ARES

CS196-2 Final Project

Post-Mortem

John Shields
Andy Bragdon
Yee-Cheng Chin
Michael Shim

May 16, 2006

Design Concept

Today's hottest video games boast photo-realistic graphics, humanlike movement, facial expressions, and accurate physics models to boot. Many of these games also claim to have "intelligent AI"—meaning that the in-game agents emulate human decision processes. The reality is that although graphics, physics, and sound have made remarkable strides towards realism, AI remains glaringly stupid. A primary cause of this stupidity is that the all-powerful AI gives orders to every non-player agent in the game. When "bosses" and "underlings" exist in a game, the AI controls them on an equal contextual level. This *monolithic AI model* fails to consider real commands in military or social organizations are passed down through a chain of command. Ideally, a non-player agent's choices are not only dependent the *game state*, but also on the *goals of his commanding officer or social superior*. A second problem with the existing monolithic AI model is that of *global information*: when one enemy grunt discovers the player's location, then *all* other enemies are also instantly made aware of this information. Real social and military structures exhibit *information lag*, that is, that any information detected at a low level in the organization will take time to propagate upwards to the commanding ranks and then downwards to all concerned parties in the hierarchy.

ARES presents a solution to this problem: a *hierarchal AI model* where units receive orders and information through a chain of command, then use that information to make decisions for themselves and their inferiors. The *ARES AI System* will provide real time strategy games with an enhanced sense of realism and enable new levels of engagement for players. Our game *ARES* is not a "game" in the conventional sense of continual player involvement, but rather a showcase of the *ARES AI System*: the user constructs a scenario using the *ARES Studio* editor program, then runs the app and watches the action unfold as troop movements are coordinated in an AI hierarchy. As the game was conceived and implemented by a team of four people in one semester, we chose to focus our efforts on a never-been-done-before AI model instead of implementing gameplay elements found in other action and strategy games; this choice was inline with the CS196-2 course goals. Any number of popular games—from *Warcraft* to *Doom* to *The Sims*—could implement our AI System to enhance the player experience. Our ultimate goal is to hear the player exclaim: “I can't believe the A.I. just did that!”—a reaction we’ve had a few times ourselves when testing our engine.

There are many possibilities for AI improvement in games research, and it is appropriate to distinguish briefly between what *ARES* is and is not. The *ARES AI model* does not emulate a human video-game opponent but rather a real-world human army. For this reason, *ARES* has a primary application to Real-time Strategy games. Note also that it can be used both as an opponent AI and also a support AI; in the latter case, the player would be able to issue orders to groups of units which would be executed in a realistic military fashion. We do not seek to create an *invincible AI* which can beat a human player every time (e.g. IBM’s *Deep Blue* Chess AI), or a *scalable AI* which can be adjusted in difficulty to provide the player with the optimum sense of challenge. There many excellent opportunities for AI improvement, some of which were explored in other CS196-2 projects including Q-Boltz Millenium Wars and Kung Fu Tamagotchi. In *ARES*, we kept our goal focused on the hierarchical AI model as a way to create an added degree of in-game realism.

ARES *at a glance*

Team
Developers (full-time): 4
Budget: $30.00
Release Date: May 12, 2006
Development: < 12 weeks
Platform: Windows XP

Workstations (x4)
Intel Pentium D 3.0 Ghz w/ 2MB
2 GB DDR2 RAM
2x 250 GB Hard Drive
ATI X800 w/ 256 MB RAM
Windows XP Professional (32-bit)

Software
Visual Studio 2005 Professional
Autodesk 3ds max 8
Adobe Photoshop CS2
Tortoise SVN
Sony SoundForge
oFusion
Skype
Bugzilla
Windows XP Professional

Technologies
DirectX v9.0c
OGRE v1.2 “Dagon”
BASS Audio Library v2.2
CEGUI v0.4.1
.NET Framework 2.0
SandDock 1.0

Size
25,190 source code lines
20,206 game asset lines

ARES AI System

Description

The *ARES* AI System is structured around four military ranks as follows: 1 *General*, 5 *Colonels*, 10 *Captains*, and 130 *Grunts*. A standard game configuration can handle up to 300 units total on a 300x300 tile map; by comparison Blizzard Entertainment's *StarCraft* has a *maximum* map size of 256x256 tiles and roughly 125 total units in a game. The ARES AI operates in turns. Each turn, units perform basic actions such as moving, detecting enemy units, and firing. Additionally, *decision-making units* (Captains and above) *think* and give orders every couple of turns, with Captains *thinking* more frequently than either Colonels or Generals. Different ranks use different decision making algorithms as follows:

- The two *Generals* (one per side) are concerned with the whole map; they play "a game of chess" using a Minimax look-ahead search algorithm to make orders to move the Colonels. A General makes decisions based on the location of its troops and the known locations of the enemy troops. Much like a realistic general, however, ARES Generals take a broad view of the game state: they consider the map as a 10x10 grid and look at groups of units rather than individuals.
- *Colonels* receive an order from the General to move to a broad region on the map, then organize their subordinate Captains to best attack enemy units and capture the target area.
- The *Captains* focus on most zoomed-in extent of the game, coordinating the movements of Grunts to attack enemy forces. Unlike Generals, Colonels and Captain make decisions using utility functions as follows:

    1. Game state factors such as the number of combatants on each side are considered. Then the AI walks through limited set of possible tactics (predetermined attack routes) looking for chances to "flank" the enemy from the side or rear. Using this model, a chance of winning is evaluated for each tactic.

2. The possible outcomes are then weighted by personality factors such as Aggressiveness which are unique to the unit.
3. If the Utility Function evaluates below a certain threshold, the unit will order its subordinates to retreat. Otherwise, the attack will proceed with the tactic of greatest utility. The Captain generates routes for his troops by mapping the attack route to a list of tiles using A* pathfinding

- *Grunts* implement only the most basic level of AI agent actions, including detecting enemy units, moving, and firing. Like the real military, our model gives grunts almost no opportunity for independent thinking; they are merely cogs in the machine.

Conclusions

It important to distinguish between the macro-scale Minimax AI of the Generals and the micro-scale "Utility function" decisions made by the Colonels and Captains; the Generals are really in a whole different ballpark. Generals thought ahead quite a bit and while it helped them make better decisions, it also didn't help to a certain degree. Since each unit was its own agent, it was very difficult to predict exactly what was going to happen. The Minimax algorithm had to be simplified so that it would run in the right time requirements which meant that we had to strip down the state to a significantly smaller subset of what the real game was. Therefore, the decisions that the General would make several turns in the future wouldn't affect it that significantly. In that light, we found that the Generals who made short term decisions (we found 3 turns to be pretty good) did better. We also found that when the Generals would attack initially with one brigade and then bring in the rest of the army would win most of the time. There was actually a case where one brigade was sent forward, the rest of the army sent backwards and then when the enemy attacked the advance brigade retreating and the rear brigades advanced and wiped out the entire enemy force with minimal damage. Bringing everybody up at the same time and attacking actually seemed to be a poor attack strategy. Also, defending seemed to be very useful. Defending brigades have actually been known to wipe out 2 – 3 enemy brigades by themselves on occasions. However, since the goal of the game is to defeat the other side, the general has been weighted to attack more often than defend.

As for the Captains and Colonels, the different tactics they employed would yield significantly different results. We experimented with various tactics using test-setups in our *ARES* Studio editor. We found that, in general, tactics that split the units into multiple squads randomly does not fare very well against a group of concentrated enemy units. One reason for this is that usually the split squads approach the enemy one-at-time. The group of concentrated units focuses its fire and can usually first destroy the first enemy squad, then the second. Note that this could change if grenades were put into the game, which could destroy a closely packed cluster of soldiers. When units move together in a large group, they all face the same direction and are then vulnerable to flanking. Our current build of the game gives a large attack bonus to flanking in part to make a disincentive for having a large group. In our current release, we find that the best tactic is one that splits the units into two flanking routes, especially if the squad's enemies are in a line formation. In a chaotic battlefield, however, the units are susceptible to attack before they have a change to fan-out in their flank patter. Therefore, whether the flank tactic is superior depends on how far and how safe it is for the units to fan out and form the flanking pattern before they see the enemies. Another possible tactic is to split the units into three squads: two squads will flank, and one will move forward in a straight line. This tactic, however, seems to spread the units thin too much, and the middle "frontal assault" squad usually fares poorly. In summary, a tactic's success rate is generally determined by spread out the units are and what directions the units will end up facing, but still superior tactics are no substitute for superior numbers in the majority of cases. We explore possible improvements to our AI model in a subsequent section of this document.

Quantitative Evaluation

Our project met and exceeded the **A+ Project Requirements** as specified in our proposal, reprinted here word-for-word:

AI

- Generals
    - MiniMax algorithm is used to issue orders and take actions
    - Evaluates state information based on a state evaluation function and chooses the move it will make based on the highest possible state evaluation among the examined states
    - State evaluation function should take into account number of living friendly units as well as number of killed enemy units, and it should use weights for unit types (e.g. a captain is worth more than a grunt, etc.)
- Colonels
    - Markov decision process is used to issue orders and take actions
    - MDP takes into account number of friendly and enemy units in its vicinity
- Captains
    - Markov decision process is used to issue orders and take actions
    - MDP takes into account number of friendly and enemy units in its vicinity

Graphics

- Two views: symbolic view and battle view. Symbolic uses symbols to show battle progress, battle view shows 3D isometric view of the battle complete with units

Game Logic

- Game logic system must function as an API for accessing elements within the game as well as game concepts, including firing and the accuracy model, time, orders
- Command/information lag:
    - Each unit will have their own perceived state that they will use to make decisions

Formations

- Formations govern the relative motions of units to produce an overall shape or formation of units
- Graphical formations editor
    - Uses XML files to store formation data

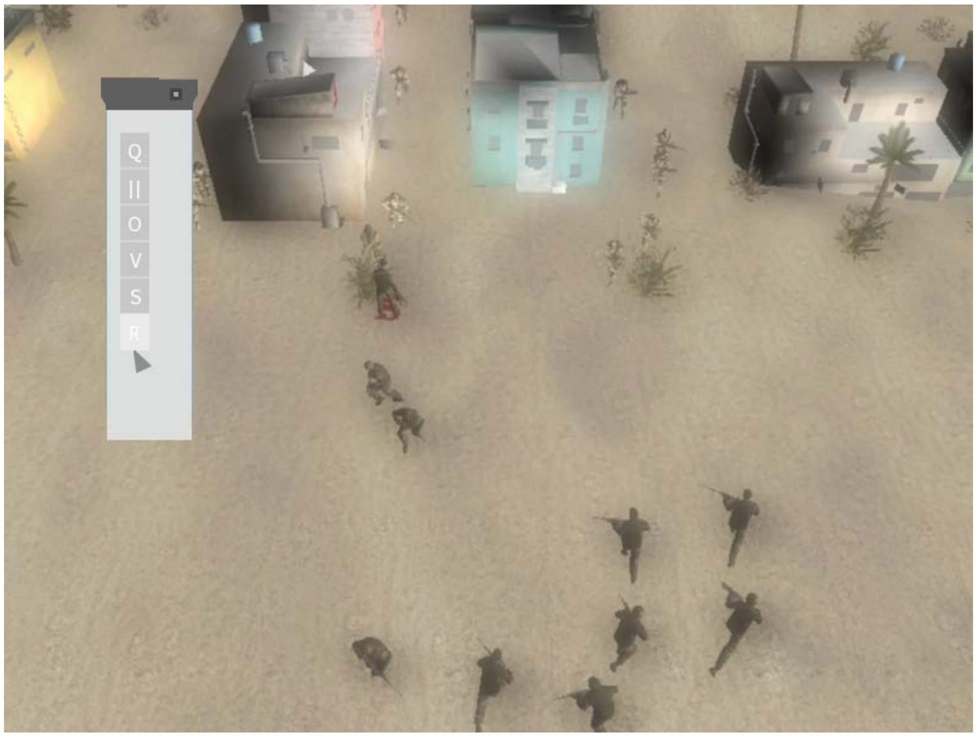
Interactivity

- Clicking on a unit with information tool will display the unit's current status in the UI
- GUI button-based UI:
    - Ability to pause the simulation
    - Ability to change views

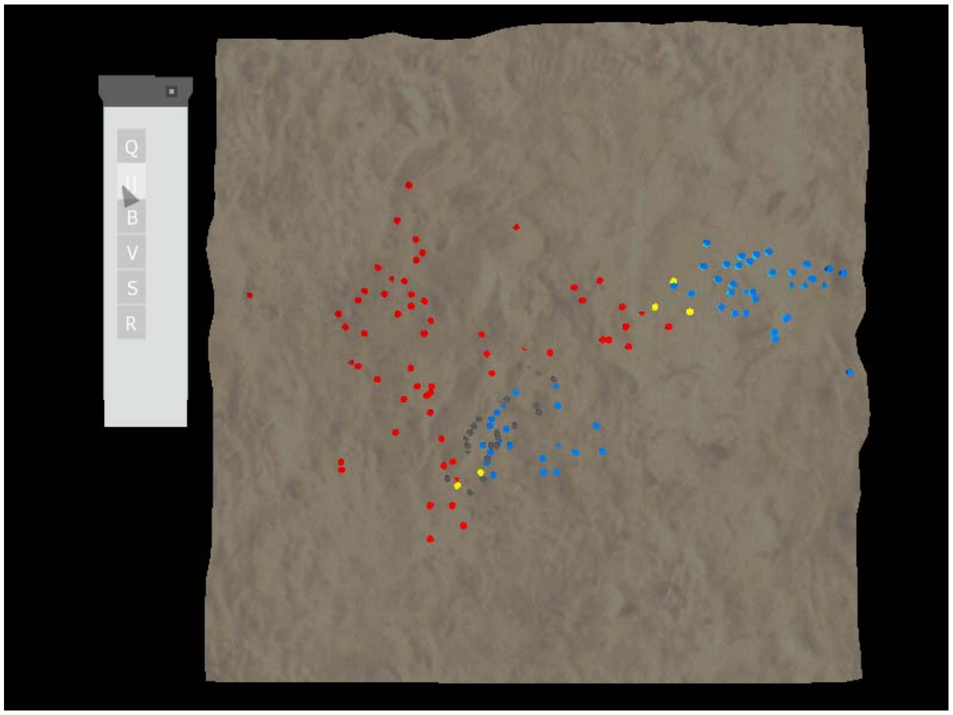
Weekly Presentations

- Weekly update presentations beginning March 17, 2006 except during Spring Break

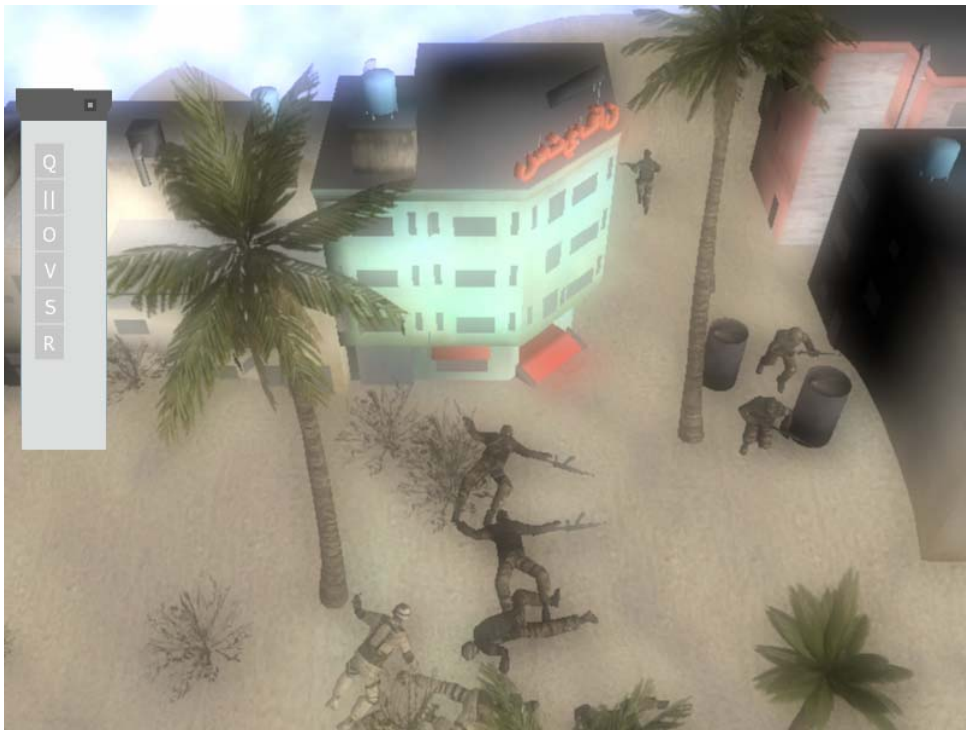
VISUALS



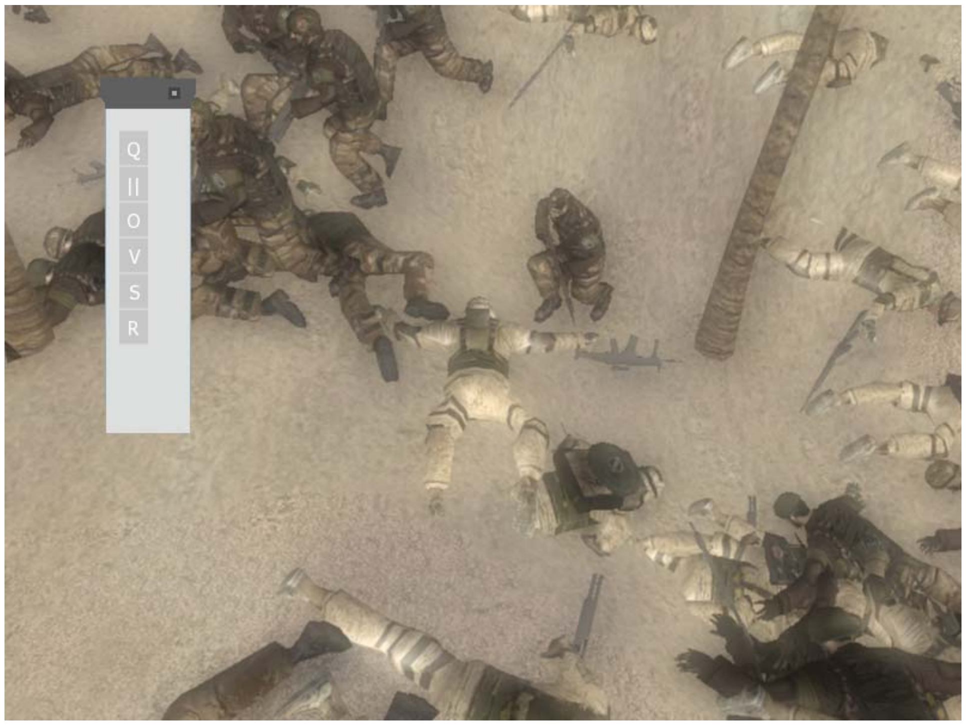
Troops advance and retreat in coordinated squads as directed by their chain of superior officers.



Symbolic View shows troop the movements in a high framerate context.

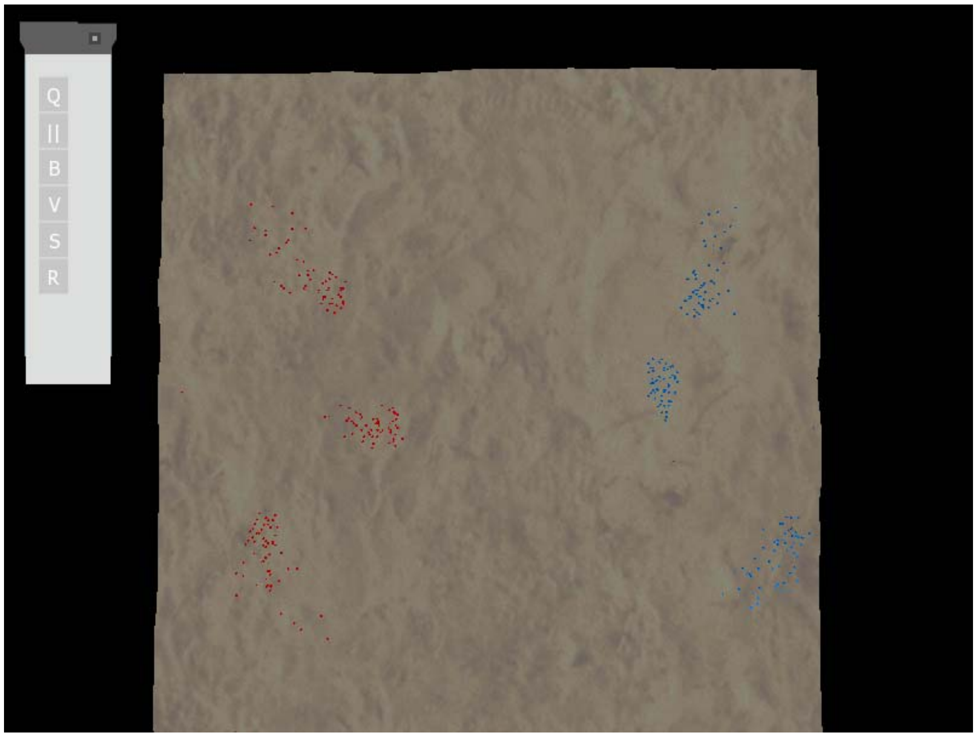


Soldiers use **A* Pathfinding** to maneuver around obstacles and buildings—look at those two squeeze between the barrels!

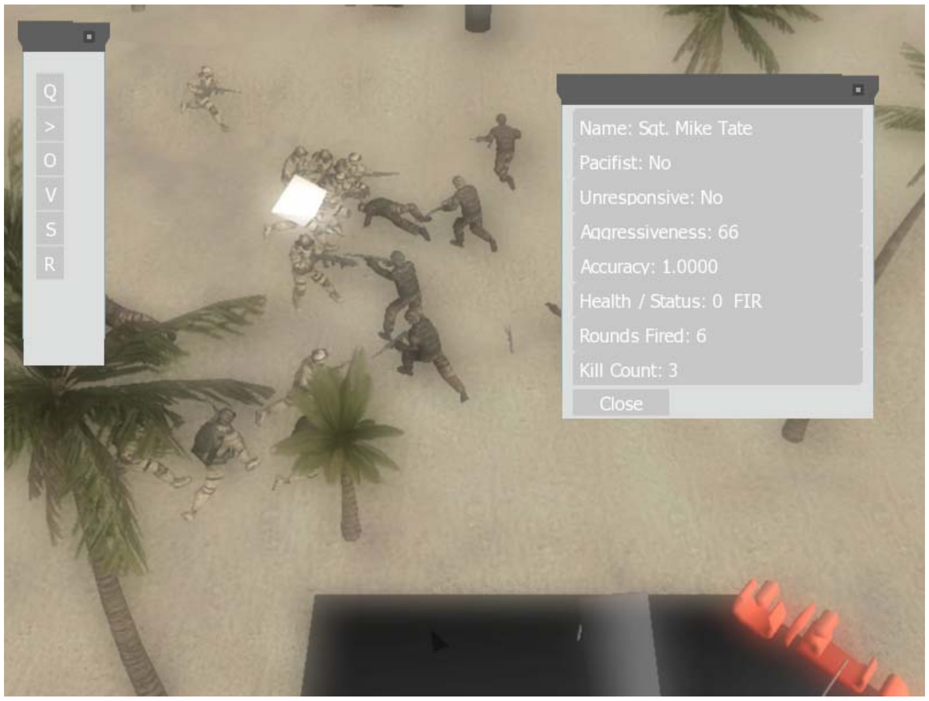


Zoom to full extent and watch the carnage up close in stunning 3D. Every soldier appears slightly different in terms of kit and weaponry.

Smoke and blood particle systems and sound effects immerse the player in a cinematic war environment

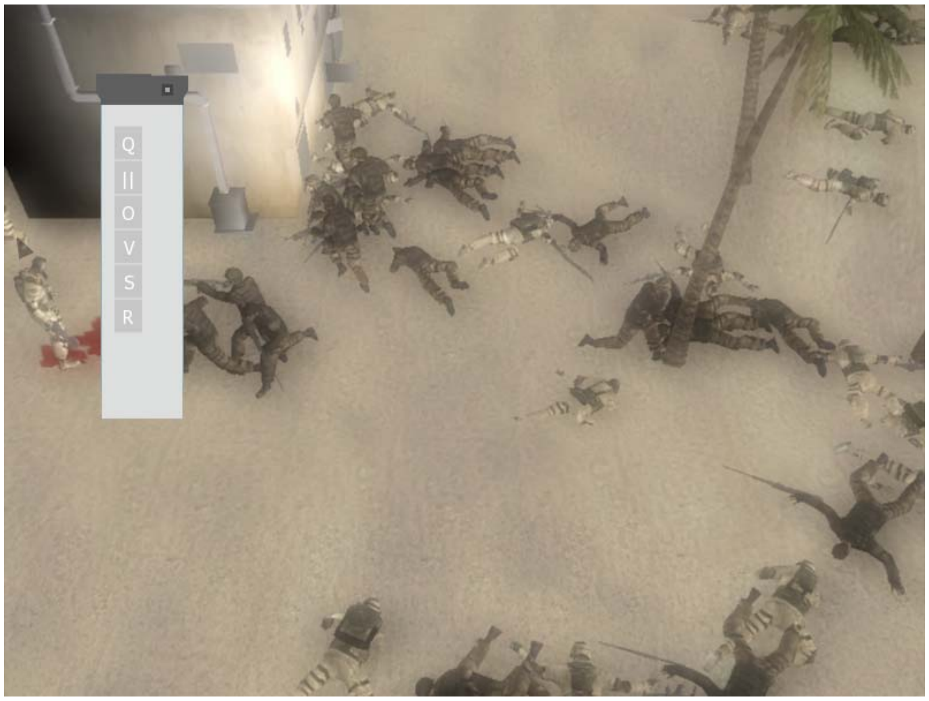


The Ares AI can be **massive**—can you see the 300 individual dots on the 300x300 tile map shown above? That beats out most contemporary Real-time Strategy games in terms of size!

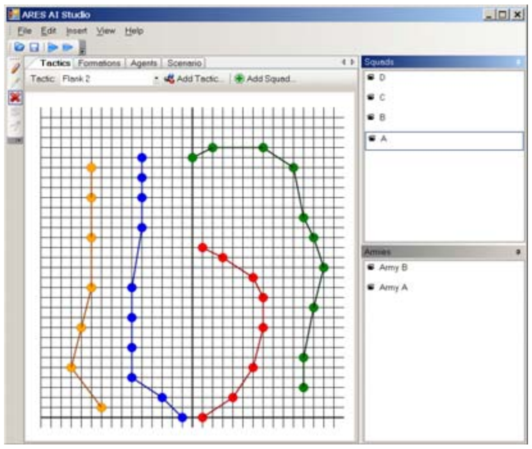


Name: Sgt. Mike Tate
Pacifist: No
Unresponsive: No
Aggressiveness: 66
Accuracy: 1.0000
Health / Status: 0 FIR
Rounds Fired: 6
Kill Count: 3

Click on a unit to view its attributes and battle statistics.

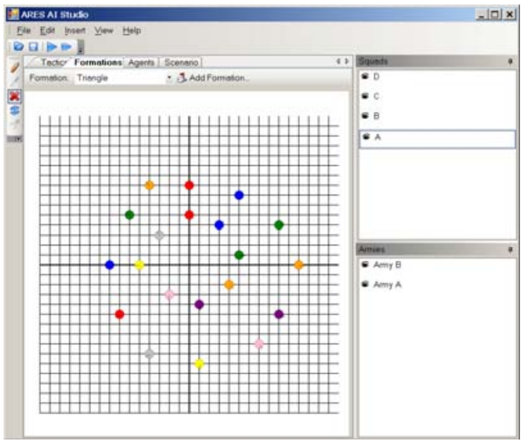


The ARES Studio is Scenario Editor and game tool for ARES, enabling the player to create scenario maps and AI features. The screenshots below detail its robust features:



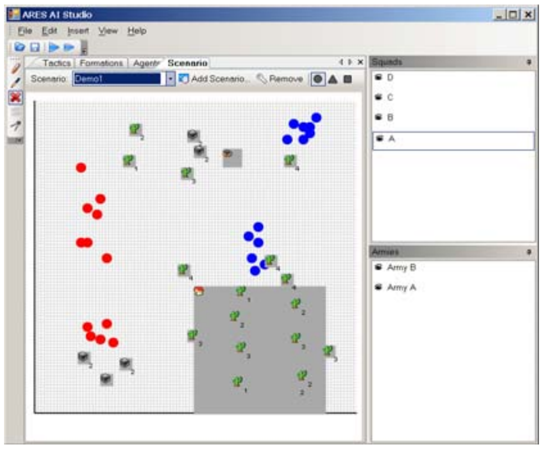
Tactic: Flank 2

Tactics Editor used to create preplanned "attack routes" for squads of soldiers which Captains will choose from when attacking an enemy. Flanking can take enemies by surprise—or leave your forces in disarray!



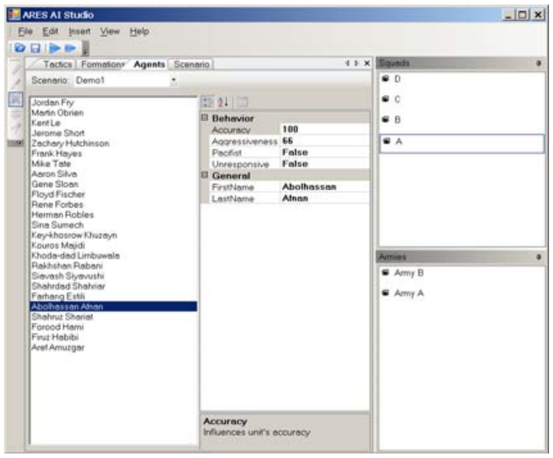
Formation: Triangle

Formation Editor used to create formations used in moving and defending



Scenario: Demo1

Scenario Editor used to create a map for the game, including place soldiers of all ranks and objects such as trees, buildings, shrubs, barrels, and burning cars.



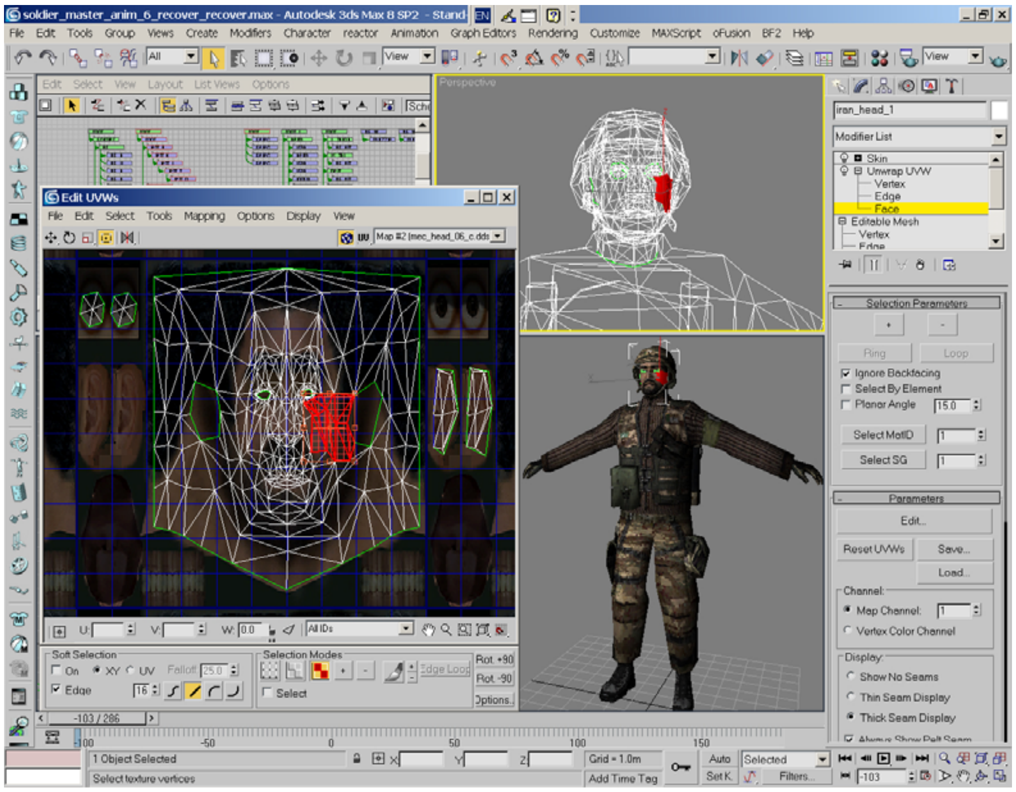
Scenario: Demo1

Jordan Fry
Martin Obrien
Kent Le
Jerome Short
Zachary Hutchinson
Frank Hayes
Mike Tate
Aaron Silva
Gene Sloan
Floyd Fischer
Rene Forbes
Herman Robles
Sina Sumech
Key-khosrow Khuzayn
Kouros Majidi
Khoda-dad Limbuwala
Rakhshan Rabani
Siavash Siyavushi
Shahrdad Shahriar
Farhang Estili
Abolhassan Afnan
Shahruz Shariat
Forood Hami
Firuz Habibi
Aref Amuzgar

Behavior	
Accuracy	100
Aggressiveness	66
Pacifist	False
Unresponsive	False
General	
FirstName	Abolhassan
LastName	Afnan

Accuracy
Influences unit's accuracy

Agent Editor edits attributes of soldiers placed on the map in the Scenario Editor.


soldier_master_anim_6_recover_recover.max - Autodesk 3ds Max 8 SP2 - Stand
File Edit Tools Group Views Create Modifiers Character reactor Animation Graph Editors Rendering Customize MAXScript oFusion BF2 Help
Edit UVWs
File Edit Select Tools Mapping Options Display View
Perspective
Modifier List
Skin
Unwrap UVW
Vertex
Edge
Face
Editable Mesh
Selection Parameters
Parameters
Edit...
Reset UVWs
Save...
Load...

iran_head_1

Map #2 (mec_head_06_c.dds)

Ignore Backfacing: ✓ · Select By Element · Planar Angle: 15.0

Select MatID: 1 · Select SG: 1

Channel: ● Map Channel: 1 · ○ Vertex Color Channel

Display: ○ Show No Seams · ○ Thin Seam Display · ● Thick Seam Display

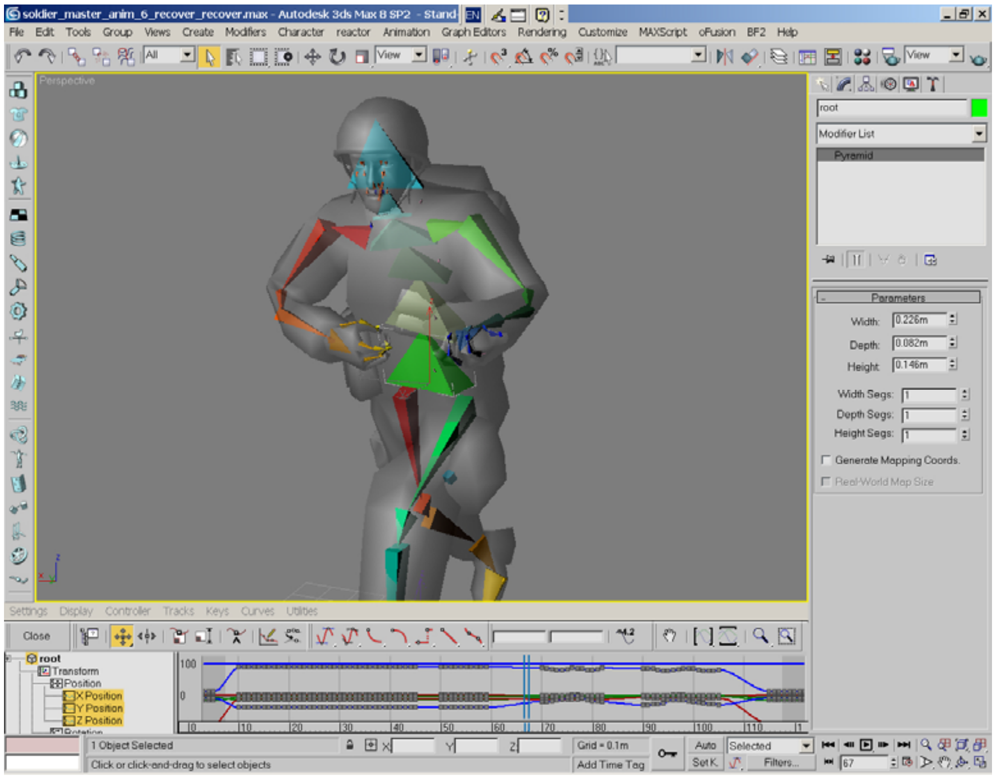
Soft Selection: On · ● XY ○ UV · Falloff: 25.0 · ✓ Edge: 16

-103 / 286

1 Object Selected

Select texture vertices

Grid = 1.0m


soldier_master_anim_6_recover_recover.max - Autodesk 3ds Max 8 SP2 - Stand
File Edit Tools Group Views Create Modifiers Character reactor Animation Graph Editors Rendering Customize MAXScript oFusion BF2 Help
Perspective
Modifier List
Pyramid
Parameters
Settings Display Controller Tracks Keys Curves Utilities
Close
root
Transform
Position
X Position
Y Position
Z Position

root

Width: 0.226m · Depth: 0.082m · Height: 0.146m

Width Segs: 1 · Depth Segs: 1 · Height Segs: 1

Generate Mapping Coords. · Real-World Map Size

1 Object Selected

Click or click-and-drag to select objects

Grid = 0.1m

3D Studio Max 8 was used for UVW mapping and skeleton animating the models

Five Rights & Five Wrongs

Five Rights

1. *Ogre3D Engine*. Using the open-source Ogre3D graphics engine from the very start turned out to be a great choice. The real plus to using Ogre was not that it was necessarily more powerful than alternatives, but that it had strong community support, lots of tutorials, and forums. Another huge advantage was the oFusion plugin for 3dsmax 8, which allowed Johnny, our graphics man, to see how his artwork would appear in-game before he exported models. The built-in scenegraph was also a nice feature.

2. *The C# Programming Language*. Andy blew us all away when he created the Win32 *Ares Studio* editor using C# in a little over a week (he should be a poster-boy for Microsoft). This editor was a huge productivity gain and allowed us an easy way to test the range of possible scenario configurations for our application. The BiG Studios lecturers were right—game tools are important after all.

3. *Graphic Assets from Battlefield 2*. Before any graphics were implemented in *ARES*, there was a big looming question—how can get game graphics that look hot in our game? Johnny had a hunch that it may be possible to rip graphics from EA's popular war-game *Battlefield 2* and put them in our game. After long hours of trial and error, he was able to find a way to rip BF2 assets into 3dsmax, though they were unanimated and unskinned—this was still in February! Obviously if this were a commercial game we'd have to create our own original graphics, but in terms of making a class project this BF2 ripping proved crucial for building the "killer app" that the course staff demanded.

4. *Superior Planning*. Our schedule turned out to be perfect, as we were able to implement everything in our ambitious A+ spec with little time to spare. This can be attributed to a number of factors: good assessment of our individual abilities, good division of labor, regular meetings, and designated weekly coding nights where we all got on a Skype conference-call and coded at the same time. Forcing ourselves to demo new features *every week*—an A+ spec requirement that we met—was a good way to ensure that we never slacked behind on development.

5. *Une certaine je ne sais quoi*. From start to finish, we challenged each other to think outside the box and devise new features—tactics, formations, personality—to make meeting our specs easier. As we came to trust that we were all competent

thinkers and coders—and that we were all putting in long hours to make this project work—we were able to divide up the labor in such a way that each person was doing a well-defined job that played to our strengths. This in turn boosted team morale and encouraged us to put our best work into the project. We all incorporated subtleties that went above and beyond the A+ spec—look closely to see what you can find!

Five Wrongs

1. *Debugging the AI*. Debugging the AI was harder than anticipated because there turned out to be so many special cases and combinations of unit actions. To address this, we implemented an XML dump feature where we could press a button and instantly dump all game state parameters into a well-organized format and see what the heck was going on.
2. *Skinning and Animating the Models*. While ripping the graphics assets from *Battlefield 2* was a big boost, it turned out that that was merely the beginning of graphics work. Skinning and animating the models was a larger than anticipated task, and the only way to accomplish it was to slog through it by brute force. Naturally, the oFusion Ogre3D plug-in aided in the graphics implementation, but the graphics work was still a pain—we all have a better appreciation for what graphics artists have to go through!
3. The ARES AI model is inherently computationally expensive. To address this problem, we made a number of design choices to streamline our app. For instance, we looked at ways to use A* pathfinding a minimal amount of times to form a route, such as breaking the route into smaller segments and only using A* when needed; there's no reason to compute a whole path if a unit will be killed before he can take three steps. Mike optimized the General's Minimax search by cutting down the state significantly. In the end, we got 300 units to run on screen with reasonable framerate—better than many contemporary RTS games.
4. *We worked our asses off*. It's possible that we could have done a simpler project, spent less time, gotten more sleep, and still have done well in the class… but we wanted to be unique and innovative, and that's the price to pay.
5. *If only we had more time*… we could implement all of the really cool pipe dream ideas we had. A big challenge with *ARES* was to stay focused on core features in the A+ spec. Read the following pages and take a glimpse at what we could do with more time!

The Future of ARES

Throughout the development cycle, our team continually thought of ways to expand the original ARES spec above and beyond what was required. Some of those ideas, including the ARES Studio scenario manager, were so good that our team agreed they were worth extra time investment; the rest were left as pipe dreams as we focused on our core set of features. Here we present some of our best ideas that didn't make into our May 12th spec:

AI Improvements

- *Promotions.* If we had another month to work on Ares, our team agrees that making unit promotions when a leader is killed is the first feature we'd implement. In our May 12th release, a known er… feature is that when a unit's commander is killed, eventually he will cease doing new actions and stand in place because he is no receiving orders. This is as should be; we don't have any system in place to create a new commanding officer. In our early designs, we intended for units to have some period of time of chaos after their commander is killed, at which point they elect a new captain and continue the war.
- *Dynamic reorganization.* Why stop at promotions? The army should be able to dynamically reorganize itself. A colonel may decide to merge two of his Captain's units, or move soldiers from a heavily-staffed unit to an understaffed unit. If one unit's aggressive personality is bothering his comrades in arms, he may be shuffled somewhere else in the army. Of course, for any switches or merges to occur, the two units would have to be in physical proximity of one another.
- *Horizontal AI / peer-to-peer AI*. Rather than a purely "vertical" hierarchical AI structure—that is, Generals, Colonels, Captains, etc.—we could add a "horizontal" structure where the Captains could know about other Captains in their proximity and could team up to accomplish objectives. Captains may decide to abstain from fighting if they know that more firepower is only a minute away. Arguably this function could be managed entirely by the Colonel or General, but it would be interesting to experiment with even on paper.
- *Game abstraction layer*. To make the ARES AI system a true middleware application, we'd need some kind of game abstraction layer that would allow our

system to be dropped into any game—i.e. any end user could implement it. Such an abstraction should be able to work in a variety of genres, including Real-time Strategy, First Person Shooter, and Simulation.

- *Parallel processing*. The architecture employed by ARES is inherently parallelizable, and could benefit greatly from modern symmetric mult-core CPU architectures. Units would be divided into processing groups; at the beginning of each turn, units would "think" on an appropriately assigned processor. This would require a locking layer to make our data structures thread safe.
- *AI priorities*. The AI should have different possible objectives or overall strategies to guide it's choice of tactics. For instance, "kill the enemy general" is different from "kill the maximum number of enemy units" is different from "occupy as much terrain as possible." Allowing a player to set these objectives for Colonels he commands would produce some pretty cool results.
- *Location context*. This means having some way to give the AI understanding of terrain more than just "open space". The AI General should be able to prioritize controlling bridges, roads, and hills as would be done in a real war. The AI should know that by controlling a bridge, it will open future opportunities, and conversely, giving the enemy control will expose his weaknesses.
- *Unit emotions*. The AI should implement behavior responses to emotions such as intimidation, high or low morale, rage, etc. All this was spec'd out pretty well in our design stage but got cut when we were determining our spec.
- *Bases / Patrolling*. Another idea we had early on is that when units reach an order's target destination, they would then set up a base and "dig-in" so to speak which would give them a defensive bonus. From the base they set up, the captain would send squads of grunts to go out and scout/patrol the area to extend the unit's sight range. This is not an easy feature to build and was cut early on.
- General George S. Patton said "A good plan violently executed now is better than a perfect plan executed next week." This quote inspires an idea for *ARES*: what if Captains were able to take more than one turn to make plans, but plans would get better with the amount of time given to plan them. Currently every time a Captain makes a battle plan, he searches the space of tactics for an optimal plan (see "*ARES AI System*" section). What if players could tell Captains to take additional turns of planning and search a wider space of possible actions? This would add a very human touch to the AI system.

- At the moment, the AI considers tactics based on results—not on move times or current destinations of units. Thus units in the same squad can be assigned routes which will make them do criss-cross patterns on the battlefield. With another month of coding, this would be resolved cleanly.

ARES AI Studio

- AI studio could be expanded to provide additional properties which affect the agents, including physical attributes (speed, armor, etc.) and emotional attributes (morale, moodiness, integrity, etc.)
- More usability features could be added such as group select, copy and paste, and undo/redo.
- An XML style sheet (.xsl) could be provided to visualize game XML files in web browsers, both locally and on the Internet.

Graphics and Presentation

- Add more types of obstacles to the AI studio, providing a more realistic in-game experience while in realistic view. ARES currently supports 15 types of obstacles.
- Add moving obstacles such as animals and civilians
- Add transit systems such as roads and bridges could add to the game's realism. Units could have different movement speeds on different terrain types. In addition, weather particle systems such as rain, snow, and sand storms could also enhance realism.
- More animations, including prone position…
- Add different camera angles, including a chase-cam or "first-person perspective"

Gameplay Variety

- Add vehicle units such as tanks, armored vehicles, helicopter gunships, artillery, and warplanes. Such units would exist in game logic as simply units with a large number of hit points, making them more difficult to destroy. Their strategic value would need to be taken into account by the MDP and minimax AI systems. In addition, they would require extensive animations, models, textures and sound.
- Add grenades, landmines, and other explosive weapons. In addition to looking *sweet*, explosive weapons would add a new dimension of AI realism—captains would now have to balance units between clustering up and spreading too far out.

- Add third-person RTS-style control or even first-person shooter style control of individual units.
- Make a God-game or General-game where you not only play against the AI, but your units use the AI system to carry out your orders in an organized fashion.
- The AI Obstacles/targets such as bridges. Objectives could be tiered so that accomplishing one objective opens up other possible objectives.
- Make game bigger—our early specs envisioned battles taking place over a 50 square mile area, but in the finished game it seems to take place in a section of a small village. In order to make the game have a bigger area, we'd need vehicles and roads to move the troops; it wouldn't be believable or interesting to see two squads take an hour to run 5 miles on foot towards each other for a 30 second fire-fight.

Conclusion

ARES was an outstanding success. The CS196-2 class challenged us to break the mold on the same tired video game models on the market now, and our team responded by designing an innovative new hierarchical model for AI—something never been done before. Three months and 25,109 lines of code later, our four-man team had a “killer app” to demo our AI system. Truth be told, every member of our team gave his utmost effort in developing this app and every member shined in his respective work. We wish to thank the CS196-2 staff for giving us the unique opportunity to work on a semester-long project. This course was a valuable experience for all of us, and many lessons about design, application structure, coding, graphics, and teamwork were learned along the way. While at the moment we’re all a bit tired of developing this app, who knows what we may cook up next semester?

“I shall return.”

- General Douglas MacArthur

ARES Instructions for Use

Prerequisites

ARES is a complex project and relies on a number of external dependencies, all of which must be satisfied for ARES to function. Please ensure that the following recommended minimum system specifications are met:

- Intel Pentium 4 3.0 Ghz CPU
- 1 GB DDR RAM
- NVIDIA GeForce 6800 with 256 MB RAM or ATI X800 with 256 MB RAM
- Microsoft Windows XP Professional SP2
- DirectX 9.0c and latest video card drivers
- 1 GB Disk Space
- 2-button mouse with scroll wheel

Note that in the above configuration a video card that is Shader Model 2.0 compliant is required. ARES is a graphically and computationally intense application, and therefore deviating from the above specifications will produce sub-optimal performance.

In addition to the base system requirements listed above, the following software packages are **required** as well:

- Microsoft Visual Studio **2005** (supplies necessary runtime libraries)
- NET Framework **2.0** (required for ARES AI Studio which is C#-based)
- Microsoft Internet Explorer 6.0 or later (supplies MSXML library)
- Microsoft Windows XP Service Pack 2
- DirectX 9.0c (please ensure that you are running revision “c” of version 9.0)

Quick-start

Once you have reviewed and applied the above operating environment you are ready to use ARES. Extract the binary distribution using Windows Explorer into a new folder called, say, “ARES”. Ensure that this folder and its contents have read/write access (do not run ARES off of a CD-ROM/DVD-ROM).

To begin ARES, run **/bin/release/AresStudio.exe** using Windows Explorer. ARES is distributed with a sample game.xml file. To load it, access “Open…” from the File menu. Open the file **/media/game.xml** to open the default scenario.

You are now ready to try ARES. Notice that there is a drop down list labeled “Scenario:”. Select a scenario of your choice and then choose “Start Scenario” from the View menu.

The “OGRE Engine Rendering Setup” dialog box will open. From the “Rendering Subsystem:” drop down list, choose “**Direct3D9 Rendering System**”. If this option is not available, then press cancel and refer to the prerequisites listed above.

Under Rendering Device, **ensure that your video card is selected**. Specify a resolution preference as well as whether to run in Full Screen mode. Click OK to begin the demo.

ARES starts in pause mode. To unpause, click the play button. To quit, choose Q from the buttons at left. Click the O button to switch views (battle view or symbolic). For larger battles, symbolic is recommended. Click the V button to turn on order hierarchy (this option is only available in symbolic view). Clicking on a unit will display a window of the unit’s details. Pan by moving the mouse to each edge of the screen; zoom using the scroll wheel.

Known Issues

- Ray-scene intersection is imperfect, and can produce erroneous unit selections. In larger maps, sometimes units may only be selected when they are near the central area of the map.
- Staring OGRE with the wrong video card selected may cause problems.
- Illegal tactics and formations are not validated by game logic, and so deliberately specifying invalid tactics or formations will cause a crash.
- Zooming with the scroll wheel was only tested using Microsoft Intellimouse 4.0 mice, and so using other mice (such as a Dell wired mouse) will result in slow zooming.
- Zooming very close to the terrain in battle view may cause the terrain to disappear. This is caused by a bug in the HLSL post filter. Zooming out will cause the terrain to re-appear. This only happens on certain video cards under certain conditions.