Empires of Avatharia

Group 22

Christopher Raschke
Johan Wessman
Jashar Ghavampour
Johan Granberg

5.5 Detailed Design

About the classes:

In the following pages each class will be presented in the following format:

> **Class Name**
> (References to RD)
>
> Int ***variableName***; // Comments that clarifies the use of the variable (if necessary)
>
> /*
> * Comments about the method
> */
> **methodName(String input)**

-pre:	Prerequisites.
-post:	Any post-method conditions
-return:	What the method returns
-data access:	Database accesses
-called by:	The classes that calls this method
-calls:	The classes (and/or methods) this method calls

It is also assumed that each class contains “setters” and “getters” for each local variable. (“Setters” and “getters” are methods that sets a variable respective returns a variable). Local variables are marked with bold and italic characters.
Some methods may be split into two or more methods when they are implemented but they may, for simplicity reasons, be viewed as one single entity.

Cross-reference chart

Should be read as:
RD section : classes implementing the contents of this section

4.1.1 Registering: class User.
4.1.2 Logging in: class User, class Player, class accountmanager.
4.1.3 Recover password: class User, class accountmanager.
4.1.4 The in-game-environment: class GameEngine, class Building, class UnitShell, class Province, class Area, class CombatLog and class Upgrades.
4.1.5 Build buildings: class Buildings, class Upgrades.
4.1.6 Train units: class UnitShell, class Unit and class Building.
4.1.7 Chat with other players: class Channel, class Messages.
4.1.8 Send private messages to other players: class Mail, class Messages.
4.1.9 Manage armies: class Province, class Combat, class Unit, class UnitShell and class Area.
4.1.10 Select a view: class CombatLog, class Province, class Area and class Game Engine.
6.1.1 Registering: class User.
6.1.2 The in-game-environment: class GameEngine, class Building, class UnitShell, class Province, class Area, class CombatLog and class Upgrades.

6.1.3 Build buildings: class Buildings, class Upgrades.
6.1.4 Train units: class UnitShell, class Unit and class Building.
6.1.5 Chat with other players/ Send private messages to other players: class Channel, class Messages and class Mail.
6.1.6 Manage armies: class Province, class Combat, class Unit, class UnitShell and class Area.

Class Building

(Partly implements the Build Buildings (4.1.5), Build Building (4.3.3) and “Build buildings” (6.1.3) requirements in the RD)

Int ***level***;
LinkedList ***upgrades***;
String ***name***;
String ***image***;
Int[] ***coords***;

/*
* Increments the level of a Building by one and updates the image
*/

levelUp()

-pre:	The building is level X and has ***image*** Y.
-post:	The building is level X+1 and has ***image*** Z, where $Z \neq Y$.
-return:	Void
-data access:	Reads ***image*** string Z from the Database via the class DB.
-called by:	class GameEngine
-calls:	None

/*
* Applies an upgrade to a building
*/

applyUpgrade(Upgrade upgrade)

-pre:	The building has ***upgrades*** $\{X_1,\ldots,X_n\} = Z$ where Z may be {Ø}
-post:	The building has ***upgrades*** $\{X_1,\ldots,X_n,X_{n+1}\}= X$
-return:	Void
-data access:	None
-called by:	class GameEngine
-calls:	None

/*
* Creates a new building object
*/

Building(String n, int[] co)

-pre:	No building at these coordinates exists
-post:	A building with ***name*** = n, ***coords*** = co, ***level*** = 1, **upgrades** = {Ø}, ***Image*** = Z exists
-return:	Void
-data access:	Reads ***image*** string Z from the Database via the class DB.
-called by:	class GameEngine
-calls:	None

Class Upgrade

(Partly implements the Build Building (4.3.3 (see Variations 7)), Build Buildings (4.1.5) and the “Build buildings” (6.1.3) requirements in the RD)

String ***name***;
Int[] ***bonuses***; //Contains bonus percentages

/*
* Creates a new upgrade object
*/

Upgrade(String n, int[] b)

-pre:	This upgrade object does not exist
-post:	An upgrade object exists with ***name*** = n, **bonuses** = b.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

Class Area

(Partly implements the View Map (4.3.9 (see “Main Succcess scenario” 2 and “Extensions” 2b1)) and the “In-game environment” (6.1.2) requirements and use case in the RD)

Int[] ***coords***.	
Province ***p***;	//May be { Ø }
Int ***factionId***;	//The faction it belongs to (0 if no faction)

/*
* Creates a new area object
*/

Area(int[] c)

-pre:	This area object does not exist
-post:	An area object with ***coords*** = c, ***factionId*** = 0, ***p*** = null exists.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

Class Province

(Partly implements the “Build buildings” (4.1.5) , “Train units” (4.1.6), “Manage Armies” (4.1.9) , “Select a view” (4.1.10) and “In-game environment” (6.1.2) requirements)

String ***name***;
Int[] ***coords***;
Int ***playerID***;
Building[][] ***buildings***;
Sides ***armys***;
Int ***gold***;
Int ***mana***;

/*
* Creates a new province object
*/

Province(String n, int[] c, int p)

-pre:	This province object does not exist
-post:	A province object with ***name*** = n, ***coords*** = c, ***playerID*** = p exists.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

/*
* Adds a building to a specific coordinate.
*/

addBuilding(int[] c, Building b)

-pre:	There is no building at coordinate c.
-post:	***buildings***[c_x , c_y] contains building b
-return:	Void
-data access:	None
-called by:	class GameEngine
-calls	None

Class Army

(Implements the “Manage Armies” (4.1.9) and the “Manage armies” (6.1.6) requirements in the RD)

Int[] ***coords***;
Int ***factionId***;
UnitShell[] ***units***;
Int ***mission*** // If the army has a ***mission*** then ***mission*** $\neq 0$
// Ex 1100 , 1 = Defend, 100 = 100 minutes

/*
* Creates a new army object
*/

Army(int[] c, int f)

-pre:	This army object does not exist
-post:	An army object with ***coords*** = c, ***factionId*** = f and ***units*** = {Ø} exists.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

/*
* Adds a new unit to the army
*/

addUnit(UnitShell u, int pos)

-pre:	No unit shells exists at position pos in ***units***;
-post:	A unit shell u is added to position pos.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

/*
* Removes a unit from the army
*/

removeUnit(UnitShell u, int pos)

-pre:	A unit shell u exists at position pos.
-post:	No unit shells exists at position pos in ***units***.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

Class Sides

(Partly implements the War (4.3.9 (see “Main success scenario” 5) use case) and the “Manage armies” (6.1.6) requirement)

ArrayList ***defendingSide***;
ArrayList ***attackingSide***;
Int ***defendingSideFactionId***;

/*
* Creates a new sides object
*/

Sides(int def)

-pre:	This side object does not exist
-post:	A side object with ***defendingSideFactionId*** = def.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

/*
* Adds a new army to the sides
*/

addArmy(Army a, int i)

-pre:	No side contains a.
-post:	If I == ***defendingSideFactionId*** then the defending side contains a, else the attacking side does.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

Class Faction

(Partly implements the “In-game environment” (6.1.2) requirement)

Int ***number***;
Unit[] ***unitPool***;
HashMap ***players***;   // Consists of all the players in that faction

/*
* Creates a new faction object
*/
Faction(int n)

-pre:	This Faction object does not exist
-post:	A Faction object with ***number*** = n exists.
-return:	Void
-data acess:	None
-called by:	class Startup
-calls:	None

/*
* Adds a new player to the faction
*/
addPlayer(Player p)

-pre:	Player p is not in ***players***
-post:	Player p is in ***players***.
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

/*
* Removes a player to the faction
*/
removePlayer(Player p)

-pre:	Player p is in ***players***.
-post:	Player p is not in ***players***
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

/*
* Returns a player object from the faction
*/
getPlayer(int id)

-pre:	No prerequisites.
-post:	Player p is returned where p has id ***id***.
-return:	Player p
-data acess:	None
-called by:	class GameEngine

-calls: None

Class Mail

(Partly implements the “Send private messages to other players” (4.1.8) and the “Chat with other players / Send private messages to other players” (6.1.5) requirements as well as the “Private Message” use case (4.3.6))

String ***sender***;
String ***receiver***;
String ***message***;
Date ***timestamp***;
boolean ***read***;

/*
* Creates a new Mail object
*/

Mail(String s, String r, String mess)

-pre:	There exists no mail object
-post:	There exists a mail object with ***sender*** = s, ***receiver*** = r, ***message*** = mess, ***timestamp*** = current date, ***read*** = false
-return:	Void
-data acess:	None
-called by:	class GameEngine
-calls:	None

Class Combat

(Partly implements the “Manage Armies” (4.1.9) and the “Manage armies” (6.1.6) requirements as well as the “War” (4.3.5) use case)

Sides ***s***;
HashMap ***combatlogs***;

/*
* Calculates the outcome of a combat based on two different sides and calls the
*CombatLog class to print a combat log for the combat.
*/

Combat(Sides s)

-pre:	Two different sides are in the same province.
-post:	Combat has been calculated and one side left victorious. this.***s*** = s;
-return:	Void
-data access:	None
-called by:	class GameEngine
-calls:	class CombatLog

/*
* Computes a turn of combat. Called every ten minutes by the GameEngine.
* Writes down reseults in a HashMap, this is saved until the combat ends.
*/

doCombat()

-pre:	A combat has been started.
-post:	One more turn of combat calculated. Returns true if one side has been vanquished.
-return:	boolean
-data access:	None
-called by:	class GameEngine
-calls:	class CombatLog

/*
* After combat has ended the HashMapp containing the logs for the different players is
*returned to the GameEngine.
*/

finishCombat()

-pre:	A combat has ended, a doCombat()-call has returned true.
-post:	A HashMap, ***combatlogs***, containing all of the logs for the players is returned to the GameEngine.
-return:	HashMap
-data access:	None
-called by:	class GameEngine
-calls:	None

Class Messages

(Partly implements the “Send private messages to other players” (4.1.8), “Chat with other players” (4.1.7) and the “Chat with other players / Send private messages to other players” (6.1.5) requirements as well as the “Private Message” (4.3.6) and the “Chat Message” (4.3.6) use cases)

/*
* Searches for mails within the specified interval and after a certain timestamp for the
*chosen player.
*/
mail[] getMail(String PlayerID, int from, int to, int timestamp)

-pre:	The user has accessed the mail-section in the game.
-post:	The mails within the specified parameters are showed to the user.
-return:	mail[] , returns all the relevant mails in a vector.
-data access:	Accesses the DB for mails via the DB-class.
-called by:	class GameEngine
-calls:	class DB

/*
* Returns how many mails a player has in his/her mailbox.
*/
int howManyMail(String PlayerID)

-pre:	The user has accessed the mail-section in the game.
-post:	How many mails the players has in his/her inbox is displayed.
-return:	int, the # of mails.
-data access:	Accesses the DB and mail-table and returns the number of posts in it.
-called by:	class GameEngine
-calls:	class DB

/*
* Sends a mail to the user specified in the Mail-object being sent.
*/
sendMail(Mail a)

-pre:	The user has pressed “New” in the mail section of the game.
-post:	The mail ***a*** is sent to according to the information contained in the Mail-object.
-return:	None
-data access:	Accesses the DB and enters the mail to the correct players mail table.
-called by:	class GameEngine
-calls:	class DB

Class Unit

(Partly implements the “Manage Armies” (4.1.9), “Train Units” (4.1.6) and the “Manage armies” (6.1.6) requirements as well as the “War” (4.3.5) and the “Train a unit” (4.3.4) use cases)

```
String name;
Int unitID
Int specialFeat;
Int[] stats;                    //{maxHealth, attack, defence}
Int[] unitTypeModefiers; //damage modifiers against other units depending on unittypes

/*
* Creates a new unit object
*/
Unit(String n, Int id Int sFeat, Int[] startstats, Int[] startUnitTypeModefiers)
-pre:           No unit.
-post:          A unit with name = n, unitID = id, specialFeat = sFeat, stats = startstats,
                and unitTypeModefiers  = startUnitTypeModefiers
-return:        Void
-data access:   None.
-called by:     class Startup
-calls:         None
```

Class UnitShell

(Partly implements the “Manage Armies” (4.1.9), “Train Units” (4.1.6) and the “Manage armies” (6.1.6) requirements as well as the “War” (4.3.5) and the “Train a unit” (4.3.4) use cases)

Int ***level***;
Int ***health***;
Int ***originalUnitID***;
Int ***experience***;

/*
* Increments the level of a UnitShell by one
*/

levelUp()

-pre:	The unit is ***level*** X
-post:	The unit is ***level*** X+1
-return:	Void
-data access:	None
-called by:	class Combat
-calls:	None

/*
* Creates a new unitshell object
*/

Unit(Int orgID)

-pre:	No unit.
-post:	A unit with ***originalUnitID*** = orgID, ***level*** = 0, health equal to the original units ***maxHealth*** and ***experience*** = 0
-return:	Void
-data access:	None.
-called by:	class GameEngine
-calls:	(faction.unit(orgID)).getMaxHealth();

Class Event

Int ***evenType***;
Object[] ***eventParameters***;
/*
* Creates a new event.
*/
Event(Int event, Object[] parameters)

-pre:	No event.
-post:	An event that stores relevant data that applies to the event.
-return:	Void
-data access:	None.
-called by:	class Combat.
-calls:	None

Class EventQueue

Timer ***timer***;
Heap ***queue***;

getEvent()

-pre:	the timer tells GameEngine that its time to get the next event.
-post:	The event has been sent to the GameEngine and the event is removed from the EventQueue.
-return:	Event
-data access:	None.
-called by:	class GameEngine.
-calls:	None.

addEvent(Event insertevent, Date eventdate)

-pre:	The eventqueue does not contain the event.
-post:	The eventqueue contains the event and the timer is set accordingly after eventdate if needed.
-return:	Void
-data access:	None.
-called by:	class GameEngine.
-calls:	None.

/*
* Starts up an eventqueue.
*/
EventQueue(Date enddate)

-pre:	No eventqueue.
-post:	A Eventqueue started with a running timer set to an “end game event” with the time set to enddate that is inserted into the heap as the only element so far.
-return:	Void
-data access:	None.
-called by:	class Startup.
-calls:	None.

Class CombatLog

(Partly implements the “Select a view” (4.1.10) (the “combat log” is a view) and the “In-game environment” (6.1.2) requirements)

Int ***time***;
String ***mission***
String ***province***;
String ***log***;
/*
* Creates a new combatlog object from the information given by combat.
*/
CombatLog(String[ ] n)

-pre:	No combatlog.
-post:	A combatlog with the information from n.
-return:	Void
-data access:	None.
-called by:	class Combat.
-calls:	None

Class Channel

(Partly implements the “Chat with other players” (4.1.7) and the “Chat with other players / Send private messages to other players” (6.1.5) requirements as well as the “Private Message” (4.3.6) use case)

String ***name***;
String ***password***;
LinkedList ***messageLog***;
ArrayList ***playersInChannel***;
HashSet ***playerNamesToPlayersSet***;

/*
* Used by players to join a channel.
*/
addPlayer(String playerName, String channelPassword)

-pre:	Player is not in the channel;
-post:	If the password was correct the player is in the channel and has loaded the messageLog.
-return:	Void
-data access:	None.
-called by:	class Channel.
-calls:	None

/*
* Used by players to leave a channel.
*/
removePlayer(String playerName)

-pre:	The player is in the channel;
-post:	The player is in the channel
-return:	Void
-data access:	None.
-called by:	class Channel.
-calls:	None

/*
* Used by players send a message to the channel.
*/
sendMessage(String playerName, String message)

-pre:	The player wants to send a message
-post:	The has sent a message to the channel(updating ***messageLog***), which in turn sends the message to the other players in the channel telling them that there has been a message sent.
-return:	Void
-data access:	None.
-called by:	class Channel.
-calls:	None

/*

* Used by players to receive messages sent by other players.
*/

recieveMessage()

-pre:	The player has the old message log;
-post:	The player has the new message log and have thus received the message.
-return:	LinkedList messageLog
-data access:	None.
-called by:	class Channel.
-calls:	None

/*
* Creates a new empty channel, if password is empty it's a new public channel
*/

Channel(String channelName, String channelPassword)

-pre:	No channel with ***name*** = channelname;
-post:	A channel with ***name*** = channelname and ***password*** = channelpassword.
-return:	Void
-data access:	None.
-called by:	class Messages.
-calls:	None

Class GameEngine

(Partly implements “The in-game-environment” (4.1.4), “Build Buildings” (4.1.5), “Train units” (4.1.6), “Manage armies” (4.1.9), “Build a building” (4.3.3), “Train a unit” (4.3.4), “War” (4.3.5) requirements and usecases in the RD)

Faction[] ***factions***
HashMap ***activePlayers***

/*
* Takes an event from the httpserver and checks if it would be legal do add the event to
*to the EventQueue. Player pays the costs for putting the event on the EventQueue. This
* method together with handleEvent will probably be split up into many methods but you
* may view them as one entity.
*/

putEvent(Event inevent)

-pre:	No inevent in EventQueue.
-post:	If the event is legal, there will be an Event object with inevent data in the EventQueue. Made necessary adjustments to the database for the events to be allowed to occur.
-return:	Void
-data access:	Reads relevant data to verify that the Event is legal(gold in a province etc) and writes new relevant information to the database.
-called by:	class HTTP server
-calls:	class EventQueue, class DB

/*
* Gets an event from the EventQueue and handles it appropriately. This method together
* with putEvent will probably be split up into many methods but you may view them as
* one entity.
*/

handleEvent(Event eventToHandle)

-pre:	Event eventToHandle has not been handled and was in the EventQueue
-post:	The object eventToHandle has been handled and the appropriate methods in other classes has been called.
-return:	Void
-data access:	None
-called by:	class EventQueue
-calls:	class Building, class UnitShell, Class Combat, …

/*
* Creates a new GameEngine object
*/

GameEngine(Faction[] f)

-pre:	No GameEngine exists.
-post:	A GameEngine oject exists and it has started necessary game threads.
-return:	Void

-data access:	None
-called by:	class Startup
-calls:	None

Class Player

(Partly implements the “Logging in” (4.1.2), “To login to the game”(4.3.2), “Private message”(4.3.6), “Chat message”(4.3.7) requirement and usecases in the RD)

ArrayList ***provinces***; //Contains all the province-id’s that a player controls.
int ***faction***;

/*
*Creates a new player object and takes the id of the faction that the player play as, as
*parameter.
*/
Player(int faction)
-pre: None
-post: A new player object is created and successfully logged in.
Return: Void
data access: None
called by: The object itself when created.
calls: None

Abstract Class StartUp

/*
* Initiates the game engine and all other nessecary components
*/

StartUp()

-pre:	The game is not started.
-post:	The game engine and all other nessecary components has been created and initiated.
-return:	Void
-data access:	None
-called by:	class EmpiresOfAvatharia.main
-calls:	class Factions, class HTTPserver, class GameEngine, class Messages, class DB

Class User

(Partly implements the Registering (4.1.1)(6.1.1), Logging in (4.1.2) and Recover Password (4.1.3) requirements in the RD)

String[] userinfo;
String ***playerid***;

/*
* Returns the Player from the database whose id is the same as the User-objects.
*/

getPlayer()

-pre:	Player information is needed.
-post:	The corresponding player info is returned.
-return:	Player p.id == ***playerid***;
-data access:	The database for player information.
-called by:	class GameEngine
-calls:	None

Class AccountManager

(Partly implements the Logging in (4.1.2) and the Recover Password (4.1.3) requirements in the RD)

/*
* Logs in a user and returns the corresponding user-object.
*/

login (String name, String pass)

-pre:	A user is not logged in and tries to log in.
-post:	The user is logged in.
-return:	User u.
-data access:	None
-called by:	class HTTPServer
-calls:	None

/*
* Registers a new user and ads the entered information to the database.
*/

register(String[] info)

-pre:	A user who's not registered tries to register.
-post:	The user is now registered.
-return:	None
-data access:	Contacts the database and enters the new information for the new user.
-called by:	class HTTPServer
-calls:	None

/*
* Checks if a certain player has the privilege to perform a certain event, and returns to
*true or false depending on the result.
*/

hasPrivilege (Event e, Player p)

-pre:	A user tries to perform something in the system, clicking a button etc.
-post:	The user gets to perform the action if true is returned and the other way around if false is returned.
-return:	True/False
-data access:	Accesses the database to check the users privilege.
-called by:	class HTTPServer
-calls:	None

Class DB

String name;
String password;
Connection conn;

/*
* Opens connection to the database
*/
init()

-pre:	There is no connection to the database.
-post:	A connection is established to the database if the username and password are correct and the database is online.
return:	Void
-data access:	None.
-called by:	Startup
-calls:	None

/*
* Checks if there is a connection to the database
*/

isConnected()

-pre:	Some system component requests connection status.
-post:	The method returns status as a Boolean value.
-return:	boolean
-data access:	None.
-called by:	All classes requesting access to the database.
-calls:	None

/*
* Closes database connection
*/

destroy()

-pre:	There is an active database connection.
-post:	The active connection is disconnected.
-return:	Void
-data access:	None.
-called by:	All classes requesting access to the database.
-calls:	None

/*
* Executes a query that modifies the database by inserting, deleting or updating data.
*/

executeModifyQuery(String query)

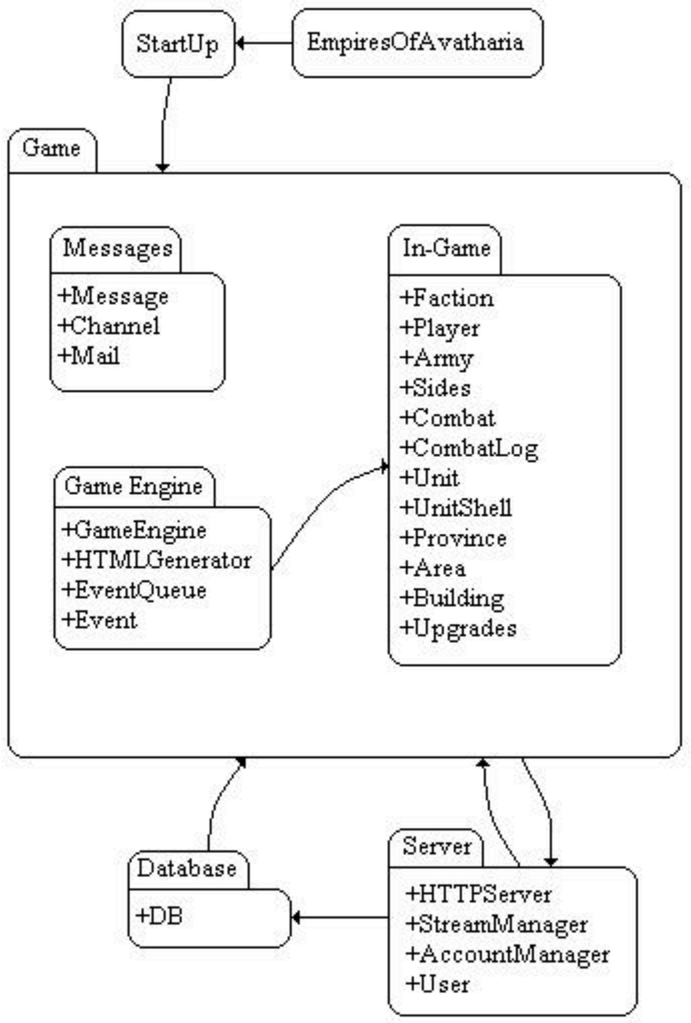
-pre:	There is an active database connection and a valid query is sent as an argument.
-post:	A database query has been performed
-return:	Int queryResultState
-data access:	Tables affected by query.
-called by:	All classes requesting access to the database.
-calls:	None

/*
* Executes a query that fetches data.
*/

executeSelectQuery(String query)

-pre:	There is an active database connection and a valid query is sent as an argument.
-post:	A database query has been performed
-return:	Array resultSet
-data access:	Tables affected by query.
-called by:	All classes requesting access to the database.
-calls:	None

5.6 Package Diagram


StartUp
EmpiresOfAvatharia
Game
Messages
+Message
+Channel
+Mail
In-Game
+Faction
+Player
+Army
+Sides
+Combat
+CombatLog
+Unit
+UnitShell
+Province
+Area
+Building
+Upgrades
Game Engine
+GameEngine
+HTMLGenerator
+EventQueue
+Event
Database
+DB
Server
+HTTPServer
+StreamManager
+AccountManager
+User