Requirements Document

Draft 1.0

29.02.16

Nicolas

George Y

Haden

Hao Yan

George T

# Domain Requirements

|  |  |  |
| --- | --- | --- |
| Requirement | Rational | Source quote |
| The program shall be supplied with a “novel” ant brain, that shall be compatible with other ant simulations | See quote | *A novel, proof-of-concept ant-brain for the competition taking place at the end of the course between all the teams* |

# Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| Requirement | Rational | Source quote |
| Game shall be two player, and strategic | See quote | *The game proposed by the customer is a competitive two-player strategy game* |
| Game shall allow brains to be uploaded into a world | See quote | *To run the game, the brains are uploaded into an ant world with two ant colonies, one for each player* |
| Game shall support a “tournament” mode, allowing many players to upload their brains | See quote | *A program that allows to play tournaments, where an arbitrary number of players can upload ant-brains, who are all paired up to play against each other. The overall tournament winner is the ant brain that wins the most individual games* |
| Program will be written in Java | This is the language most commonly used by our development team. | *You are free to choose your implementation language. Good choices would be Java, OCaml, F#, Scala, C#, Python, or Javascript if you want to run your game in the browser directly* |

# Functional Requirements

## Program requirements

|  |  |  |
| --- | --- | --- |
| Requirement | Rational | Source quote |
| Program shall check that a supplied ant brain is syntactically correct | Since this program will be using brains from many different teams, they must all adhere to a set of standards | *A program that checks if an ant-brain supplied by a player is syntactically well-formed* |
| Program shall check that any given ant world is syntactically correct | ‘ ‘ | *A program that checks if a given description of an ant world is syntactically well-formed and meets the requirements for ant worlds used in tournaments* |
| Program shall “visualise” the world | This will make de-bugging and program usage easier for the end user and development team | *A program that can visualise a given ant world* |
| Program shall be able to randomly generate correct world | See quote | *A program that allows the generation of random but well-formed ant worlds* |

## Game requirements

|  |  |  |
| --- | --- | --- |
| Requirement | Rationale | Source quote |
| Game shall contain two ant hills, one per player | See quote | *two species of ants are placed in a random world containing two anthills* |
| Game shall run for 300,000 rounds | See quote | *The match is won by the species with the most food in its anthill at the end of 300000 rounds* |
| Game shall be won by the species with the most food | See quote | *“ “* |

## World requirements

|  |  |  |
| --- | --- | --- |
| Requirement | Rational | Source quote |
| World shall contain “some” food sources, and “several” obstacles | See quote | *the two species of ants are placed in a random world containing two anthills, some food sources, and several obstacles* |
| World shall be made up of hexagonal cells | See quote | *The world on which the game is played is a hexagonal grid (just for fun)* |
| A cell in the world shall be defined as a position [ pos = (x, y) ] | See quote | *we use the type pos as an abbreviation for pairs of integers* |
| Adjacent cells shall be labelled in a clockwise order, starting at east, with integers 0-5, and this is given the type dir | See quote | *On a hexagonal grid, each cell is adjacent to six other cells. We number these in clockwise order, from 0 (east) to 5 (north-east)* |

Simulation shall implement the function Step, as given in the specification document (kinetics)

Upon game initialisation, each cell within an anthill (7 cells) is populated with an ant of the correct colour, and identities are then assigned top-to-bottom, and left-to-right. All ants initially face direction 0 (due north).

At the end of the 300,000 rounds, the anthill with the most food within it wins. Food carried by ants does NOT count for final scoring

There must be at least one empty cell between “adjacent” non-food items

## Ant API

Do we need to spell colour the American way? (color)

* Shall be able to read/write variables of enum left\_or\_right [left, right]
* Shall contain a variable of enum type Color [red, black]
* Shall contain an enum type Marker[0..5]
* Shall contain an enum type Sense[10]
* Shall contain between [0...9999] states
* Shall contain an enum States[8]

*“each ant brain is just an array of instructions, indexed by states”*

|  |  |  |
| --- | --- | --- |
| Variable | Description | Get/Set |
| Integer id | Ant ID | Both |
| Color color | Ant colour | Get |
| Integer state [0..9999] | Current state of the ant brain | Both |
| Integer resting | Number of rounds ant has rested for so far | Both |
| Integer direction [0..5] | Current direction of ant | Both |
| Boolean has\_food | Is this ant holding a food particle? | Both |

Ant API shall contain at a minimum the following states and functions

|  |  |
| --- | --- |
| State/Function | Description |
| sense (sensedir st1 st2 cond) | Go to state st1 if cond holds in sensedir; and to state st2 otherwise. |
| mark( i st) | Set mark i in current cell and go to st. |
| unmark ( i st) | Clear mark i in current cell and go to st. |
| pickUp (st1 st2) | Pick up food from current cell and go to st1; go to st2 if there is no food in the current cell. |
| drop (st) | Drop food in current cell and go to st. |
| turn (lr st) | Turn left or right and go to st. |
| move (st1 st2) | Move forward and go to st1; go to st2 if the cell ahead is blocked. |
| flip (p st1 st2) | Choose a random number x from 0 to p-1; go to st1 if x=0 and st2 otherwise. |
| other\_color() : Color | Returns the opposite colour |
| Set\_marker\_at(pos p, color c, marker i) | Set a marker of color c at position p |
| Clear\_marker\_at(pos p, color c, marker i) | Clears marker I at position p |
| Check\_marker\_at(pos p, color c) : Bool | True if marker I of color c is at position p |
| Check\_any\_marker\_at(pos p, color c) : Bool | True if any marker of color c is in cell p |

Ant chemistry shall consist of 6 chemical markers, between [0..5], which remain until cleared

The sense function will operate over ONE of the following conditions:

1. Friend
2. Foe
3. FriendWithFood
4. FoeWithFood
5. Food
6. Rock
7. Marker(marker)
8. FoeMarker
9. Home
10. FoeHome

The function cell\_matches ( as defined in the specification) will check conditions from the sense enum at given positions

A given ant brain file shall take the following format:

* Each line represents a state (as an integer)
* A file may not exceed 10,000 lines
* Each line consists of a sequence of whitespace-separated tokens, followed (optionally) by a comment beginning with a semicolon and extending to the end of the line
* Tokens are keywords or integers (see specification for full list)

If an ant is ever adjacent to 5 OR 6 ants of the opposite colour, it dies and turns into 3 particles of food, using the following functions:

* Adjacent\_ants(pos p, color c) : int
* Check\_for\_surrounded\_ant\_at(pos p)
* Check\_for\_surrounded\_ants(pos p)

The Flip instruction will return the same random numbers, and follow the mathematical specification given in the specification document.

## World API

* Shall contain the enum type Sense\_dir [ahead, left\_ahead, right\_ahead, current]

Each cell in the world shall be characterised by the following variables:

|  |  |
| --- | --- |
| Variable | Description |
| Boolean rocky | Is this cell impassable? |
| Integer food [>= 0] | Quantity of food on this cell |
| Boolean occupied | Is this cell occupied by an ant? |
| Marker black | Black ant marker details |
| Marker red | Red ant marker details |

A cell shall contain at minimum the following functions:

|  |  |
| --- | --- |
| Function | Description |
| some\_ant\_is\_at(pos p) : bool | p = position of cell to investigate  returns if the cell is occupied |
| ant\_at(pos p) : ant | p = position of ant to return  returns the ant at that cell |
| set\_ant\_at(pos p, ant a) | p = position of cell to place ant in  a = ant to place in cell |
| clear\_ant\_at(pos p) | p = position of cell to clear ant from  \*can only be called on cells that are occupied\* |
| ant\_is\_alive(int id) : bool | ID = id of ant to inspect  Returns if this ant ID exists in the world |
| find\_ant(int id) : pos | ID = id of ant to find  Returns the position of this ant |
| kill\_ant\_at(pos p) | Calls clear\_ant\_at(p) |
| food\_at(pos p) : integer | p = position of cell to check for food  returns amount of food on that cell |
| set\_food\_at(pos p, int f) | p = position of cell to place food  f = quantity of food to place |
| anthill\_at(pos p, color c) : bool | p = position of anthill  c = colour of anthill  returns true is the cell specified is an anthill belonging to colour c |

World API shall contain at a minimum the following functions

|  |  |
| --- | --- |
| Function | Description |
| adjacent\_cell(p,d) | p = position  d = direction |
| turn(left\_or\_right ld, dir d) : dir | ld = direction of turn  d = current direction of ant  returns new direction of ant |
| sensed\_cell(pos p, dir d, sense\_dir sd) : pos | p = curent position  d = curent direction  sd = sense direction being used  returns the position of the sensed cell |

World files shall be formatted in the following way:

* X dimension (integer)
* Y Dimension (Integer)
* [1..Y] Lines of the following characters (they may not overlap):
  + # rocky cells
  + . clear cell
  + + red anthill
  + – black anthill
  + [1..9] food particles

Contest world files shall contain:

* The dimensions of the world are always 150 × 150 cells.
* The cells on the perimeter are always rocky.
* Every world contains exactly the same elements, of particular shapes: 2 anthills, 14 rocks, and 11 blobs of food. The anthills, in particular, are hexagons with sides of length 7. Also, a food blob is always a 5-by-5 rectangle, with each cell containing 5 food particles.
* The positions and orientations of the elements are chosen randomly, subject to the constraint that there is always at least one empty cell between non-food elements. Also, no elements overlap. (The anthill elements are 6-ways-symmetric, so their orientation actually does not matter. All ants are initially facing in direction 0.)