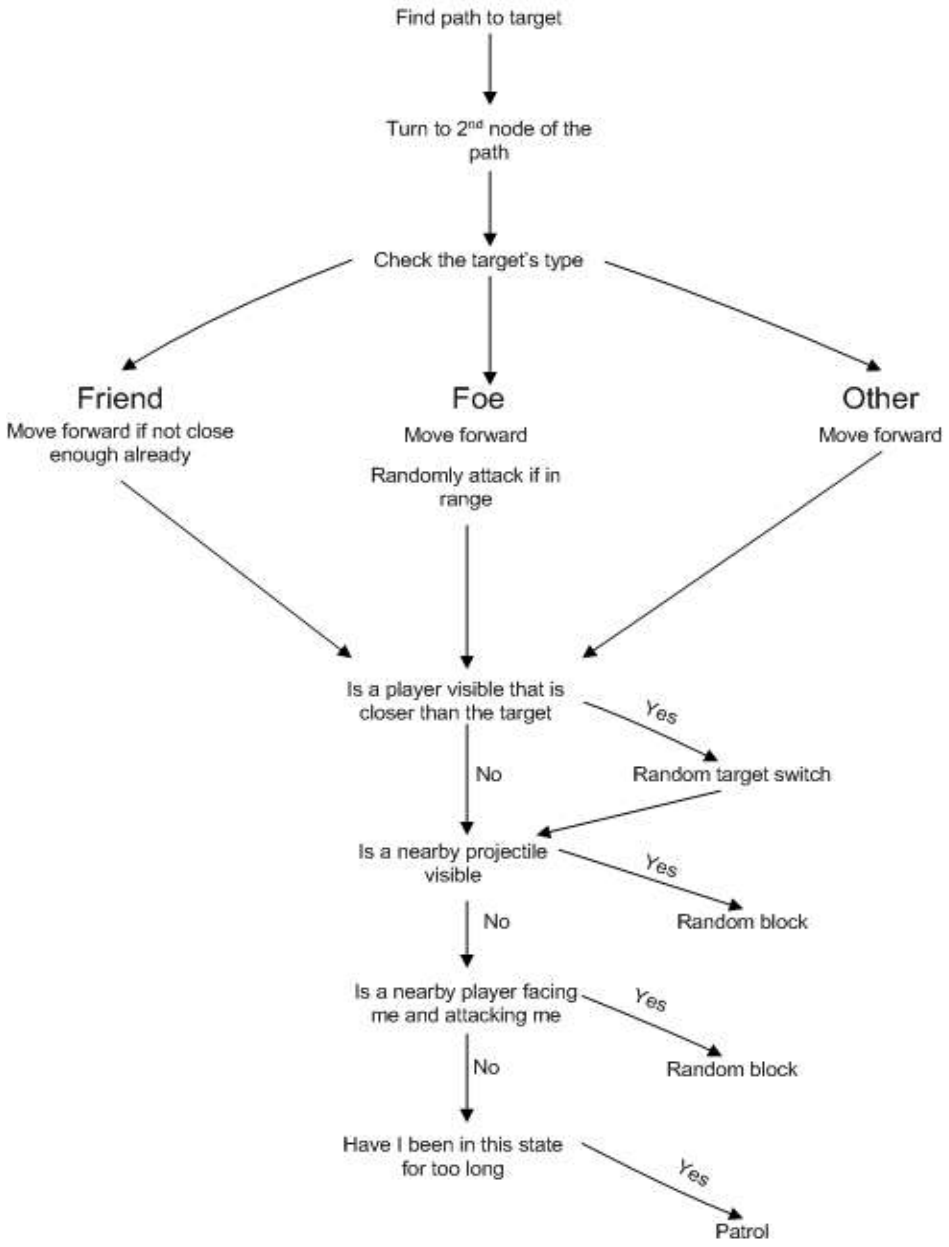
- **Block**

Enemies try to block incoming attacks, but only if they are coming from the front. The delay associated with the block as well as its duration is random. As a general rule, if an enemy blocks a hit of a flurry of attacks, it continues blocking all consecutive hits. This is to encourage the player to use variety in his attack patterns.

Artificial Intelligence

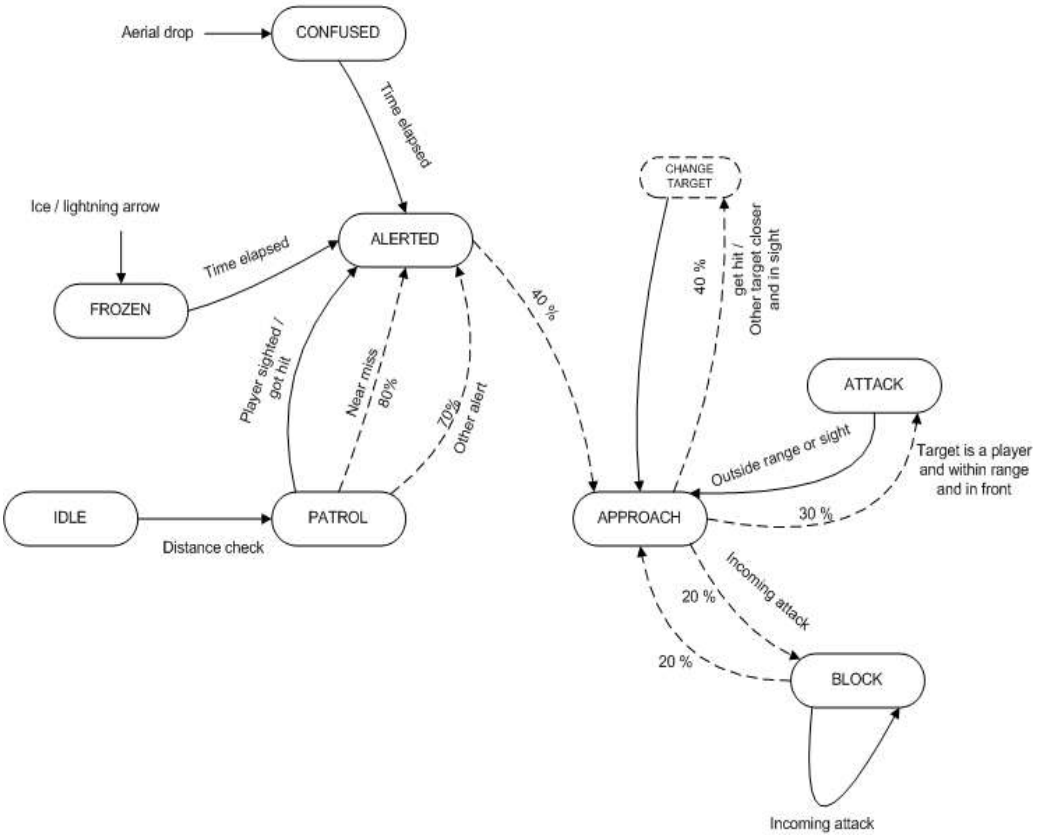
THE APPROACH STATE

The approach state has the most complicated dynamics, since its behavior depends on the object being approached. The following flowchart is pseudo-code for an enemy's behavior within this state



Artificial Intelligence

ENEMY BEHAVIOR



Artificial Intelligence

Applying the AI

Enemies apply their AI in the aptly named ApplyAI() function. The most important part about updating the AI is that **the AI only updates every 350 milliseconds**. This is for several reasons. First of all, it saves a lot of computation time, allowing the AI to be more sophisticated than would be possible on a per-frame basis (or speeding up the game, alternatively). Also, it helps to have a fixed amount of time between updates so that the AI's behavior does not change based on the game's frame rate. Note that this is necessary due to the finite state machine's probability-based transitions. Most importantly, it makes the AI more human by introducing slightly delayed response times.

Level Editor

Uses	Used By
World Renderer	Game

Key Features:

- Uses in-game renderer, characters, collision detection, etc.
- Populate an existing level with entities
- View and modify AI nodes and edges
- View Object, collision box, and terrain data

Key Member Variables

INT iGameMode	The level editor adds 3 extra game modes to the game class: object editor mode, AI editor mode, and debug mode.
INT iObjectEditMode	If the user is in object editor mode, this determines which kind of entity is currently selected
INT iAIEditMode	If the user is in AI editor mode, this determines what kind of AI info the user is trying to modify
INT iDebugMode	If the user is in debug mode, this determines what kind of debug info the user is trying to view
Node* currentNode	If the user is in AI editor mode and trying to add edges, the user needs to highlight two nodes to create an edge. This variable keeps track of the first node highlighted.

Inventory of Key Functionality

OnAction	None	Determines what happens if the user pressed the action button based on the game mode and the position of the cursor.
OnQuery	None	Determines what happens if the user pressed the query button
OnDelete	None	Determines what happens if the user pressed the delete button

Level Editor

Purpose

The level editor is intended to support a limited set of specific features, allow the user to view and modify level data quickly, and do so with a minimum amount of additional effort. To achieve this, the level editor is incorporated directly into the game. The level editor has three major purposes. Its first goal is to populate worlds with objects and enemies. A secondary purpose is to modify data that is usually invisible, like AI related information or object properties. The third purpose is to facilitate debugging. Each of these 3 purposes is implemented as a separate game mode in the game's main loop.

Editor Modes

The level editor has two kinds of modes: game modes and the editor modes. The game modes determine whether to edit entities, edit AI, or to debug, and are implemented as game modes in the main game loop. The editor modes determine specifically what kind of object to edit. See the diagram on the following page for details. The modes are as follows:

- **Entity Editor Mode**

The entity editor mode is used to populate the level with enemies and other entities, and is the default mode of the level editor. It works as follows: the user chooses a type of entity to place by scrolling through the game's entities using the 'J' and 'L' keys. An instance of the currently selected entity replaces the player. Upon entering the entity editor mode, the level editor uses the renderer to restore a default rendering context (enable lighting, texturing, depth test, etc).

- **AI Editor Mode**

The AI editor mode is used to modify AI vertices and edges. By choosing the AI editor mode, the cursor changes to a wireframe sphere of radius 0.1 (this is implemented by creating an empty base class with a sphere bounding box). The level editor uses the World class to activate AI data. The color of the sphere reflects the editing mode as follows:

	Red	Green	Blue	Color
Ground Node	**1**	**1**	**0**	**Yellow**
Ground Edge	**0.7**	**0.7**	**0**	**Dark Yellow**
Air Node	**0.5**	**0**	**1**	**Light Blue**
Air Edge	**0**	**0**	**1**	**Blue**
Patrol Point	**1**	**0**	**0**	**Red**

Level Editor

Editor Modes (continued)

- **Debug Mode**

The debug mode is used to view data. By entering the debug mode, additional information gets output to the screen. The top left of the screen displays the player's coordinates. If the player collides with something, the word "Collision" appears on the screen, as well as the coordinates of the object with which the player collided. As with the other modes, there are many forms of the debug mode. The Object mode shows only visible objects. The Box mode allows the user to view and modify collision boundaries. The terrain mode allows the user to view height mapping data and world partitions. If the need arises to view any specific variables while in mid-game, this mode provides functionality to do so.

Interface

Overview

Essentially, the interface consists simply of a cursor and three commands: ***Action*** (Space bar), ***Query*** (Enter), and ***Delete*** (Delete key). Player 1 acts as the cursor, checking collision against targets that it collides with. The look of the cursor changes with the game mode. The query button can be used to view and modify the data of the cursor or object the cursor collides with. The behavior of the action button depends on the game's and editor's states. The delete button removes the entity from the level entirely.

Reading Input

The level editor uses the same function as the main game to read and respond to user input. It then uses buffered keyboard input to check for keyboard commands that are unique to the level editor. Finally, it checks whether the mouse buttons are pressed down and calls OnAction and OnQuery appropriately.

Activating / deactivating the level editor

To enter the level editor, simply press the ***Break*** key at any time during the main game. Activating the level editor will deactivate all entities except for player 1. It will also set the game mode to GAME_DEBUG and output the message "Debug Mode" to the top left of the screen. To exit the level editor and return back to the game, press the ***Break*** key at any time during any of the level editor's modes. Doing so will reset the player to its default values (by making it a new ranger), and reactivate all entities.

Level Editor

Interface (continued)

Changing modes
At any time, you may use the 'I' and 'K' keys to cycle through the entity, AI, and debug modes. Similarly, you may cycle between the editor modes with the 'J' and 'L' keys.

Activating / deactivating entities
Obviously, it would be difficult to modify a level's entities and nodes if they kept moving. Thus, activating the level editor automatically deactivates all enemies (by setting their activity variable to 0). This is also useful for debugging, since it enables the user to view the game's data at key points in time. If the user wishes to run the actual game while in debug mode, he may manually toggle the enemies' activity by pressing CONTROL + A.

The Action Button
The purpose of the action button depends on the game mode and the cursor's position as follows:

- ***Entity editor mode***

If the cursor does not collide with any entities, the entity that the cursor represents gets placed into the world. If the cursor collides with an entity, the cursor gets replaced with that entity, allowing for an easy way to move existing entities. Ground units automatically get ground clamped upon placement.

- ***AI editor mode***

If the user is trying to modify nodes (ground node, air node, or patrol point), then the action button will create a new node at the location of the cursor if the cursor does not collide with other nodes. If the cursor collides with another node, that node replaces the cursor, allowing for a quick way to move existing nodes. If the user is trying to modify edges, the behavior is variable. To place an edge, the user must click a node using the action button to select it as the current node. The user then clicks on a different node to create an edge form the current node to that node. Notice that the current node persists, making it easier to add multiple edges to the same node.

- ***Debug mode***

In debug mode, the action button is equivalent to the query button described below.

Level Editor

Interface (continued)

The Query Button
The Query button is used to view and modify data by producing a dialog box of the selected object. Its exact behavior depends on the game mode as follows:

- ***Entity editor mode***

If the cursor collides with an entity, output that entity's data. Otherwise, output the data of the entity that the cursor represents.

- ***AI editor mode***

If the user is trying to modify nodes and the cursor collides with an AI node, output the data of that node. Otherwise output the data of the current node. The case for modifying edges is analogous.

- ***Debug mode***

In object mode, check for collision against entities and output their data if they collide with the player. In box mode, output the data pertaining to their collision box only. In terrain mode, optionally show relevant height mapping or segment data if needed. Optionally add any additional features on an as-needed basis.

The Delete Button
The behavior is analogous to the query button, except that objects get removed instead of output.

Saving changes
At any time in the Entity editor or AI editor mode, the user may press CTRL + ALT + F5 to save the current level to disk. Make sure to back up levels frequently, since an untimely CTRL+ALT+F5 could cause a lot of damage. The level editor uses the world's save function to save changes.

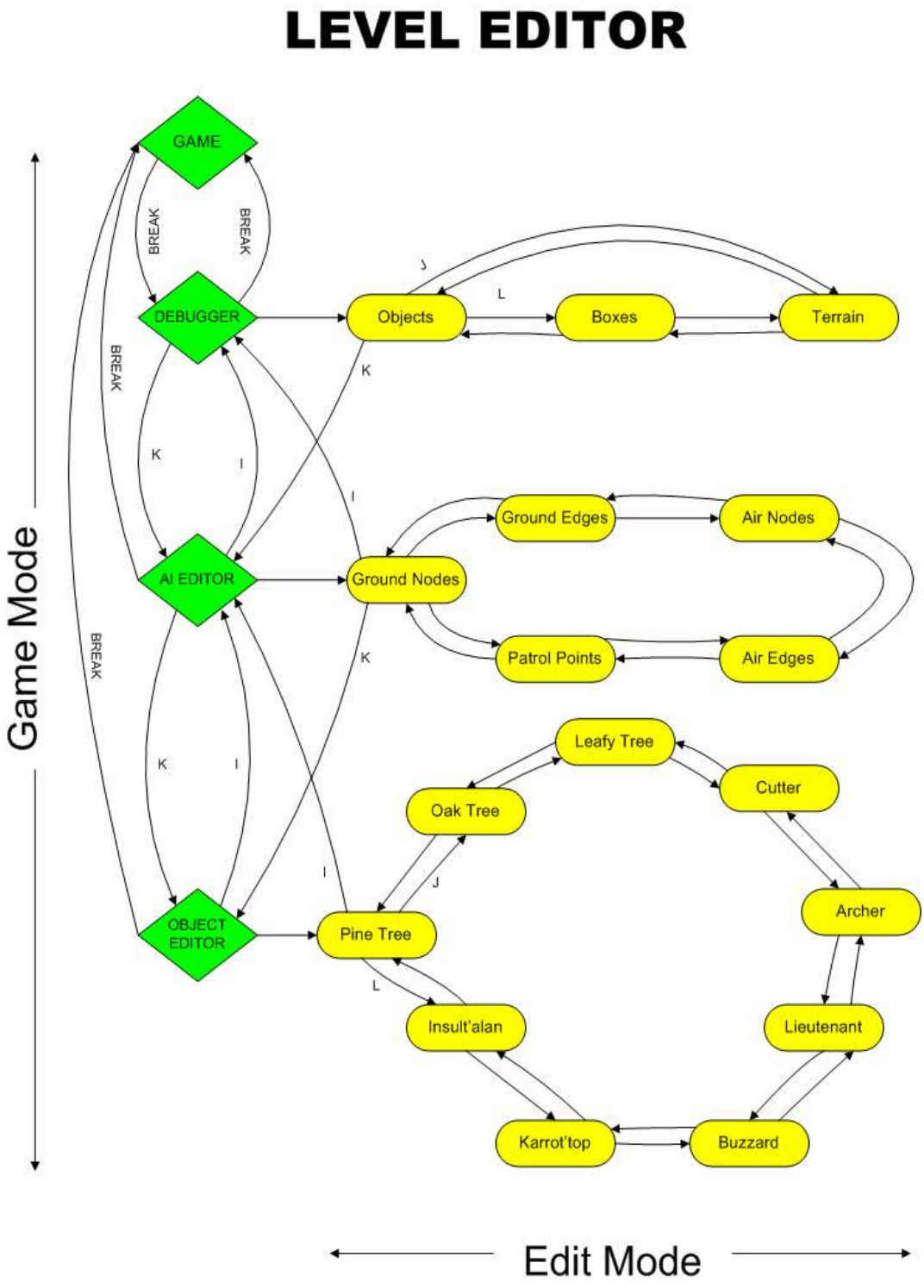
Level Editor

Implementation

The following pseudo-code demonstrates how the level editor is meant to be integrated into the main game loop:

```
Main game loop:
Handle user input;
if game mode is one of the editor's modes:
        check and respond to action, delete, and query buttons
        check and respond to I,J,K,L buttons
        check and respond to ctr+a
        check and respond to ctr+alt+f5
        (break button already handled in HandleUserInput)
Update
if in AI editor mode, apply the  AI editor's logic
if in entity editor mode, apply the entity editor's logic
otherwise apply regular game logic
Render
```

Level Editor



Heads-Up Display

Uses	Used By
Character and its children Renderer or OpenGL Emitter Vertex	Game class Level editor

Key Features

- Provides the user with information about the game and player states.
- Serves as a means to inform cooperating players how the other player is doing.

Key Member Variables

INT iPartnerTexture	A texture ID of a 2D representation of the other character in a 2-character tandem. In 1-player mode, this will be Aurexis.
INT iBGTexture	The background texture for the HUD.
INT iLifeTexture	Store the texture ID for the life bars.
INT iChargeTexture	Store the texture ID for the charge bars.
Vertex pVertices[4]	The four corners of our quad for the HUD. Since this isn't going to be lit, there is no need to tescellate the geometry.
Character *pChar	The character whose progress this HUD instance is tracking.

Notes

The sole purpose of the HUD is to present information about the character's status to the screen for the user. It is nothing more than a graphical representation, with fairly limited functionality.

Inventory of Key Functionality

Draw	None	Renders the HUD to the screen.
Initialize	Vector vLowerLeft, Vector vUpperRight,	Sets up the initial values for the HUD. Since there is no dynamic memory, no Shutdown is needed.

Front-End Menu

Uses	Used By
Renderer Model DirectInput ActionMap Game Emitter	None

Key Features

- Gives the user an engaging means to navigate the game's features.
- Allows access to unlockable features and special game modes.

Key Member Variables

Model *pModel	The model for the menu system.
INT iMenuState	Stores what section of the menu we are in.
Emitter *pEmitter	A particle emitter for a mouse trail effect.

Notes

The menu will have the feel of walking through a level, though it is entirely first-person. The geometry is very simple, mostly several large, moderately tescellated quads.

Inventory of Key Functionality

Draw	None	Renders the menu to the screen.
Update	None	Gets user input and makes changes accordingly. All game state changes (due to unlocked items, etc.) are passed directly to the Game class via its singleton interface. If this ever returns 'false', they've hit Quit, and it's time to run shutdown code and spew the user back out to Windows.
Initialize	None	Sets up the menu for use
Shutdown	None	Deallocates the menu and finalizes any changes made to the state of the overall system.