Tasks/Accomplishments to Date

Project 1: Implement Territory Persistence

* (Feb 10-11) Local dev machine setup.
  + Install MySQL, build a3\_wasteland schema, install extDB (as .dll install), and configure existing mission to use extDB as the persistence framework.
  + Test this configuration and validate that current build is working correctly
* (Feb 12) Establish gitHub repository for g4-gamers missions (repo name: g4-gamers-Wasteland)
* (Feb 13) Incorporate territory persistence prototype (iniDBi only) and extend to support extDB persistence
  + Mission Files modified:
    - \territory\server\monitorTerritories.sqf
    - \territory\server\\territoryPayroll.sqf
    - \territory\client\updateConnectingClients.sqf
    - \A3Wasteland\_settings\main\_config.sqf (add A3W\_territorySaving, A3W\_territoryLogging options)
    - \server\default\_config.sqf (add A3W\_territorySaving, A3W\_territoryLogging options)
  + Mission Files added:
    - \persistence\server\world\tLoad.sqf
    - \persistence\server\world\default\getTerritories.sqf
    - \persistence\server\world\default\saveTerritory.sqf
    - \persistence\server\world\extDB\saveTerritory.sqf \persistence\server\world\extDB\getTerritories.sqf
    - \persistence\server\world\tSaveInit.sqf
  + extDB modifications:
    - add Territory persistence, logging tables to a3\_wasteland\_db schema
      * update extdbModel,mwb to include new tables, relationships & indicies.
      * generate new schema creation .sql as a3wasteland\_db\_v2.04.sql
      * create schema 2.03->2.04 update sql as a3wasteland\_db\_v2.04.sql
    - update \extDB\db\_custom\a3wasteland.ini to include SQL for the following handlers:
      * newTerritory
      * getServerTerritoriesCaptureStatus
      * newTerritoryCaptureStatus
      * updateServerTerritoriesCaptureStatus
* (Feb 15) Initial testing of work so far, debugging tLoad & getTerritories scripts so that getTerritories.sqf will load existing territory data recs from the TerritoryCaptureStatus table, and create new records for territories not existing for the ServerID+MapID combo.
  + Tested and working though extDB call to exec the newTerritoryCaptureStatus sql in getTerritories returns \_markerID as a single element array, instead of the desired integer value. Need to look at the persistence\server\setup\extDB\async\_database.sqf to see whether return val needs to be modified or if I’m calling it incorrectly.
* (Feb 16-17) Tasks:
  + Work on extDB versions *getTerritories.sqf* and *saveTerritory.sqf* to get them functioning correctly
  + *getTerritories.sqf:* loads territory capture persistence info from DB on start-up
    - needed to follow other methods used in other database retrieval method algorithms to:
      * get the existing record set with call to extDB\_Database\_async
      * loop over the record set and save data to a temporary array
      * compare what was returned with defined mission territories and create additional records in the database for any that are missing (this will keep the db in sync if/when mission defined territories are changed)
    - created a3wasteland.ini defined sql ‘*getServerTerritoriesCaptureStatus*’ for use in the extDB\_Database\_async call, using ServerID+MapID as selection keys, and an array of DB column names -> local vars key->value pairs that are used to create a comma-seperated string of columns to retrieve. The result array is then parsed and loaded into a local copy of an array mimicking the global **currentTerritoryDetails** array.
    - This array is then compared to the *config\_territory\_markers* public variable array, and any territories that are missing from the local array are added to the database and to the local array.
      * The key to success here in creating new db records was a two-phase insert/update approach to adding new territory data to the database, and getting the form of the markerName string into the correct form. extDB\_Database\_async does a ‘call compile \_queryResult’ on returning from the extDB extension, so that stuff stored as VARCHAR in the database, needs to saved WITH string literals enclosed in quotes. Checking for a missing territory \_markerName from *config\_territory\_markers* in the database with the *checkServerTerritory* sql call, required formatting the \_markerName string as follows:
        + \_markerName = format ["""%1""", \_x select 0]; , where \_x select 0 is the markerName in *config\_territory\_markers*.
      * If the \_result from the *checkServerTerritory* sql call is false (the rec doesn’t exist), a call to *newTerritoryCaptureStatus* sql returns a newly INSERT’d record ID, that is then updated with the markerName and default data with a call to the *updateTerritoryCaptureStatus* sql which is created and formatted by using an array of [columnName, variableName] mapping pairs that are fed to extDB\_pairsToSQL to generate the correct form of data to feed to the SQL call. The new rec is created in the db, and a record is added to the local temporary territories array.
    - The local territories array is returned to the persistence type independent tLoad caller, which returns it to monitorTerritories where it is assigned to the **currentTerritoryDetails.**
    - **Tested and working as of 2/17/2015 ☺**
  + *saveTerritory.sqf:* Updates territory capture persistence info to DB on capture events
    - compiled in tSaveInit as *fn\_saveTerritory*
    - called from \_handleCapPointTick within monitorTerritories, when a territory capture has occurred. Gets passed an array containing: [\_currentTerritoryID, \_currentTerritoryName, \_newTerritoryOccupiersPlayers, \_currentTerritoryOwner, \_currentTerritoryChrono, \_newCapPointTimer].
    - It converts the list of player objects in the passed \_newTerritoryOccupiersPlayers array to and array of UIDs, and like the other use of the *updateTerritoryCaptureStatus sql,* creates an array of [columnName, variableName] mapping pairs that are fed to extDB\_pairsToSQL to generate the correct form of data to feed to the SQL call to update the db record matching \_currentTerritoryID.
    - **Tested and working as of 2/17/2015 ☺**
  + *territoryPayroll.sqf*: Updates territory capture persistence info with new TimeHeld data
    - calls *fn\_saveTerritory* with [\_territoryId, \_territoryName, \_territoryOccupiers, \_territoryOwner, \_territoryChrono, 0]
    - **Tested and working as of 2/17/2015**
  + *logTerritoryCapture.sqf*: Adds record to the territoryCaptureLog table on territory capture events
    - gets called from \_handleCapPointTick with [\_currentTerritoryID, \_currentTerritoryName, \_newTerritoryOccupiersPlayers, \_currentTerritoryOwner]
    - converts the list of player objects in the passed \_newTerritoryOccupiersPlayers array to and array of UIDs
    - creates an array of [columnName, variableName] mapping pairs that are fed to extDB\_pairsToSQL to generate the correct form of data to feed to the SQL call
    - uses *addTerritoryCaptureLog sql* and the formatted SQL to insert a new record
    - **Tested and working as of 2/17/2015**
  + Issues found:
    - On server restart with OPFOR owned/re-loaded territories, player joins as OPFOR and see’s territory’s owned by OPFOR, but with hatched territory pattern, not solid/transparent pattern. Issue appears to be with how /t/c/updateMarkers is handling data passed from server in it’s remote execVM call to updateMarkers.
      * **Resolution**: This occurs for the first player to join the server, but appears to work okay if that player leaves the server and rejoins.
    - Scores shown in captures on player screen do not reflect the territories held either from previous session, or in current one.
* Feb 18-20 Work: Handle territory capture persistence for Independents
  + Db structure:
    - Add ‘GroupHolder’ (varchar(128)) and ‘GroupHolderUIDs’ (varchar(2048)) columns to territoryCaptureStatus table, and ‘GroupHolder’ (varchar(128) column to territoryCaptureLog table.
  + Global currentTerritoriesDetails array
    - Modify getTerritories.sqf/saveTerritory to/save load new columns in territoryCaptureStatus table to the array as row elements 7&8. Revised currentTerritoriesDetails array is now:

|  |  |  |  |
| --- | --- | --- | --- |
| **Column#** | **Field** | **Data Type** | **Db Column** |
| 0 | Marker ID # | INT | ID |
| 1 | Marker Name | STRING | MarkerName |
| 2 | Players In Area (UIDs) | ARRAY | Occupiers |
| 3 | Players In Area (playerObjects) | ARRAY |  |
| 4 | Current owning team | SIDE | SideHolder |
| 5 | Time held / owned | INT | TimeHeld |
| 6 | Time contested | INT |  |
| 7 | Current owning group (groupObject) | GROUP | GroupHolder (xformed to STRING) |
| 8 | Current owning group (UIDs) | ARRAY | GroupHolderUIDs |

* + - Update getTerritories.sqf to load these new data columns in the global *currentTerritories*
    - Update saveTerritories.sqf to save new db columns in calls from monitorTerritories
  + *monitorTerritories.sqf*:
    - Starting with \_handleCapPointTick, which gets called with [\_territoryOccupiersMapConsolidated, currentTerritoryDetails], where the 1st array is an array of [\_territoryName, [\_player]], and returns a temp/new copy of currentTerritoryDetails which is immediately assigned,
    - \_handleCapPointTick loops over the territories in \_currentTerritoryDetails and for each
      * \_currentTerritoryOccupiersPlayers (col 3, above) is passed to \_teamCountsForPlayerArray, returning [\_teamCounts, \_contested, \_dominantTeam], where \_teamCounts is an array containing [\_team, #players], where \_team and \_dominantTeam can be a SIDE or GROUP object to \_currentTeamCounts. If more than one team/group is in the territory, \_dominantTeam is set to sideUnknown and \_contested is set to true.
      * \_newTerritoryOccupiersPlayers (from \_territoryOccupiersMapConsolidated) is passed to \_teamCountsForPlayerArray, returning [\_teamCounts, \_contested, \_dominantTeam], where \_teamCounts is an array containing [\_team, #players], where \_team and \_dominant can be a SIDE or GROUP object to \_newTeamCounts. If more than one team/group is in the territory, \_dominantTeam is set to sideUnknown. If more than one team/group is in the territory, \_dominantTeam is set to sideUnknown and \_contested is set to true.
      * [\_currentTeamCounts, \_newTeamCounts] is passed to \_handleTeamCounts which returns \_action containing one of [“CAPTURE”,”RESET”, or “BLOCK”]
      * If \_newTeamCounts.\_contested is true or the \_newDominantTeam (from the 2nd call to \_teamCountsForPlayerArray) is not the same as the currentTerritoryOwner:
        + If \_action is “CAPTURE”:

**1st iteration: Broadcast capture to current territory owners (NOTE: NEEDS TEAM/GROUP FIX)**

Increment \_newCapPointTimer

* + - * + If \_action is “RESET”, set \_newCapPointTimer to 0
        + If cap point timer > cap period and this is a new capture:

Set the new marker color from \_newDominantTeam

Reset \_newCapPointTimer

**Call \_onCaptureFinished with [\_currentTerritoryOwner, \_newDominantTeam, \_value, \_currentTerritoryName, \_territoryDescriptiveName]**

Set \_*currentTerritoryOwner* to \_newDominantTeam

**Save new territory status / log territory capture**

**Update player scores for all the players who just capped the territory**

* + - * + Call \_updatePlayerTerritoryActivity with [\_currentTerritoryOwner, \_newTerritoryOccupiersPlayers, \_newDominantTeam, \_action] which just loops over players in \_newTerritoryOccupiersPlayers and updates “TERRITORY\_ACTIVITY” variables on them.
      * Loop over all of the \_newTerritoryOccupiersPlayers, and create up-to-date \_newTerritoryOccupiersUIDs array
      * **Set the \_currentTerritoryData record with [\_currentTerritoryID, \_currentTerritoryName, \_newTerritoryOcupiersUIDs, \_newTerritoryOccupiersPlayers, \_currentTerritoryOwner, \_currentTerritoryChrono, \_newCapPointTimer]**
    - \_handleCapPointTick then returns the new \_currentTerritoryData which is assigned to the global currentTerritoryDetails
    - **The public variable A3W\_currentTerritoryOwners is re-initialized with the array of [territoryName, territoryOwnerTeam] values and rebroadcast**
    - If \_newPlayersWithTerritoryActivity>0, remove them from the \_oldPlayersWithTerritoryActivity array, … at next iter start, those remaining will have the “TERRITORY\_ACTIVITY” variable will be nulled out
    - Updates:
      * All references to currentTerritoryDetails and it’s passed/returned equivs updated to include the 2 new group fields
      * Many logic and data structure changes where team (i.e., side-or-group) was referenced to split out team handling and group handling into separate branches/data
      * Add a global mutex variable (monitorTerritoriesActive) that gets set to true on loop wake and to false at the onset of the next sleep so that threads can check and (hopefully) respect with waitUntil’s it goes back to false if set
  + *territoryPayroll.sqf* Updates
    - Add check to the mutex var (monitorTerritoriesActive) on wake and waitUntil it’s clear if set before proceeding.
    - Add creation of a \_newTerritoriesDetails array to permit updates to Indy team group memberships for territories that are capped by independents. When looping over territories to check if a payout is due, pull out all of the currentTerritoryDetails fields to named vars. If a territory is capped by Indy’s, get the group owning the territory, and build a list of all the members in the group. If the group membership has changed, set \_territoryOwnerGroupUIDs to the new list, and set \_refreshNeeded var. At the end of the iteration:
      * if (\_refreshNeeded && \_territorySavingOn), write back the captureTerritory rec to the db
      * set all of the vars into a new \_newTerritoryDetails rec
    - At the end of the loop, refresh currentTerritoryDetails with the newly build \_newTerritoryDetails array
    - If payouts are due:
      * Send OPFOR,BLUFOR payouts & messages to the \_team, as before
      * Send Independent payouts & messages to just the group owning the territory
  + *convertTerritoryOwner.sqf:*
    - is called by:
      * /c/s/g/acceptGroupInvite.sqf,
      * /c/s/g/leaveGroup.sqf,
      * /c/s/kickPlayerFromGroup.sqf

via client-side send of ‘pvar\_convertTerritoryOwner’ to the server when a player joins/leaves/is kicked out of a group

* + - acceptGroupInvite merges the existing group’s territories, and the player’s territories together, and sends this combined list back to the server with the pvar\_convertTerritoryOwner public variable
    - convertTerritoryOwner gets fired on the server by the publicVaribaleHandler, with the revised list of territories sent from the client, it checks the mutex, then loops over the list of territories sent, and then loops over all the recs in currentTerritoryDetails. When the currentTerritoryDetails rec matches the current list of territories rec, if the territory is owned by Independents:
      * sets group ownership of the currenTerritoryDetails rec to the newGroup
      * resets the current refills it with the UIDs of players currently in the passed newGroup
      * If territorySaving is enabled, updates the relevant db record with the new group, & groupUIDs
  + *updateTerritoryMarkers.sqf:*  (the most complicated code bit, in terms of the several ways it gets invoked
    - modify the code to support use of team+group on method 2 invocations (i.e., when called from updateConnectingClients to set markers for all capped territories).
    - For territories owned by the players team:
      * If player team != owning team, use the owning team color and hatched brush for the territory
      * If player is [OPFOR,BLUFOR] or Indy & same group and territory is owned by same team/group will set the territory marker to use the solid brush
      * If player is Independent and the territory is owned by a different independent group, will use the hatched brush
* Feb 21-22 Tasks:
  + Test/debug/test the modifications, fixing issues with territory assignment and player/team scoring along the way. Noted that the 1st player to join a newly restarted server was not getting correct territory capture data in terms of team/group ownership (sides owning were accurately reflected, but e.g. a BLUFOR joiner was seeing capped blufor territories with a hatched pattern instead of a solid one.
  + This led to re-writing how territory capture status gets sent to the player, following the model used to send persistent player status data to the connecting client. It uses the following files/constructs to ensure that all connecting clients get up-to-date territory capture data:
    - *\client\init.sqf:* 
      * call compile preprocessFileLineNumbers “*\territory\client\setupTerritories.sqf”*
        + Creates the client-global function fn\_applyTerritoryData from \territory\applyTerrritoryData to handle data returned from the server pvar, described below.
        + Creates client-global function fn\_requestTerritoryData that sends a publicVariableServer request with “pvar\_requestTerritoryData” containing the array [player, getPlayerUID player, netId player]
        + Creates a publicVariableEventHandler for the server’s return of territory data in a pvar named “pvar\_applyTerritoryData\_”+UID. This handler calls fn\_applyTerritoryData with the data payload of this pvar and sets “territoryDataLoaded” to true when that function completes
      * call fn\_requestTerritoryData
        + kicks off the request-receive-set processes
      * waitUntil “territoryDataLoaded”
    - *\territory\applyTerritoryDtaa.sqf*
      * A short stub that passes the territory data it got passed to the primary territory handler, \territory\client\*updateTerritoryMarkers.sqf*
    - *\server\init.sqf:*
      * [] spawn compile prepocessFileLineNumbers *\territory\server\setupTerritories.sqf*
        + Creates a publicVariableEventHandler for “pvar\_requestTerritoryData” that takes the [player,UID,netID] data in the pvar payload, passes \_player to updateConnectingClients, and returns the data returned by that method back to the client via another pvar called ‘pvar\_applyTerritoryData’ containing data useful for invoking updateTerritoryMarkers on the client to paint the territory marker landscape.
    - *\territory\cllient\updateConnectingClients.sqf* 
      * Called on the server by the pvar event handler for “pvar\_requestTerritoryData”, with the player object to create data for
      * Loops over the currentTerritoryDetails array, and creates and array of arrays containing [markerName, markerTeam, markerGroup] entries (1 per territory)
        + Along the way, if the player is an independent and the marker is currently capped by independents, checks to see if group ownership of the territory should be assigned to the player or whether the player should be assigned to the group owning the territory. If the player captured the territory to begin with, they get ownership of the territory. If the player was previously part of the group that owns the territory (without having left or having been kicked out), they get joined to the owning group.
      * If any territory owner re-assignments were made, it re-broadcasts A3W\_currentTerritoryOwners and updates currentTerritoryOwners variables.
  + **ALL TESTED AND WORKING AS OF Feb 22**

Project 2: Implement Object Persistence for object stored via R3F\_LOG in vehicles

**Current State:**

R3F\_LOG implements object variable attachments to set and track ‘transported’ objects. These variables are set as part of the processing when an object is loaded into a vehicle in the script \addons\R3F\_ARTY\_AND\_LOG\R3F\_LOG\transporteur\charger\_deplace. The variables are unset as part of the processing when an object is unloaded with the script decharger.sqf

* When an object and a vehicle to load it into have been actioned, the script R3F\_LOG\transporteur\charger\_deplace.sqf is invoked
  + The vehicle (\_carrier) is identified as the first returned object in the R3F\_LOG\_classes\_carriers within 22m of the player
  + The vehicle (\_carrier) variable ‘R3F\_LOG\_objets\_charges’ is retrieved and stored to \_objects\_charges. It contains a list of other objects already in the vehicle.
  + If the object will fit into the vehicle (i.e. with the object, vehicle will not exceed its carrying capacity):
    - The object being loaded gets added to the \_objects\_charges list
    - The vehicle variable ‘R3F\_LOG\_objets\_charges’ is re-set with the updated \_objects\_charges list
    - The object being loaded gets the variable ‘R3F\_LOG\_est\_transporte\_par’ set with the vehicle (\_carrier) object
    - The object is attached to the R3F\_LOG\_PUBVAR\_point\_attache object (an object of type ‘Land\_HelipadEmpty\_F’ located at point [0,0,0] with some random offset above 10km height as follows:

[R3F\_LOG\_PUBVAR\_point\_attache, true] call fn\_enableSimulationGlobal;

[\_object, true] call fn\_enableSimulationGlobal;

\_object attachTo [R3F\_LOG\_PUBVAR\_point\_attache, \_position\_attache];

\_object enableCollisionWith \_carrier;

* When an object is selected for unloading:
  + It is removed from the vehicle’s “R3F\_LOG\_objets\_charges” list and stored instead as the object \_objet\_a\_decharger.
  + The vehicle variable “R3F\_LOG\_objets\_charges” is reset with the updated list
  + The object’s variable “R3F\_LOG\_est\_transporte\_par" is set to null
  + The object is detached from the R3F\_LOG\_PUBVAR\_point\_attache object as follows:

[\_objet\_a\_decharger, true] call fn\_enableSimulationGlobal; [R3F\_LOG\_PUBVAR\_point\_attache, true] call fn\_enableSimulationGlobal;

detach \_objet\_a\_decharger;

* + If the object is ‘moveable’ object, it is passed as an arg to an execVM of “\R3F\_LOG\objet\_deplacable\deplacer.sqf" for placement in a safe location.

**Solution overview:**

* Since objects and vehicles are already persisted in the db, we need to ensure that the objects have already been saved, and if not, force saves upon them. The db record ID for an object is stored on the object as the value of the variable “A3W\_objectID”, and the db record ID for a vehicle is stored on the object as the value of the variable “A3W\_vehicleID”.
  + A3W\_vehicleID gets set when the vehicle is first saved, and is re-set on server restarts via getVehicles and vLoad scripts
  + A3W\_objectID get set when the vehicle is first saved, and is re-set on server restarts via getObjects and oLoad scripts
* When an object is loaded into a vehicle:
  + A 2nd vehicle variable list should be retrieved and updated to include the A3W\_objectIDs of loaded objects. Call it R3F\_A3W\_objectIDs
  + A 2nd object variable should be set to hold the A3W\_vehicleID of the vehicle it is loaded into. Call it R3F\_A3W\_vehicleID
  + Note: These variables should referenced in the lists of variables to store and retrieve in the saveVehicle, saveObject, getVehicles, and getObjects scripts.
* On server re-starts:
  + oLoad runs before vLoad, so follow the vehicle load into vehicle process defined above when re-instantiating objects as far as attaching them to the R3F\_LOG\_PUBVAR\_point\_attache variable. Also, create and return a list of instantiated objects and their A3W\_vehicleIDs that got this treatment, to pass to the vLoad process
  + When vLoad runs, use this list to complete the R3F\_LOG style processing for the object
    - If a vehicle has anything in its R3F\_A3W\_objectIDs, find the objects from the passed list, and add each to the vehicle object’s ‘R3F\_LOG\_objets\_charges’ variable
* Modifications to vehicle and object saving:
  + In fn\_getVehicleProperties, get the vehicle’s R3F\_A3W\_objectIDs variable, and push its contents onto the script local \_variable array
  + In fn\_getObjectProperites, get the object’s R3F\_A3W\_vehicleID , and push it onto the script local \_variable array