Phase 1- General cursor movement

1. If cursor not moving
   1. If scroll mode not active
      1. If input is C
         1. If over a unit that can act
            1. Calculate move ranges, draw them
            2. Game\_temp.range\_type = 1
            3. Phase 2
         2. If over a property that can build
            1. Draw build window
            2. Phase
         3. Else
            1. Open Menu
            2. Phase
      2. Else If input is B
         1. If over a unit or structure with range
            1. Calculate attack ranges, draw them
            2. Game\_temp.range\_type = 2
            3. Phase 2
         2. Else
            1. Scroll Mode (game\_temp var)
   2. Else
      1. If no longer holding B
         1. Deactivate scroll mode

Phase 2 – Ranges

1. If range\_type = 1
   1. If cursor not moving
      1. If Input is C
         1. If valid spot
            1. Determine commands
            2. Draw window commands
            3. Phase 3
         2. Else Buzzer
      2. If input is B
         1. Phase 1
         2. Delete ranges
2. If range\_type = 2
   1. If No longer holding B
      1. Phase 1
      2. Delete ranges

Phase 3 – Choosing commands

1. If command window is open
   1. Update window
   2. If Input is C
      1. Process command
      2. Determine if command requires specifying a location
      3. If so, deactivate window
   3. If input is B
      1. Phase 2
      2. Delete window
2. Else
   1. If cursor not moving
      1. If input is C
         1. If valid
            1. Process decision
            2. If further window processing required

Open command window again

* + - * 1. Else Phase 4
      1. Else buzzer
    1. If input is B
       1. Back up to window
       2. Delete drawn ranges, revert changes

Phase 4

Current Class Variables:

Unit Holders

~~Selected unit~~ The unit in question that is being commanded right now

~~First drop unit~~  First unit to be assigned a drop spot

~~Second drop unit~~ Second unit to be assigned a drop spot

~~Unit to join with~~ Unit that will be joined with @selected\_unit

~~Carrying unit~~  Unit that will load @selected\_unit inside it

~~Target unit~~ Unit that will be attacked by @selected\_unit

Array Collection

Passed positions [x,y]’s to make the arrow path when deciding move path

~~Per cost~~ ABSOLUTELY USELESS

~~path\_cost~~ ABSOLUTELY USELESS

Variables:

Unit index Number to remember what unit to look at next in army’s unit list

~~Drop dir 1~~ [Direction] the first dropped unit will move in from its carrier

~~Drop dir 2~~ [Direction] the second dropped unit will move in from its carrier

Player turn Holds Army object of the current player’s turn

Phase Keeps track of what update phase to execute

Preturn Keeps track of what preturn update phase to execute

Wait A generic counter to delay code

~~Action~~  String of the command given to @selected\_unit

Coordinates

~~Store cursor loc~~ ABSOLUTELY USELESS

~~Decided spot x~~  x-coordinate of tile player wishes to move @selected\_unit to

~~Decided spot y~~  y-coordinate of tile player wishes to move @selected\_unit to

~~Current cursor x~~ Stored cursor location before player input moved it. Used for updating the arrow path drawing. Most likely useless now.

~~Current cursor y~~

~~Starting position~~ ABSOLUTELY USELESS

~~First drop spot~~ [x,y] of locations first dropped unit will be placed. Used to remove available drop spots and check if @selected\_unit can unload its second unit

Flags

~~Open menu~~  User pressed C on phase 1 to open the menu

Making repairs Deducts user funds per frame if this is true

Playing income se If the rolling cash sound is playing right now

~~Show move~~ True if displaying move ranges. Useless probably.

~~show\_ranges~~ True if displaying attack range. Useless probably.

Outside of range True if the cursor is outside the move range to draw arrow path

Collection of Tile locations

~~Ranges~~  Sprite\_Range objects covering move range and attack range

~~Locations~~ Sprite\_Range objects covering target tiles (attack or drop)

~~Zones~~  MoveTile objects of places to attack or drop units

Positions MoveTile objects of unit move and attack range

Windows:

~~Officer window~~

~~Unit command window~~

~~Build window~~

~~Menu~~

Message window

Misc

Minimap

Arrow\_path ArrowSprite objects used to draw the unit movement path

~~Arrows viewport~~ ABSOLUTELY USELESS

New Variables

range type Variable to replace show\_move and show\_ranges

Proc Current cursor command taking place (includes two Procs and wait)

Unit Replacement for @selected\_unit

Active window Replacement for all the window variables

Command Holds command information like [x,y] to move to, action, target

Player Replaces player\_turn

Positions 2 Replaces @zones

If a unit that can capture property, but is not within range of one, what should it do?

It clearly needs to be given some kind of goal, whether it be moving towards a property or attacking another unit.

Possibility #1

All properties not owned by the player are to have a unit assigned to it as a goal. Units that can capture are all considered. For each property and for each unit, get the goal value if this unit were to capture it. This can be based on influence values, the city type, if owned by enemy army, distance from the unit (closer the better), and state of the unit (full HP versus damaged). The units and values are assigned with a simple formula

A = [A, B].max

Where A is the original value and B is the new, opposing value. The best value is kept when all comparisons are done, assigning the property to the winning unit as a goal. However, this does not mean this is the only goal the unit can have—we must also check for other goals like attacking an enemy unit.

So, following the same idea, for each enemy unit and for each player unit that can attack, check for goal values, which should consider influence maps, unit influence, damage done and received, unit costs, and distance. The four best player units that can attack this enemy unit are considered, and the goals are assigned to the winner units.

Repeat this process for any other goals I can think of.

Check each unit’s goals and prioritize them based on their calculated values. With that done, we can now move the units. Moving towards the goal is given a bonus value in decision making.

**When finding a path to the object in question, give the unit 4x its move range. Also, if it cannot cross the tile, assign it double the value of the cheapest tile near it + 1.**

U X X X X X U = unit 00 01 02 03 04 05

X X X W X X X = land 01 02 03 07 05 06

X W W W W X W = water 02 05 07 15 11 07

X W W W W X C = city 03 07 15 19 17 08

X W X X W X 04 09 08 09 19 09

X X X W C X 05 06 07 15 11 10

So one question is if the goal is even worth it if a transport unit is required to get there. Thus, there must be some way I can do all of these at the same time:

1) Distance from unit to goal, adding a penalty to impassible tiles

2) Saving impassible tile types

3) Finding a reasonable path in a short amount of time

Reasonable time would be if the unit can reach the goal in 4 days or less. Any longer and a transport unit with a higher move speed would be needed. However, for the sake of simplicity, if the unit cannot reach the goal on its own in 4 days, the value of the move is highly unfavorable. If the unit finds a transport unit it can ride in, see if loading onto it would be favorable. If so, assign the unit to the transport with a value (the transport unit will sort these values if multiple units request it, and will choose the best one to try to load onto it).

So how will this path finding work?

Start at unit. Look at tiles surrounding it and evaluate them. If can move onto it, set move cost to get there in MoveTile object. If it cannot move onto the tile, record the tile ID in the MoveTile, increment a “impassible tile” counter by one, and add the total move cost by unit’s move \* 4. Repeat this process until the goal is reached.

The value of moving this unit to this goal will consider strength of unit, distance (result of total move from last MoveTile), and if transport is necessary (checks the impassible tiles). The best value will tell the Unit that its goal is to reach this property. If the amount of spaces it needs to move is greater than 4 days’ worth, it will turn on a flag indicating that a transport unit that can move faster than it may carry it to its goal (however, it does not try to prioritize finding a transport unit—if one just happens to be in range, it will consider it). If a transport unit is required, the goal is to find a transport unit that can pass the most impassible terrain types found, has high movement, and can actually carry the unit.

Note that while this goal is assigned to the unit, it is still not the “best” goal. The highest value goal the unit can be assigned will be the winner. **But could this be a problem for units that don’t have a goal assigned, but were a second consideration to reach the property?** Assume two units that can capture and one unowned property. One of those units is assigned to capture the property while the other lazes about. Now the other goals are assigned (i.e. attacking a target), and the one already assigned to capture the property is now considering to attack (higher goal value). The other unit may never reach the property, especially if it’s on an island. SO the best solution seems to be that, if a unit’s goal changes, it must tell its old goal to delete it from the list and now consider the second best unit. Repeat this process until all units are assigned a goal.

**What if the goal can’t be reached? For example, a city surrounded by pipes is just a decoy to the whole AI calculations.**

I’m going to need to know if a path truly does exist then. Upon game loading, need to collect data of what units can transport and on what tiles. For example, Infantry can be carried by landers and TCops. On first evaluated spaces, if land tile, tcop can be assumed in use and can now fly over sea tiles. If found a shoal tile, lander can be assumed in use and cross over sea tiles. If tile cannot be crossed by unit and currently not assuming a transport unit to help carry them, then do not evaluate the tile. Find path to goal and get its cost. If path not found, disregard this unit. If path found, check what transport units were needed. Signal any transport units nearby a goal to pick up this unit.

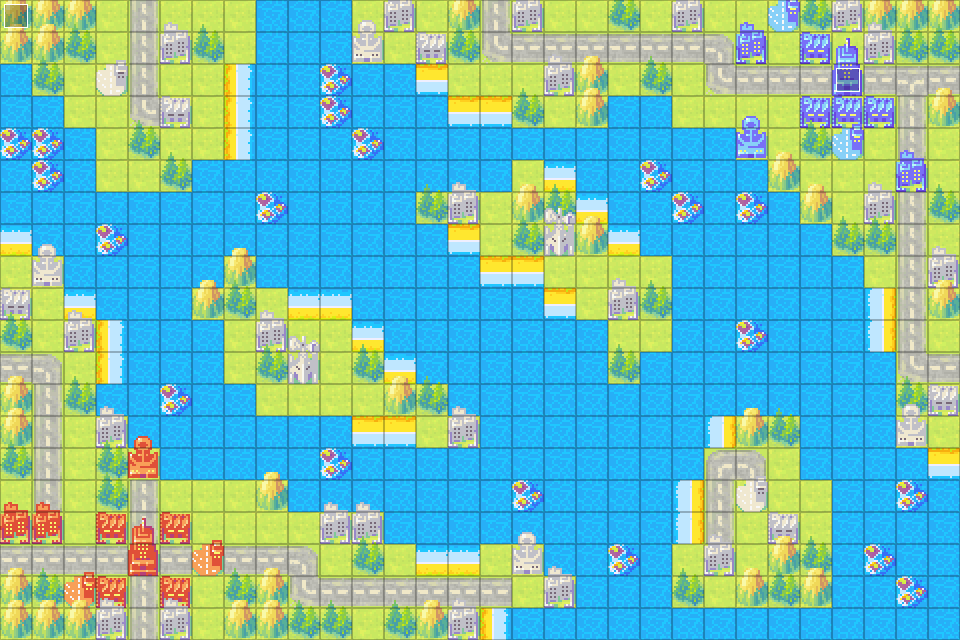
**The problem with that design though is that tiles will need to be evaluated multiple times to ensure that all potential data is transferred over. For example, if a sea tile is first evaluated as being crossed only by TCOP when there clearly is a shoal on the island that allows a lander to cross exact same tile, I would need to check all those tiles again.**

Populate the map, for each tile, the potential places a transport unit can assist with moving units. For example, if an island has a shoal, a lander can populate all the tiles on that island. However, what if the shoal is surrounded by mountains? This would only benefit infantry. As such, I would need to keep a list for different move types.

This of course means I will need to redefine how to check if a unit can be loaded onto the transport and what tiles are drop off points. Perhaps store unit IDs and just extract their move types

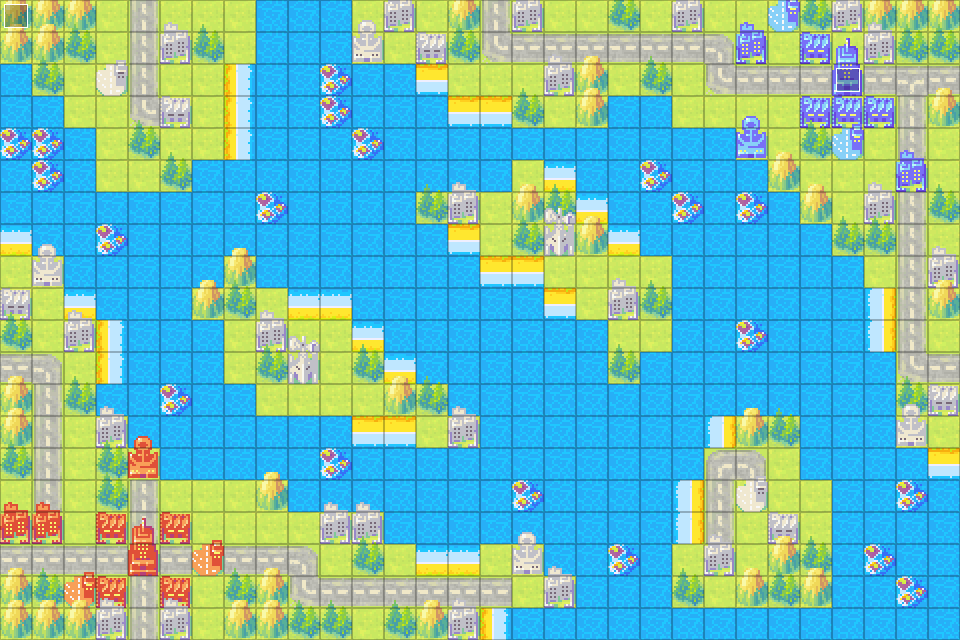
Another idea to propose. Suppose I want to establish major sections of the map. This could include islands, lakes, oceans, and pipe fortresses. To do this, I would do a basic double-pass of entire map, checking each tile and its surroundings for identification.

Here’s our map for example:



We are going to use SEA as our boundary tile. I start at the top left corner and check the tile. It’s not a SEA so let’s assign it a 1. Go to the right of it. Not a SEA so let’s assign it a number. It checks to the north and west for associative tiles. It sees the 1 on the left, so it takes the 1. This repeats for a while until we encounter our first SEA, where we will immediately assign it a 99 or something absurd—basically anything that will help distinguish it from the land tiles. We do this three times until we hit land again. Check the north and west and see there are no land tiles. We then assign it as 2 since we already used 1. This repeats until we move down to the next row at (0,1). It’s a land tile so check north and west. 1 value at north, so assign it a 1. And so on and so forth. Repeat this exact process again, keeping the assigned values, but by starting at bottom right corner checking south and east neighbors.

In the end, all SEA (and REEF) tiles will be 99.



All of these numbers will be mapped onto a 2D array.

Next step is to connect these regions together. Since all these maps have shoals and land, it’s pretty obvious that Landers and T.Copters can both help transport units across the sea. This will be indicated somewhere within, like, an Island class variable. Islands will keep track of things like does it have shoals and how many properties it has. Units can obtain what region it’s on by just checking the AI Map array at the unit’s location.

If a city on 5 broadcasts its location to an infantry on 4, the infantry knows it needs a lander or Tcopter for transport due to the Island object’s data. Infantry creates influence map for requesting transport units. If it cant find transport, just move it naturally.

If I want, I can define more regions. For example, a lake (SEA surrounded by land) could have region number of 100 (where 99 was the vast open sea). Pipes can be boundaries too as they can encapsulate land tiles. A city surrounded by pipes will be ignored since no unit can reach it.

The tricky part is figuring out what units can transport over to each region, and perhaps how far are they from each other. But I do know this will address the transport problem I’ve been having for so long.

So I have made a Region class that will hold what regions it borders. Border tiles, like SEA and PIPES, will be assigned a Region object as well. Assume we have this map with the following regions assigned.



Clearly region 1 has all the other regions as its borders and those have 1 as their border. Looking at infantry units, to get from 5 to 2 or 4, we know a Lander and a T.Copter can do the job. 5 to 3 requires a T.Copter.

If we look at this from another unit’s perspective, say a bike infantry, then we can infer that only regions 2 and 4 can be reached via a Lander. Region 3 is off limits for it. The same can be said for any other land-based unit.

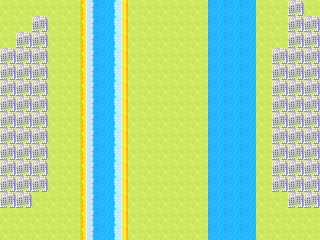
For the Lander, regions 2, 4, and 5 are reachable on its own. For other sea units, only 2 and 5.

Air units have no restrictions and can access all regions without assistance.

How can I tell the AI about this information?

First off, we need to let it know what units are capable of transporting and for what kinds. When defining region 1, we can say that Landers and Air units are capable of passing this region. For 2, 4, and 5, we can also say that Landers are capable of reaching this region too. We need to inform the Region objects that there are Shoals on the region, allowing a Lander to carry units over.

As for a situation like this:



The game doesn’t know if the lands on the far ends of the map are reachable since they don’t share the same borders. As such, we need to search the tree.

Assuming the regions are 1-5 from left to right, for an infantry unit:

First phase:

Region1.borders = [2] => Lander or Tcop

Region5.borders = [4] => Tcop

Second phase:

Region2.borders = [1,3] => Remove 1 since evaluated => [3] => Lander or Tcop

Region4.borders = [3,5] => Remove 5 since evaluated => [3] => Tcop

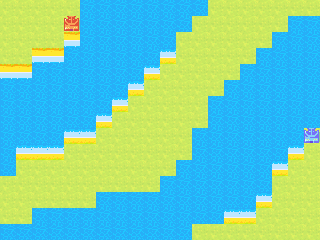
Third phase:

Converge at Region 3. Transport options are [lander, tcop] & [tcop] => Tcop

If there was an even number of regions, it would converge upon knowing that the two regions in the current phase go to each other.

If at any point either of them loses all available transports, the path is deemed impossible.

Here’s a problem to think about.



Orange Star can clearly get to the middle island via a lander since both regions have shoals. However, Blue Moon would think that it too can reach the region via a lander even though there is no shoal to be seen.

Perhaps the region in between (the SEA) should have a check for what special tiles it borders with on other regions.

Region1.border = [2 => [SHOAL,PORT]]

Region2.border = [1 => [SHOAL,PORT], 3 => [SHOAL]]

Region3.border = [2 => [SHOAL], 4 => []]

Region4.border = [3 => [], 5 => [SHOAL, PORT]]

Region5.border = [4 => [SHOAL, PORT]]

Getting a unit from region 5 to 3 would be

First Phase:

Region5.borders = [4] => SHOAL, PORT => Lander or Tcop

Region3.borders = [4] => nil => Tcop

Second Phase:  
Converges at 4 => [Lander, Tcop] & [Tcop] => Tcop

Also, I should consider that, even though only a Tcop can transfer a unit from region 1 to 5, a lander can still help me get to my goal.

Basic AI strategy: Assumes no transport units

1. Collect all units that can capture
   1. If the unit is already capturing, just let them capture.
   2. For each of the remaining capture units, check what properties are within their movement range. Store this into their AI object.
   3. Select the units that have the least amount of properties they can capture. Of those, choose the property that is furthest away from the unit and with the best value.
   4. If nothing to capture or not fit for capture, pass the units into the next block
2. Collect all units that can attack
   1. For each unit, get attack range (indirect) or move range (direct)
   2. Check each spot for an enemy unit to attack. Calculate damage, counter damage, cost of units, and units that can attack this spot too
      1. If damage to enemy \* value of enemy unit > counter damage \* value of own unit
      2. Check potential enemy units that can target this spot—see if damage is capable of destroying unit
      3. Check own units that can support the unit at this spot.
      4. If enemy units influence < player units influence
      5. Consider this move
   3. Choose the highest valued attack
3. If no more attacks can be made
   1. Check for capture units again and see what’s left to capture
   2. Move towards the areas with the highest tension while staying out of enemy range
   3. If in danger, retreat to where your units are most influential and out of enemy attack range
   4. Order of moving should be capture, direct, indirect