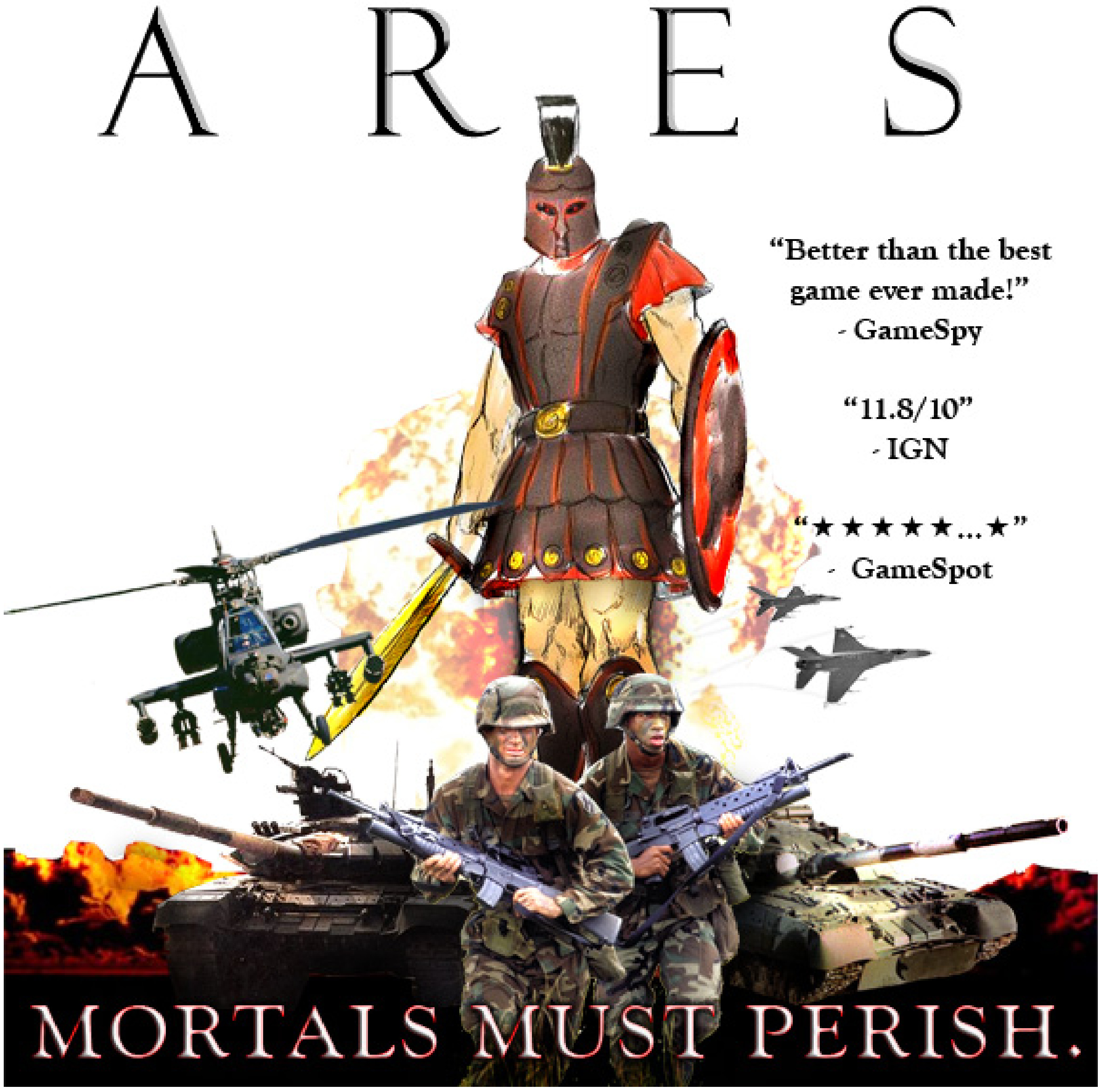


*CS196-2*

# *Project Proposal v3*

Mike Shim
Andrew Bragdon
John Shields
Yee Cheng Chin

## 1.0.0 Motivation

While a lot of games today boast intelligent A.I., with units making intelligent decision, the fact is that units in games nowadays are still incredibly stupid. A primary reason is that the way the A.I. works involves a monolithic A.I. that controls every single unit's actions, with each unit blindly following orders from the all-knowing A.I.

Another problem to the monolithic A.I. is that it doesn't account for the fact that real commands in an organization like the military are passed down through a chain of command. A sergeant would not receive orders from the general, but would probably be receiving the order from a captain, who receives the order from the colonel, who receives the order from the general. This lack of indirection often leads to the conception of the A.I. being inhuman. For example, an enemy grunt will sometimes know the player's location even without having seen the player because the A.I. would know where the player is and issue a direct order to that unit to attack the player.

A solution to this problem is instead of making units blindly following orders, implement a hierarchal A.I. system where each unit would be able to choose what to do, and also receives order through a chain of command, instead of directly from the command overlord. For example, in a military, the general could be issuing an attack order to a colonel, the colonel would then take the surroundings into account, and attempt to make the right decision and give the correct orders to its captains, who then order its grunts.

The A.I. system powering Ares will provide real time strategy games with an enhanced sense of realism, suspension of disbelief, and enable new levels of engagement on the part of players. Our ultimate goal with unlimited resources would be to create player exclamations such as the following: “Oh my god! I can't believe the A.I. just did that...”. Existing real time strategy games never produce sensations like these and really only act as a placeholder opponent for users who want to learn the game or play offline. In short: serious real time strategy game players spend their time playing against other humans because the A.I. is artificial, but not very intelligent. We aim to change that.

## 1.0.1 Our Vision

Our vision is to develop a decision-making AI where agents interact in a hierarchical structure towards a common goal. Agents receive and interpret orders from superiors; they coordinate actions with peers; and they issue new orders to inferiors. This AI would be implemented in a war game to provide realistic perceptions of chain of command, teamwork, and military duty.

For our CS196-2 project we will approach this AI problem by implementing a limited Minimax algorithm for the highest-ranking agent and use Markov decision trees for the mid- and low-ranking agents. We will use four classes of soldiers: Generals, Colonels, Captains, and Grunts. Graphics will be in limited 3D without a physics engine to allow greater computational power to be devoted to the AI engine.

With five years and an unlimited budget, we would:

- Rigorously explore genetic and learning AI models to get the most bang for the amount of processing power.

- Implement a variety of human and armored units and weaponry, including: tanks, snipers, helicopter gunships, airstrikes, paratroopers, artillery, and more with unique AIs for each to provide realistic combat coordination.
- A variety of reconnaissance tools including night vision, satellite spying, and drones to simulate real military intelligence gathering
- God controls (assuming we are sticking with the ARES theme) including forced commands, emotion tweaking, and natural disasters (tornados, fires, floods, etc.)
- First-person shooter style control of units to immerse the player in the action
- Optimized graphics, textures, and explosions
- 5.1 channel surround sound
- Storylines and missions
- Stripped down physics engine that provides for ballistic projectiles and other physics elements while not using much processing power.
- Multiplayer and/or MMO type gameplay
- Embezzle some of our unlimited funding

Our AI technology is the main feature of our game, and after years of development it could be licensed to other game developers for a variety of genres. Examples of possible games include:

- A “Sim-Corporation” game, where company agents from the CEO down to the mail clerk interact in a hierarchical structure. Managers evaluate business opportunities then give commands to their subordinates to exploit those opportunities.
- An FPS combat game like Doom 3 where you must execute your mission as part of a team in order to survive.
- An environment where a food chain of animals exist and they manage their actions to ensure the survival of the ecosystem.
- A family game to teach children about respect for their elders.

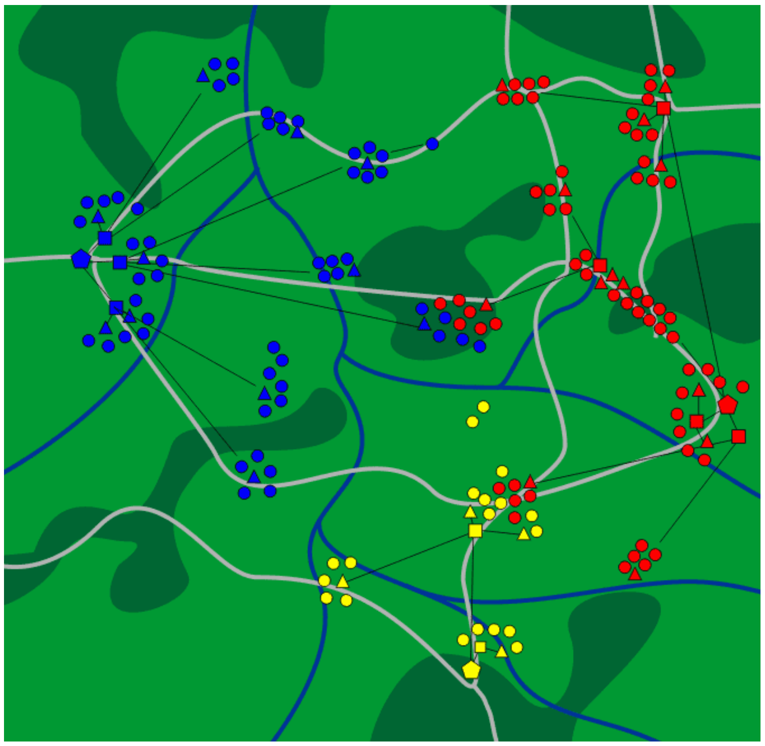
### 1.0.2 Possible Killer Demo

This game would basically be a hybrid of FPS/RTS where the game would follow the progression of an individual soldier. You start out the game as a new recruit that’s just gotten into boot camp. The game is essentially a first person shooter and you know very little of what is going on around you in the world. For the time being, you are given orders by your superiors that you have to obey. Eventually you will move up to the rank of a Captain. You will be in charge of a whole squad and you’ll be able to issue commands to your soldiers. However, you still have superiors that will be giving you commands. You’ll continue to progress and eventually reach the rank of Colonel where you’ll be in a more real time strategy role. You’ll have control of hundreds of soldiers in your hands. You’ll be in charge of maintaining supply lines, training new units, and several other tasks to run an effective Brigade. However, you’ll still have to report to your general. Finally, you’ll reach the esteemed position of the General in which you run the show. You can decide anything. From where to produce units to where to strike, everything is in your hands.

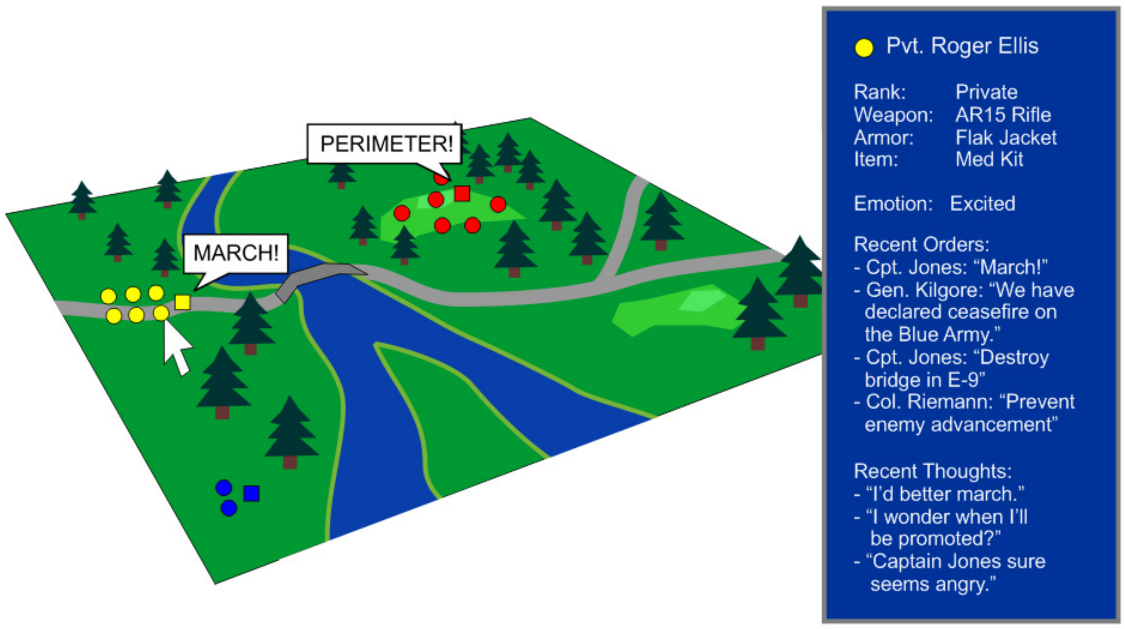
Your superiors and underlings will not be dumb units that follow your command like mindless drones.  Each unit will be intelligent enough to be able to make decisions for themselves and act on those decisions.  You won't have to micromanage when playing a commander which will allow you to focus on broader command goals.  Your underlings may also decide that your command isn't the right one to follow and may go ahead and follow their own commands.  Also, since this is to simulate a real war, information between units will be lagged.  You may run into situations where your allies will fire upon you because they didn't know you were there.  In the end, this should produce the most realistic war simulation to date.

## 1.1 Story Board

*Map View*



*Zoomed-In, Info on a Private*



● Pvt. Roger Ellis

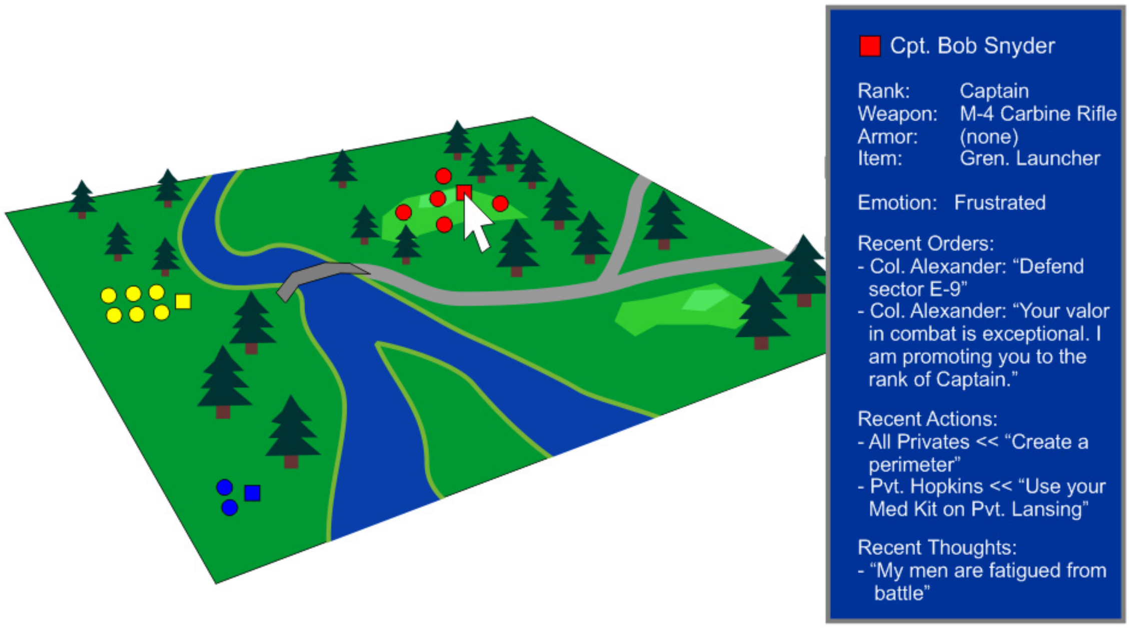
| | |
|---|---|
| Rank: | Private |
| Weapon: | AR15 Rifle |
| Armor: | Flak Jacket |
| Item: | Med Kit |

Emotion: Excited

Recent Orders:
- Cpt. Jones: “March!”
- Gen. Kilgore: “We have declared ceasefire on the Blue Army.”
- Cpt. Jones: “Destroy bridge in E-9”
- Col. Riemann: “Prevent enemy advancement”

Recent Thoughts:
- “I’d better march.”
- “I wonder when I’ll be promoted?”
- “Captain Jones sure seems angry.”

*Zoomed-In, Info on a Captain*



🟥 Cpt. Bob Snyder

| | |
|---|---|
| Rank: | Captain |
| Weapon: | M-4 Carbine Rifle |
| Armor: | (none) |
| Item: | Gren. Launcher |

Emotion: Frustrated

Recent Orders:
- Col. Alexander: “Defend sector E-9”
- Col. Alexander: “Your valor in combat is exceptional. I am promoting you to the rank of Captain.”

Recent Actions:
- All Privates << “Create a perimeter”
- Pvt. Hopkins << “Use your Med Kit on Pvt. Lansing”

Recent Thoughts:
- “My men are fatigued from battle”

### 1.1.1 Gameplay

It is appropriate to again emphasize that our project is primarily an AI Research project and secondarily a game, and we aim for our proposal to be accepted on those terms. That being said, we are exploring a number of gameplay options and here are a few of them:

- *God Game:*
  As Ares, God of War, you must achieve scenario objectives by influencing the world and minds of the warring mortals. For instance, your object may be to cause as many casualties in five minutes as possible. You would achieve this by making the commanders of the losing side brave so they will continue fighting, and cause a flood to block their retreat. You have a limited “God Power Bar” that is depleted by your actions and recharges with time. Most actions would be executed by “Pointing and Clicking” like other RTS and simulation games.
- *Gods Game:*
  Like above, except now you compete against and Athena, Goddess of Strategy and Wisdom, who will aid soldiers to act intelligently, end the war quickly, and minimize casualties. You could play the game as either God, or play in a 1-on-1 versus game.
- *General Game:*
  You are a General and you play against a Minimax AI General or a human. You issue orders/objectives, the only caveat being that there is command lag before your orders get through. Your first order is free. Lag is longer with distance, so you’ll have to move all your units and yourself smartly to win the game. You can also tweak organizational mechanics. The game ends when one General is killed. You can issue several orders per colonel at a time, which are then queued. You can also rearrange hierarchical structure, but units must be in contact range to execute your organizational changes.
- *Command Everybody Game*
  Like above, but now you can command everybody, but are limited by command lag. All commands will be automated unless you are controlling that unit. You will want to spend as much time as possible controlling at the higher levels, but once you make an order as a Colonel you will then play as a Captain to execute that order. You are prompted to switch up when an order has been accomplished or failed and a new order is needed. Generals and colonels become temporarily unavailable after orders have been given, and there is a big penalty for revising orders. This gameplay format lends itself to multiplayer, but two people can’t possess the same unit at once; if a player wants to be the general the other player gets 30 seconds before he is booted/switch is forced.
- *Gods and Generals*
  Some combination of all the above ideas: play as any unit or a God and wage war!

## 2.1 Analysis and Statement of Requirements

The primary goal of this project is to design, implement, and optimize a new Artificial Intelligence system for playing strategy games. The secondary goal of this project is to create a game that demonstrates this system. Note that this is a broader set of requirements, please refer to the Design section for implementation details.

The innovative concept in this project is the introduction of tiered, hierarchical, chain-of-command-like decision making on the part of the A.I. system. Rather than a monolithic A.I. which simultaneously controls all outstanding units (which is utilized in most games, such as Age of Empires, Starcraft, etc.), our system will treat each unit as an autonomous agent which makes decisions in the context of orders from its superiors as well as available information.

Our hierarchy will consist of four levels: Generals, Colonels, Captain, and Foot Soldiers. Orders propagate downwards and are subject to interpretation at every level, however they are parameterized. There will typically be a single general, the highest commanding officer, and two or more colonels directly under the general. The general will make decisions based on a minimax/state evaluation look-ahead algorithm. The exact number of look-ahead moves will be determined during testing to best-optimize performance and difficulty. The colonels, captains, and soldiers, on the other hand, will use simple MDP models to determine their actions – as we do not anticipate the CPU headroom necessary for look-ahead on all units in real time.

Orders will be parameterized but vague enough to be subject to interpretation. The orders we are currently considering will be “Move To” and “Attack”. These orders will be accompanied by a sector location to attack. It is important to understand that the sector size – and therefore the granularity of the order – will change based on the command level. For example, when a general issues an order, each sector size will be quite large – perhaps 1/16th of the total map area. Whereas a colonel’s order granularity will be smaller, say 1/64th of the total map area, and so on.

The two Generals of either side will use a Minimax look-ahead state-evaluation to plan broad strategy. We believe at time of writing that Minimax is the most intelligent AI algorithm that will fit within our processor restraints. Colonels and Captains will use a limited Utility function evaluation to choose one of several preset attack/defense/retreat patterns that they believe will produce the best results given the current state. These decisions may be heavily influenced by the units’ personalities, beliefs, and leadership styles. Grunts will generally follow orders in Military “Standard Operating Procedure” fashion, unless they get a cue to behave in a random way from rage or shell-shock. For more on this please read sections 2.2.3 to 2.2.6.

*A.I. System Requirements*

- Multiple hierarchy system which models the chain of command in a modern military, such as the U.S. Army.
- Each soldier, Captain, colonel, and general will be represented by an autonomous agent within the A.I. system of the corresponding type (4 in total).

- Just as in a real chain of command orders will be issued from commanding officers to subordinates.
- The map will be subdivided into sectors that vary in size depending on the rank of the agent. So a general agent will work with much larger-sized sectors than say a Captain. This varying granularity simulates the tactical versus strategic perspectives that real officers face.
- Orders will be propagated downwards from an agent to agents under its command, i.e. General sends orders to Colonels, Colonel sends orders to Captain, Captain sends orders to troops. Orders, like the sector system, will change in granularity depending on the command level (i.e. a General's orders are more specific than a Colonel's). Orders will be parameterized (see design section).
- Generals and Colonels will choose orders based on a minimax/state evaluation look-ahead algorithm. The state evaluation functions will be provided in the design section of this document. Captains and soldiers, for computational reasons, will use simple heuristics, rather than the minimax algorithm.
- In the event of a leader fatality, the subordinates will automatically reorganize themselves based on a time-to-reorganize penalty.

### *Information Lag Concept*

Colonels and captains will receive information about the broader battle via a realistic information-lag system. The purpose of this system will be to emulate battlefield intelligence reports, which are of a time-sensitive nature and can become out of date. This means that colonels will have information about the location of units outside their range of vision, but they will be unsure as to the reliability of this information. Information reports will be transmitted periodically from superior officers down to inferior officers: colonels will receive reports from generals, and captains will receive reports from colonels. Reports will be inserted into a queue so as to introduce variable time lag between the reports being sent and actually received by the recipient.

The end result of this is that the officers who receive reports will now have more information to work with, but will know that it may be out of date. This means that this information will influence the Markov decision process (MDP) executed by the agents, but may not be a deciding factor. This will help to create realism as units may make mistakes based on inaccurate information, but they may also make superior decisions based on the information.

### *User Interaction*

As mentioned previously, the primary goal in this project is to develop a solid A.I. system. However, we will also be implementing a game designed to demonstrate the underlying technology – namely, the A.I. system. The game will feature a 3D view of the battle including, but not limited to, a realistic 3D terrain environment, troop locations, and actions (such as units firing). This view will feature a simple button interface at the bottom that will allow the user to control settings and interact with the game.

The user will be able to switch between two views: presentation view and symbolic view. Presentation view will essentially be the view mentioned above, including units and terrain. The symbolic view will also include terrain, however instead of units, symbols representing the officers: a pentagon for the general, a cube for the colonels, a triangle for the Captains, and circles for the troops. Lines and other symbology will be used to show the hierarchical chain of command and outstanding orders.

The user will be able to quickly zoom in and out to various levels of the battle. In addition, they will be able to set up the battle prior to its unfolding. Rather than controlling a particular side in the conflict, the user's role will be to play as Ares, the God of War, presiding over the conflict. The user will be able to query the detailed status of a given unit. To aid in this, each unit will receive a generated name based on a list of common first and last names. The objective of the game will be for the user to watch over the battle.

We may have the ability to adjust unit personalities and perhaps give orders to units.

*Formations*

We plan to implement formations for units. A formation dictates how units will move from one point to another. We will have formations for flanking, pincer attack, etc. The advantage of using formations is that it will allow the units to carry out orders more dynamically. Different formations will affect the outcome of a battle since when a unit is flanked (attacked from the side), it will be at a disadvantage. This can be implemented either by the unit being easier to hit or having a lower accuracy. Formations can also be used in situations like one unit approach the enemies and lure them into an ambush.

Formations will be implemented as a set of paths. Each path is a set of waypoints. Each path also has a probability associated to it, dictating how much of the team should use that path when attacking. The paths will be in a base coordinate system starting from (0,0), moving in the positive y direction. When a commander issues an order, he may also choose a formation that is the best for the order. The paths in the formation will then be transformed to use the commander's coordinate system using linear algebra. For example, if a captain issues an order to move in the positive x direction, the paths will be rotated 90 degrees clockwise, and translated to the starting point of the order.

After transforming the set of paths in the formation, the commander will then assign units to each path according to the path's probability. Given a path, a unit will then attempt to move along it using A* pathfinding. Right now, the unit will move to each waypoint in the path given, but more intelligent ways of moving along a path may be implemented.

For example, if a commander considers that it is advantageous to perform a pincer movement, then it will use the pincer formation. The formation will likely have two paths, one flanking from right, the other one from left, each with a 0.5 probability. Then half the team with move along the path on the right, and the other half moving along the path on the left.

*Game Requirements*

- Three-dimensional battlefield view, which includes terrain and unit locations.
- Two views will be selectable by the user: a presentation view which shows soldier models and a symbolic view which shows the command hierarchy in terms of a tree structure.
- User may choose game parameters.
- User may see current stats, orders, name, and "thoughts" of any agent in the battlefield.
- Game will feature sound.
- Kickass boxart, marketing and website.

## 2.2 Design

Computer Programming Languages: C++ possibly supplemented by XML and Python

Development Environment: Visual Studio 2005 Professional Edition

Version Control and Collaboration: Subversion (SVN), CVS if you really want it

Graphics Environment: 3D Studio Max 8 with oFusion Ogre3d Plugin, Adobe Photoshop CS2

Sound Environment: Sony SoundForge, Acid and Home Sound Studio

Machines: 4 Windows XP Pro and Media Center

Graphics Engine: Ogre 3D v1.0.6

Sound Engine: BASS v2.2.0.3

Artificial Intelligence Support Package: OpenSteer v0.8.2 and Shim's Awesome A* Pathfinding Library

Platform: Windows XP with DirectX v9.0c

System RAM: 512 MB

Processor: Pentium 4 2.5 Ghz or Faster/Compatible

***Classes***

*ApplicationRoot*
Provides a top-level class that is instantiated by the main() function and stores pointers to important application objects. Stores its own instance as a static variable.

*GraphicsManager*
Manages the scene graph objects, renders every frame on idle. Manages the scene graph. References other graphics objects, such as Terrain (see below). Also manages the camera and active rendering matrices.

*TerrainManager*
This will encapsulate some open source terrain code that we will be using (provided as a support project within the Ogre development community). Terrain data will be stored as a 2-dimensional heightmap.

*SoundManager*
Manages caching of sounds and plays sounds on demand. Stores its own instance as a static member variable.

*InputManager*
Handles all user interactions. Stores and instantiates objects to the input objects built into Ogre 3D. Stores its own instances as static member variable.

*GameLogic*
Manages flow of control of the game. Almost all actions go through the game logic class which will execute these actions. Will also route sound. This class interfaces with the ArmyManager class. Will stimulate animations and movements.

*ArmyManager*
Manages one side of the conflict, encapsulating the top level of all of the unit objects. Provides accessor methods.

*ScenePartition*
Manages an QuadTree (2D) of all of the units. Will be used to query nearby units.

*ForwardGeneralState* : public *ForwardState*
Encapsulates a look-ahead state in the minimax algorithm used by generals at the general granularity level.

*ForwardColonelState* : public *ForwardState*
Encapsulates a look-ahead state in the minimax algorithm used by colonels at the colonel granularity level.

*Order*
Encapsulates information about an order, issuer, issuee, parameter, order type.

*Unit* (abstract class)
  Move()
  Fire()
  GiveOrder()
  ReceiveOrder()

*MDPUnit* : public Unit (abstract class)
Provides a basis for heuristic units which make decisions based on a stochastic algorithm.

*StateUnit* : public Unit (abstract class)
Provides a basis for units which make decisions based on minimax state evaluations.

*Grunt* : public MDPUnit
Implements a foot soldier unit – the most basic unit in the game.

*Captain* : public MDPUnit
Implements a Captain unit, the lowest-level leader unit in the game.

*Colonel* : public MDPUnit
Implements a colonel unit which issues orders to Captain units under its command.

*General* : public StateUnit
Implements a general unit, the top-level unit in the game, which issues orders to colonels.

### 2.2.1 Flow of Information

The GamePlay class will be in charge of the game and the master State. It will tell the ArmyManager class when the AI can run. It will also be in charge of determining whether a unit has been killed or not. The ArmyManager will be in charge of telling each unit when they can run their AI code. It will allow the grunts to run continuously while the Generals will run only every once in a while. The State of each unit will be determined by their field of view as well as information it has received from other units. For example, the General will only be able to see a little bit around it and information sent in from the Colonels. The grunt will report in its state to the Captain in which the captain will have a lag time of a few seconds or so. The Captain will report to the Colonel the combined states of all the grunts the captain commands. This will probably have a lag time longer than the grunt-captain lag time. Finally, the Colonels will report state to the General every once in a while. This will take a lot longer than the other lag times. Also, at each step, the State will be sent downwards too.

An example of this flow of information: A grunt reports seeing a hostile in front of him. 1-3 seconds later the Captain is informed and the state has been updated at the Captain level. About 3-5 seconds later the state is then sent up to the Colonel who now knows of the hostile. About 5-10 seconds later the general has been told of the hostile. During this time, the hostile has had up to about 20 seconds to have performed an action.

### 2.2.2 State Information for Units

Contextual information in our game will be represented by two sets of data for each agent. One set of data will represent the exact surroundings of the unit of absolute accuracy, while the second set of data will represent data that has been propagated down from commanding officers through a time-lag system. The data structures will be covered shortly, but it is important to note now that the data structures will make use of reference counters to automate memory management.

The big picture here is that all of our units and their locations will be stored in a quadtree data structure. This quadtree data structure will allow us to query a specific region to determine the nearest neighbors of a given agent efficiently. Using this backend, the contextual state information an agent uses will query the quadtree from a given point and radius. Quadtree is a good choice for our design because of its nominal running time of O(log n), where n is the number of points. This will return the exact surroundings of the unit, which will not be in question. Likewise the superior officers, will periodically pass down data with a larger radius relative to the officer down to subordinate units via a time lag system, meaning that this information will be of less material value. This will allow more room for units to “pass the Turing test” by making mistakes and generally acting in a more realistic and believable manner.

When a superior officer passes information down to subordinates, due to the time lag factor, will have to clone the data and encapsulate it in a linked list. This special linked list will store its own reference count, which will be incremented by an AddRef() function call for every unit which receives it and stores a pointer. Furthermore, a Release() function call will be used in place of delete to decrement reference count. When the count reaches zero, the object will delete itself. We do not anticipate reference count problems given our design strategy. The quadtree query to determine non-time lag information will not clone the data, due to its immediate nature.

In summary, all of our units will be stored in a quadtree making nearest neighbor and spatial queries fast. Perceived information will be stored in a linked list, which will be also cloned and reference counted in the case of information sourced from superior officers. Lastly, these linked lists will be storing links to actual unit objects.

### 2.2.3 Unit Orders and Actions

The actions in our game for the units will be the result of a function of type Order X State → Action. Given an order, a unit will make a decision of what action to perform depending on what the current state is. The function that makes this decision is described in the MDP section.

There will only be one type of order given to a unit: capture area. This order will encapsulate scenarios such as attacking, defending, and retreating. The order will have two parameters: a parameter in real number indicating the aggressiveness preferred, and the location of the destination. For example, if the number is 0, then the unit will avoid all combat with other enemy units, and if the number is 1, then the unit will try to eliminate all the enemy units it sees. For example, if the commander intends to tell the unit to retreat, it will assign a capture area command to the base, with a very low aggressiveness (e.g. 0). If the commander intends for the unit to defend a certain area, it will assign an order to that area with a relatively low aggressiveness to prevent the unit from pursuing an enemy, but allow the unit to open fire on enemy units (e.g. 0.4).

The units will have three action types: Attack, Defend, and Move. As stated above, the unit will attempt the make the right action based on the order and the current state.

- Attack is a general command that takes no parameter. It will attempt to eliminate all enemies within its radius. When a unit is attacking, it will look for the closest enemy, attempt to shoot at it, or order its subordinates to do so. The success rate of the shooting will be based on an accuracy model that gives a unit some percentage of how often it will hit the target.
- Defend is another action without parameter. It will essentially tell the unit to stay at where it is, and defend the area. It enemies appear, the command will likely be changed to attack.
- Move is an action that takes a location as parameter. Higher level units like colonels will only have a large general location that they can move to, where they will then order lower level units like grunts to move to more specific areas within that general area.

### 2.2.4 Generals' Decision Process

The state evaluations done by the Generals and will be very important to the overall nature of the A.I. system. Therefore we have considered this and determined the following factors that will be weighted (final weights will be determined during the course of our research) and summed together:

- Number of Living Friendly Units (Captains, colonels, and generals will receive progressively higher weight)
- -1 * Number of Living Enemy Units (Captains, colonels, and generals will receive progressively higher weight)
- Area of Territory Controlled (weighted by terrain bonuses, such as high ground)
- Enemy Units Killed (Captains, colonels, and generals will receive progressively higher weight)
- Pre-existing Orders (this will receive a large weight; and will be 1 if the colonel is following an outstanding order from the general, and zero otherwise)

Using a Minimax search, the General will look at all possible actions and make "attack" "defend", and/or "relocate-base" orders for his Colonels.

### 2.2.5 Colonel and Captains' Decision Process

Captains and Colonels will use a Utility Function model to make decisions. When a significant event occurs or an order is received, a Captain will consider up to three plans of action, do a high-level Utility evaluation of each plan given the current state, then choose the plan of the highest Utility. For instance, if a Captain's unit sights an enemy, the Captain will consider the plans "attack," "defend," or "retreat". The Utility for the attack plan may be calculated as follows:

- **Utility of Attack** = Perceived Odds of Winning * Perceived Outcome of Winning + Perceived Odds of Losing * Perceived Outcome of Losing
- **Perceived Odds of Winning** = # of Grunts in Unit * Captain's Faith in Grunts / (# of Grunts in Unit * Captain's Faith  + # of Enemies to Attack * Captain's Fear of Enemies)
- **Perceived Outcome of Winning** = Enemies Killed + Grunts Remaining * Captain's Will to Keep His Grunts Alive

Likewise for retreating:

- **Utility of Retreat** = Perceived Odds of Escaping * Perceived Outcome of Escaping + Perceived Odds of Getting Caught * Perceived Outcome of Getting Caught

Note that in our implementation these functions will be considerably more complex. Once high-level Utility functions are calculated and the Captain chooses the highest-Utility plan, he then does another round of Utility Evaluations to further flesh-out that plan. For instance, if the Captain chose to attack, he would consider 3 of 10 possible preset attack plans and calculate a Utility for each. Again his knowledge of the enemy, his unit, the terrain, and his personality will influence the Utility function. Once a fleshed out plan has been chosen, the Captain will then assign orders to his grunts.

Colonels will follow a similar process to devise plans and give orders to their captains.

### 2.2.6 Grunts Order-Following Process

Grunts in this game will essentially follow order and preset responses to changes in state. We may add some behavioral quirks to grunts such as deserting or charging ahead into enemy ranks, but if this feature is implemented it will have a minor effect on gameplay.

## 2.3 Testing Plan

Bug tracking with Visual Studio team system

*Early Stage Testing*

Incremental testing:

- One unit versus one unit

- Squad (Captain + 10 grunts) vs. squad
- Brigade (Colonel + 10 Captains + 100 grunts) vs. brigade
- Army vs. army

*Late Stage Testing*

We will develop the game with extensive debug and tweaking features that will allow us to adjust gameplay parameters for optimum performance and behavioral complexity. We expect to do numerous trials to determine parameters including:

- Number of units and unit ratios
- Terrain structure
- Combat damage, accuracy, and killing rate
- Unit speed
- Optimal state evaluation processes

We also anticipate during testing we will identify god-like debug features (creating/killing units, changing unit behaviors, etc.) will make for a good God Game.

*Beta Testing*

We will implement a small-scale beta testing team, including Tyler Shields and Eric Shim.

## 3.0 Milestones

Our milestones are based on a bi-weekly review.

| **Time** | **Milestone** |
|---|---|
| February 10 | **Project Proposal Handed In** |
| February 10 | Development environment and source control server online |
| February 17 | 3D Terrain Demo (Ogre)<br>Simple Units Displayable<br><br>Completed Rudimentary Class Hierarchy<br>Working Game Logic<br><br><br>Website online |
| February 24 | |
| March 3 | Polygon-modeled Units<br>Moving Units<br>Working Captain-Soldier Squads |
| March 10 | |
| March 17 | Unit Animations Completed<br>Working Colonel AI and Brigades |
| March 24 | |
| March 31 | *Spring Break* |
| April 7 | Working General AI and Divisions<br>Functioning User Interface<br>Sound |
| April 14 | |
| April 21 | Alpha Software |
| April 28 | |
| May 5 | Beta Software |
| May 12 | **Demo Day** |

## 4.0 Project Rubric

A+ requirements:

- 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
    - Grunts
        - 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

A- requirements:

- AI
    - 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
    - Grunts
        - 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
- 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

B+ requirements:

- AI
  - 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
  - Grunts
    - 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
- Formations
  - Formations govern the relative motions of units to produce an overall shape or formation of units
- Interactivity
  - 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

B- requirements:

- AI
  - 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
  - Grunts
    - 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
- Formations
    - Formations govern the relative motions of units to produce an overall shape or formation of units
- Interactivity
    - GUI button-based UI
        - Ability to pause the simulation
        - Ability to change views

C requirements:

- AI
    - Grunts
        - 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
- Formations
    - Formations govern the relative motions of units to produce an overall shape or formation of units
- Interactivity
    - GUI button-based UI
        - Ability to pause the simulation
        - Ability to change views

## 4.1 Group Workload Breakdown

*Johnny*

80% Project Grade
20% Project Design, Artwork, Sound, Website and Support Code
A – Units make decisions as specified above, good project design, sound, game looks modestly pretty
B – Units make any decisions at all, 3D in-game graphics, game design, website exists
C – No website, complete design failure

*Mike*

80% Project Grade
20% AI
A – General uses MiniMax algorithm to make decisions that further the cause of the Army
B – General makes relatively intelligent decisions
C – General that makes some sort of decision

*Andrew*

80% Project Grade
20% Formation Tools and Graphics Implementation:
A – C#-based graphical formations editor, XML definition files for formations, 3D in-game graphics
B – 3D in-game graphics, some form of formation definitions
C – Some sort of representation of the map

*Yee Cheng*

80% Project Grade
20% AI
A – Captain/Colonels capable of making intelligent decisions using an MDP algorithm
B – Captain/Colonel capable of making smart decisions
C – Captain/Colonel makes some sort of decision

## 5.0 Mentor Selection

Alex Rice has agreed to be our mentor.

## A. Appendix : Story

Ares, God of War, walked into the local Blockbuster video rental store looking for Lion King II: Simba’s Pride. Being a God of War is a tough job and occasionally he needs to watch cheesy Disney movies to get by. After waking around for 15 minutes (No self respecting God would ask for help), he finally finds the video.

But alas, it has been checked out.

“NOOOOOOOOOOOOOOOOOOOOOOOOOOOO,” shouted Ares. Windows shattered and car alarms went off in all directions.

But as he turned around, he saw her. The most beautiful creature to walk the earth. He was instantly in love. Beautiful flowing blonde hair, silk like skin, sapphire eyes, and the most luscious lips he’d ever seen. And to top it all off, she had Lion King 2 in her hands. Obviously a woman who had taste.

He knew he had to act before he lost the love of her life. He walks up to beautiful dame and said in his deepest voice “Was your father a thief? Because he stole the stars and put them in your eyes.”

She turned and looked straight at Ares. A tall, muscular, handsome man. He looked like he had high earning potential. He probably drove a nice car. Plus, he was interested in her. “Why not?” she thought.

“Hello there,” she said in her sultriest voice, “I’m Katie.”

Ares knew immediately that she was the one he’d been waiting for since the dawn of time. “Are you doing anything later? I’ve, like, totally been wanting to watch Lion King II: Simba’s Pride.” And so together they went to her place and watched Lion King II: Simba’s Pride.

Years passed and Ares was the happiest God on Mount Olympus.

And then…It happened…

He came home early one night from a business trip to Dubai. As he was about to open the door, two guys approached him and said “Are you here for the gang bang too?”

Ares quickly realized what was happening. In his infinite rage, he drew his fiery bronze spear and impaled the dudes. Ares realized what had to be done: he ran from the house through the neighborhood slaughtering every man, woman and child.

But soon he realized the inefficiency of killing the humans one-by-one. “There are 6 billion mortal humans on this planet,” he thought, “And my with my current business commitments, I can’t possibly kill them all.” So Ares unleashed the dogs of war and soon the world’s nations were plunged in to chaos. For the first time since that day at the video store, Ares again felt true happiness.

Tired from a long day of vengeance, the blood-soaked Ares returned home to confront Katie. As Ares barged into the kitchen, sword-in-hand, he heard her sweet voice call out: “Honey, the pancakes are ready!”

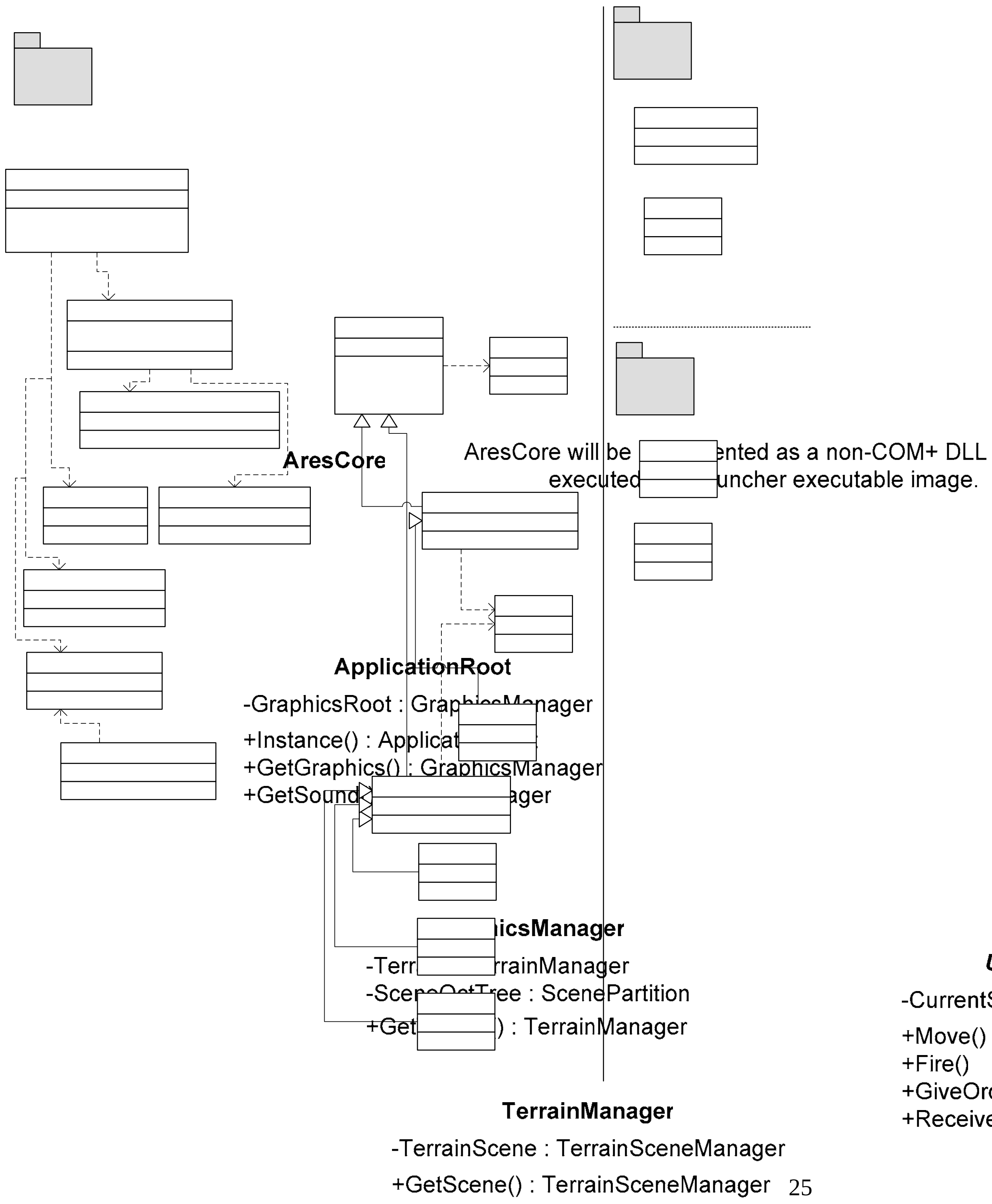
“Pancakes?” thought Ares “And all along I thought they said “gangbang”. Oops, my bad… er… too late. Mortals must perish.”

“Story in a game is like a story in a porn movie. It's expected to be there, but it's not that important.”
– *John Carmack*

« Je ne comprend pas du tout cette jeu vidéo. »
– *Napoleon Bonaparte*

**B. Appendix: Class Hierarchy (UML)**

See implementation section for information flow and calling information.


AresCore
AresCore will be ented as a non-COM+ DLL
executed uncher executable image.
ApplicationRoot
-GraphicsRoot : Graph nager
+Instance() : Applicat
+GetGraphics() : GraphicsManager
+GetSound ager
icsManager
-Terr rrainManager
-SceneOctTree : ScenePartition
+Get ) : TerrainManager
TerrainManager
-TerrainScene : TerrainSceneManager
+GetScene() : TerrainSceneManager
-Current
+Move()
+Fire()
+GiveOr
+Receive

## D. Appendix: Comparisons to similar games

| Game | Similarities | Differences |
|---|---|---|
| The Ancient Art of War | Warfare simulation including troop movements and skirmishes. | Units are homogenous not “rocks-paper-scissors”, gameboard is much much larger, Ares graphics are way cooler |
| Ogre Battle | Multiple levels of strategy/tactics in both close-up fighting of battles and in troop movements. Armies are very big, but individual units have names/attributes. Units have leaders. | Ares has a universal times scale, i.e. battles and troop movements are occurring simultaneously. Ares also features multiple tiers of leadership hierarchy. Ares also has command lag to simulate the experience of managing an army. |
| Command and Conquer / Starcraft / Age of Empires | Gameplay happens in real-time like an RTS. Ares has other RTS elements like line of sight | Ares units are hierarchical; a chain of command determines unit actions. Ares also does not involve resource gathering or upgrading units |
| Warcraft: DotA Mod | RTS elements, player may have to defend multiple channels simultaneously to prevent an enemy from “breaking through” and attacking the base/general | Gameplay not focused on a “hero” character but on the collaborative efforts of lots of little characters. |
| Battlefield 1942 / Counterstrike | Modern warfare setting, simulation of the combat experience | Ares is not an FPS. Battlefield has around 10 units or each side; Ares has 1000. For that reason, FPS elements like ray-tracing on bullets is not possible in Ares. |
| Risk / Chess | Broad strategy is important; Ares and Risk are both about dominating land area. At the general level in Ares, the map is divided into an 8x8 grid like a chess board. | Ares features both micro- and macro- elements of strategy. Broad decisions are not made rapid-fire; rather battles must play out before colonels and generals decide the next action |
| Tactics Ogre / Final Fantasy Tactics / Advance Wars | Broad war strategy actions takes place on a game board | Ares gameplay is not turn-based. |
| Lemmings | Lots of little dudes running around who act autonomously until provoked to behave differently. | Ares agents are smarter than mindless lemmings (we hope) |

## D. Appendix: Changes

Version 1.0

We added everything

Version 2.0

Vision added
Information/Command lag added
User interactions split off from requirements to be viewed easily
Rubrics changed.  Added ABC specs
Colonel is now a MDP based unit
Sergeant renamed to Captain
Added a second beta tester
Classes slightly modified to reflect changes in Colonel
Class UML diagram
Software changes (- ODE, - Visual Studio Team Foundation Server, + subversion)

Version 3.0

1.1.1 Gameplay added.
State Information (2.2.2), Action/Order (2.2.3), Generals’ Decision Process (2.2.4), Colonel and Captains’ Decision Process (2.2.5), Grunts’ Order-Following (2.2.6) added.

Version 4.0

1.0.2 Killer app added
Appendix C added – Comparisons between similar games
Formations section added to gameplay
Rubric revised