# HGAM 6210: Special Topics in Software Development

## Assignment 1: Basic Ant Colony AI

SMU/Guildhall, 2016 Fall B, Module 7, Cohort 25

Instructor: Squirrel Eiserloh

Requirements

Write a DLL plugin which provides the logic for an ant colony that satisfies the following criteria:

1. **Game Rules Overview**
   1. Multiple ant colonies battle to the death in a square arena of size 40x40 to 200x200 consisting of **stone** (a.k.a. rock), **dirt**, **food**, and **open** spaces.
   2. Each map has distinctive characteristics and composition; for example, one map may have a few large clumps of dirt with plentiful food and no rocks; another may have many small rock outcroppings with scarce initial food but steady food regeneration, etc.
   3. Each colony has a mix of 4 types of ants: a unique **queen** (who eats food and births new ants); **workers** (who can carry food back to the queen), **soldiers** (who fight better), and **scouts** (who see farther and dig through dirt faster).
   4. Each colony begins with a queen ant, 3 worker ants, 3 scout ants, 3 soldier ants, and 2000 nutrient reserves.
   5. Each turn, a colony’s nutrient reserves are reduced by an upkeep cost of 1 unit per living worker, 2 units per living soldier, and 8 units for its queen. Scout ants require 0 upkeep (but still cost 500 nutrients to birth, the same as any other ant).
   6. Each turn in which a colony cannot pay its full nutrient upkeep cost, one of its ants will starve to death. This penalty is waived on any turn in which the colony had at least one ant commit suicide.
      1. The algorithm for who is chosen to starve (if no suicide was ordered) is:
         1. A random soldier, if any soldiers exist; if not,
         2. A random scout, if any scouts exist; if not,
         3. A random worker, if any workers exit; if not,
         4. The queen, in which case your colony has been defeated and is now dead.
   7. Each food morsel eaten by the queen (either by her moving onto a food square or by a worker dropping a food morsel on her) increases the colony’s nutrient reserves by 1000 units.
   8. Queens may birth new ants, at a cost of 500 units from the colony’s nutrient reserves.
   9. Each ant may move 1 square in a cardinal (N/S/E/W) direction per turn, or perform some other action appropriate to its type.
   10. Multiple ants from the same colony can occupy the same square; ants from different colonies cannot.
   11. Ants from all colonies move and act simultaneously, after which all colonies are informed of the results of the total combined move.
   12. A combat occurs whenever ants of different colonies either (a) attempt to move through each other, or (b) end up in the same square.
   13. Each combat is resolved in steps; in each step, the strongest ant from each colony fights. The ant(s) with the lowest strength in the matchup die. Steps are repeated until the combat is fully resolved (either all ants in the combat have died or only one colony’s non-queen ants remain).
   14. Workers and scouts have a combat strength of 1; soldiers have a strength of 2; queens have a strength of 0.
   15. Any time a queen would normally die as a result of a combat, the queen does not die; instead, her colony loses 5 nutrients per combat strength of all hostile ants that fought her (which do not die).
   16. If a worker ant is carrying food when it dies (for any reason), it will drop its food if possible, as if it had automatically attempted to execute the DROP\_FOOD command. If unsuccessful, the food it was carrying is lost.
   17. If two or more queens move in such a way as to collide with each other, they all fail to move (and no combat between them ensues). Likewise, if a queen attempts to enter another queen’s square (who is not moving), its movement fails.
   18. An ant can dig through dirt by moving into it twice; the first move is blocked but removes the dirt. If multiple ants attempt to dig the same square of dirt at once, it is random which ant will actually dig; that ant (and the dig) is resolved first, with the others moving after the dirt is removed (in the same turn). Note that Scouts are unaffected by this rule, and move through dirt squares (digging automatically as they go) as if they were open squares.
   19. Ants have a limited visibility range: 4 squares away (Manhattan distance) for workers and soldiers, 8 for scouts, and 6 for the queen. Squares outside the visible range of a colony’s ants are obscured to the colony.
   20. Queens are slow and cannot move or act back to back in consecutive turns; a queen must rest (HOLD) at least every other turn.
   21. Each battle rages until only one ant colony remains alive. After 1000 turns the game enters Sudden Death mode, and no more new food will spawn. Since queens themselves have a nutrient upkeep cost, this means eventually ALL colonies will starve to death.
   22. A “war” is when the same colonies face each other in a series of 7 group battles, each on a different map (usually with very different map characteristics). The winner of each battle receives 2 points; the runner-up, 1 point. At the end of the series, the two highest-scoring colonies will face off in a one-on-one championship series of 3 battles. (Ties are resolved by a limited 3-battle series between any tied players.) The final 3 battle maps for the 1-on-1 championship series are chosen by the first and second-ranked players in alternating order (highest-ranked player picks map #1, then lowest map #2, then highest map #3).
2. **DLL-Arena communications**
   1. At startup, the arena program scans the Players/ folder for DLLs. Each DLL found is assumed to be a player (ant colony) AI plugin module and is loaded into memory via LoadLibrary().
   2. **GetInterfaceVersion**() is called on each DLL; the version reported by the DLL must match the arena’s current version.
   3. **GetPlayerName**() is called on each DLL; the DLL returns the player name (e.g. “Hordes of Doom”).
   4. **GetAuthorName**() is called on each DLL; the DLL returns the DLL’s author’s name (e.g. “Squirrel Eiserloh”).
   5. A map is randomly generated based on the parameters specified by the first (uncommented) <Arena> element found in the Data/ArenaDefinitions.xml file.
   6. **PrepareForBattle**() is called on each DLL, providing basic information about the map/battle (map size, number of players, and starting position for this player) in an **ArenaInfo** structure (defined in CommonInterface.hpp).
   7. A dedicated thread for each player/DLL is created within the arena program.
   8. **BeginWork**() is called on each DLL from within that player’s dedicated thread. The DLL should do most of its work in this thread throughout the course of the battle. It should not return from this call until the battle has concluded.
   9. Every turn, the following items happen in the arena program’s main loop:
      1. **SendUpdate**() is called on each DLL from the arena’s main thread. Included in this call is an up-to-date report on the colony’s population and nutrient reserves, the number of turns remaining before the match will enter sudden death mode, and an updated view of the map squares that can currently be seen by any of the colony’s ants (including any enemy ants within those squares). The DLL should make a (thread-safe) copy of these results and then return as quickly as possible. Any DLL taking more than 1 millisecond to return from this function call (in Release) will be penalized (or disqualified).
      2. The main thread sleeps for at least 20 milliseconds have passed since the last SendUpdate() was called.
      3. **FetchNewOrders**() is called on each DLL from the arena’s main thread. The DLL should fill in the data structure provided for submitting turn orders and then return as quickly as possible. Any DLL taking more than 1 millisecond to return from this function call (in a Release build) will be penalized (or disqualified).
      4. All players’ turn orders are resolved and adjudicated simultaneously, and the simulation state is updated.
      5. Repeat.
   10. **EndWork**() is called on each DLL from the arena’s main thread when that player has either been eliminated from the battle or the battle has ended. *Note: not currently implemented.*
3. **AI minimum requirements**
   1. Each player’s AI must pass each of the following fitness tests:
      1. Your AI must be able to move ants around semi-intelligently in the map:
         1. Scouts and soldiers must (at least) wander around randomly, or search unseen areas of the map;
         2. Ants must not attempt to move into stone squares;
         3. Workers must take any food they find and try to bring it back to drop it on the queen.
         4. Plausible orders of some kind must be successfully submitted for every living ant each time FetchNewOrders is called.
      2. Your AI is functioning in a thread-safe manner, particularly with regard to FetchNewOrders and SendUpdate.
      3. Your AI must not crash.
   2. Your colony MUST be able to survive by itself (no enemies) for 1000 turns on the “Fertile Plain” map type (#4 in ArenaDefinitions.xml).
4. **The nitty gritty details:**
   1. Each map can have distinct characteristics, including: size (40-200), percentage of squares which are rock (0-25%), percentage of dirt squares (0-80%), percentage of food squares (0-15%), stone/rock “clumpiness” (1-10 max squares per patch), dirt “clumpiness” (1-50 max squares per patch), and food spawn period (convert a random OPEN square into FOOD every N turns for the first 1000 turns, from 2 to 9999, until Sudden Death is reached an no more food is spawned).
   2. Each map cell will be one of the following **material types**, depending on the composition of the map:
      1. **OPEN**: ants may freely enter this space by moving into it.
         1. A worker ant in an OPEN square may convert it to FOOD by successfully executing the DROP\_FOOD command.
      2. **STONE**: ants may not enter this space; attempting to move into it will fail.
         1. Squares at the outermost edge of the map are always STONE.
      3. **DIRT**: non-scouts attempting to move into this space will fail; however, this will result in it being converted to OPEN.
      4. **FOOD**: ants may freely enter this space by moving into it.
         1. A worker ant in a FOOD square may convert it to OPEN by successfully executing the TAKE\_FOOD command.
         2. A queen ant moving into a FOOD square automatically converts it to OPEN (and gains 1000 nutrients).
   3. Each colony starts with 1 queen (with 2000 nutrient reserves), 3 worker ants, 3 scout ants, and 3 soldier ants, all at the colony’s starting position.
      1. **SOLDIER** ants fight with a strength of 2, and have a visual range of 4 squares (Manhattan distance).
      2. **WORKER** ants fight with a strength of 1, and have a visual range of 4 squares (Manhattan distance).
         1. Only worker ants may execute the TAKE\_FOOD and DROP\_FOOD commands.
      3. **SCOUT** antsfight with a strength of 1, move without penalty through dirt, and have a visual range of 8 squares.
      4. The **QUEEN** ant is unique per colony. She has a visual range of 6 squares, and a combat strength of 0.
         1. Only the queen may execute the CREATE\_WORKER, CREATE\_SCOUT, and CREATE\_SOLDIER commands.
         2. If a queen moves into a FOOD square, she eats the food (gaining 1000 nutrients) and converts it to OPEN.
         3. ALL commands except HOLD will fail for a queen who is exhausted; that is, who acted in the previous turn. Queens therefore may effectively only act every other turn at most (e.g. act, rest, act, rest).
         4. If for any reason your colony has no queen – for example, due to her suicide – you immediately lose, and all your ants instantly die (reported for each ant as REPORT\_STARVED\_TO\_DEATH).
   4. Each colony moves in turn; a move is a set of commands, with up to one command per living ant in the colony (including the queen). Legal commands include:
      1. **ORDER\_HOLD**: The ant will attempt to hold its current position (default action).
      2. **ORDER\_MOVE\_EAST**: The ant will attempt to move one square east / right / +X.
      3. **ORDER\_MOVE\_NORTH**: The ant will attempt to move one square north / up / -Y.
      4. **ORDER\_MOVE\_WEST**: The ant will attempt to move one square west / left / -X.
      5. **ORDER\_MOVE\_SOUTH**: The ant will attempt to move one square south / down / +Y.
      6. **ORDER\_TAKE\_FOOD** (worker ants only): The ant will attempt to pick up a morsel of food from its current square.
         1. This command only succeeds if the ant is in a square of type FOOD, and is not already carrying food.
         2. If successful, this command converts the ant’s square from FOOD to OPEN; the ant is now carrying food.
      7. **ORDER\_DROP\_FOOD** (worker ants only): The ant will attempt to drop any food it is carrying in its current square.
         1. This command only succeeds if the ant is in a square of type OPEN, and is carrying food.
         2. If successful, this converts the ant’s square from OPEN to FOOD; the ant is no longer carrying food.
         3. If, however, the ant’s square has a queen in it, the food morsel will instead be eaten by the queen (and the square will remain OPEN). The queen’s colony will have its nutrient reserves increased by 1000 units.
      8. **ORDER\_CREATE\_WORKER** (queen only): The queen will attempt to birth a new worker ant in her current square.
         1. This action requires (and consumes) 500 nutrients from the colony’s reserves.
      9. **ORDER\_CREATE\_SOLDIER** (queen only): The queen will attempt to birth a new soldier ant in her current square.
         1. This action requires (and consumes) 500 nutrients from the colony’s reserves.
      10. **ORDER\_CREATE\_SCOUT** (queen only): The queen will attempt to birth a new scout ant in her current square.
          1. This action requires (and consumes) 500 nutrients from the colony’s reserves.
      11. **ORDER\_SUICIDE**: The ant will end its own life. *Note: if the colony is starving (not enough nutrients to pay its upkeep), issuing a suicide order for at least 1 ant will prevent any ants from starving to death this turn.*
   5. Each turn, the server will provide an information update via **SendUpdate()** to the colony which consists of:
      1. An updated view of the material contents of all grid squares within visual range of any ant in the colony, as well as any enemy ants sighted within visible range (including ant types and, for workers, whether they carry a food morsel);
      2. An updated count of the colony’s current nutrient reserves;
      3. The current nutrient upkeep cost this turn (already deducted from the current nutrient reserve reported);
      4. The current turn number;
      5. The number of workers, soldiers, and scouts currently present in the colony’s population;
      6. At least one command result code for each command issued in the colony’s most recent turn orders. Result codes include:
         1. **REPORT\_HOLD\_SUCCESSFUL**: The ant’s HOLD command executed successfully and without incident. Reported for each ant that held, whether explicitly or implicitly (since HOLD is the default action if none is specified).
         2. **REPORT\_MOVE\_SUCCESSFUL**: The ant’s MOVE\_XXX command executed successfully and without incident.
         3. **REPORT\_DIG\_SUCCESSFUL**: The ant’s MOVE\_XXX command was blocked by DIRT, which has now been converted (dug) to OPEN.
         4. **REPORT\_ATTACK\_SUCCESSFUL**: unused (legacy).
         5. **REPORT\_TAKE\_SUCCESSFUL**: The worker ant’s TAKE\_FOOD command executed successfully; the worker is now carrying a food morsel, and its square was converted from FOOD to OPEN.
         6. **REPORT\_DROP\_SUCCESSFUL**: The worker ant’s DROP\_FOOD command executed successfully; the worker is now no longer carrying a food morsel, and either its queen was fed or its square was converted from OPEN to FOOD.
         7. **REPORT\_CREATE\_SUCCESSFUL**: The queen ant’s CREATE\_XXX command executed successfully. A new ant was created, and will be included in this result set with a WAS\_CREATED result code.
         8. **REPORT\_SUICIDE\_SUCCESSFUL**: The ant’s SUICIDE command executed successfully. The ant is now dead.
         9. **REPORT\_WAS\_KILLED\_IN\_COMBAT**: This ant was killed in combat this turn.
         10. **REPORT\_STARVED\_TO\_DEATH**: This ant was randomly selected to die of starvation due to insufficient nutrient reserves to sustain the colony’s upkeep requirements. The ant is now dead. In order to prevent this, one of your ants may voluntarily suicide themselves. Or you could, you know, try not to run out of nutrients so much. *Note: all of your ants will receive this report (and have died) if your colony ever has no queen ant(s) alive.*
         11. **REPORT\_QUEEN\_WAS\_ASSAULTED**: The queen ant was attacked by one or more enemy ants this turn, resulting in a loss of nutrients equal to the (some amount times) attackers’ total strength.
         12. **REPORT\_QUEEN\_WAS\_FED**: The queen ant was fed a food morsel by a worker ant who dropped it onto her, or by walking onto a FOOD square herself. May receive more than one of these per queen per update.
         13. **REPORT\_WAS\_CREATED**: This ant was created new this turn, either at the start of the battle or as a result of a queen’s CREATE\_XXX command.
         14. **REPORT\_ERROR\_BAD\_ANT\_ID**: The ant ID# specified for the command is invalid, or is the ID of an ant not belonging to your colony, or the ant is dead.
         15. **REPORT\_ERROR \_ANT\_MOVED\_TWICE**: Two or more orders were issued for the same ant.
         16. **REPORT\_ERROR \_WRONG\_ANT\_TYPE**: The ant ID# specified was for an ant whose type does not support the requested command (e.g. non-worker ants cannot take or drop food; non-queens cannot create ants).
         17. **REPORT\_ERROR \_BLOCKED\_BY\_ROCK**: The ant’s MOVE\_XXX command was blocked by a STONE (a.k.a. rock) square.
         18. **REPORT\_ERROR \_OUT\_OF\_BOUNDS**: The ant’s MOVE\_XXX command was not legal (attempted to move to a non-adjacent or out-of-bounds square).
         19. **REPORT\_ERROR \_NO\_FOOD\_PRESENT**: The worker ant’s TAKE\_FOOD command failed to execute due to his square being of a material type other than FOOD.
         20. **REPORT\_ERROR \_ALREADY\_CARRYING\_FOOD**: The worker ant’s TAKE\_FOOD command failed to execute since he is already carrying a food morsel.
         21. **REPORT\_ERROR \_SQUARE\_NOT\_EMPTY**: The worker ant’s DROP\_FOOD command failed to execute due to his square being of any time other than OPEN.
         22. **REPORT\_ERROR \_NOT\_CARRYING\_FOOD**: The worker ant’s DROP\_FOOD command failed to execute due to the fact that he is not carrying a food morsel.
         23. **REPORT\_ERROR \_INSUFFICIENT\_FOOD**: The queen ant’s CREATE\_XXX command failed since the colony lacks the required amount of nutrient reserves.
         24. **REPORT\_ERROR \_QUEEN\_WAS\_EXHAUSTED**: The queen ant attempted a move other than HOLD two consecutive turns in a row.
         25. **REPORT\_ERROR \_MAXIMUM\_POPULATION\_REACHED**: The queen ant’s CREATE\_XXX command failed since the colony has already reached its maximum population of 100 ants.

Grading

This assignment is worth 100 points, divided as follows:

* **(20 points)** DLL correctly implements all required export functions.
* **(20 points)** Code is thread-safe, especially in regards to FetchNewOrders() and SendUpdate().
* **(20 points)** Calls to FetchNewOrders() and SendUpdate() always return within < 1ms (in Release).
* **(40 points)** Basic plausible ant colony AI is present (per section 3 above).
* **(-20 points)** Up to 20 points off if your code hangs or crashes, depending on severity, frequency, and difficulty to work around
* **(-20 points)** Up to 20 points off if your code violates any of the letter or spirit of the law of the Arena.