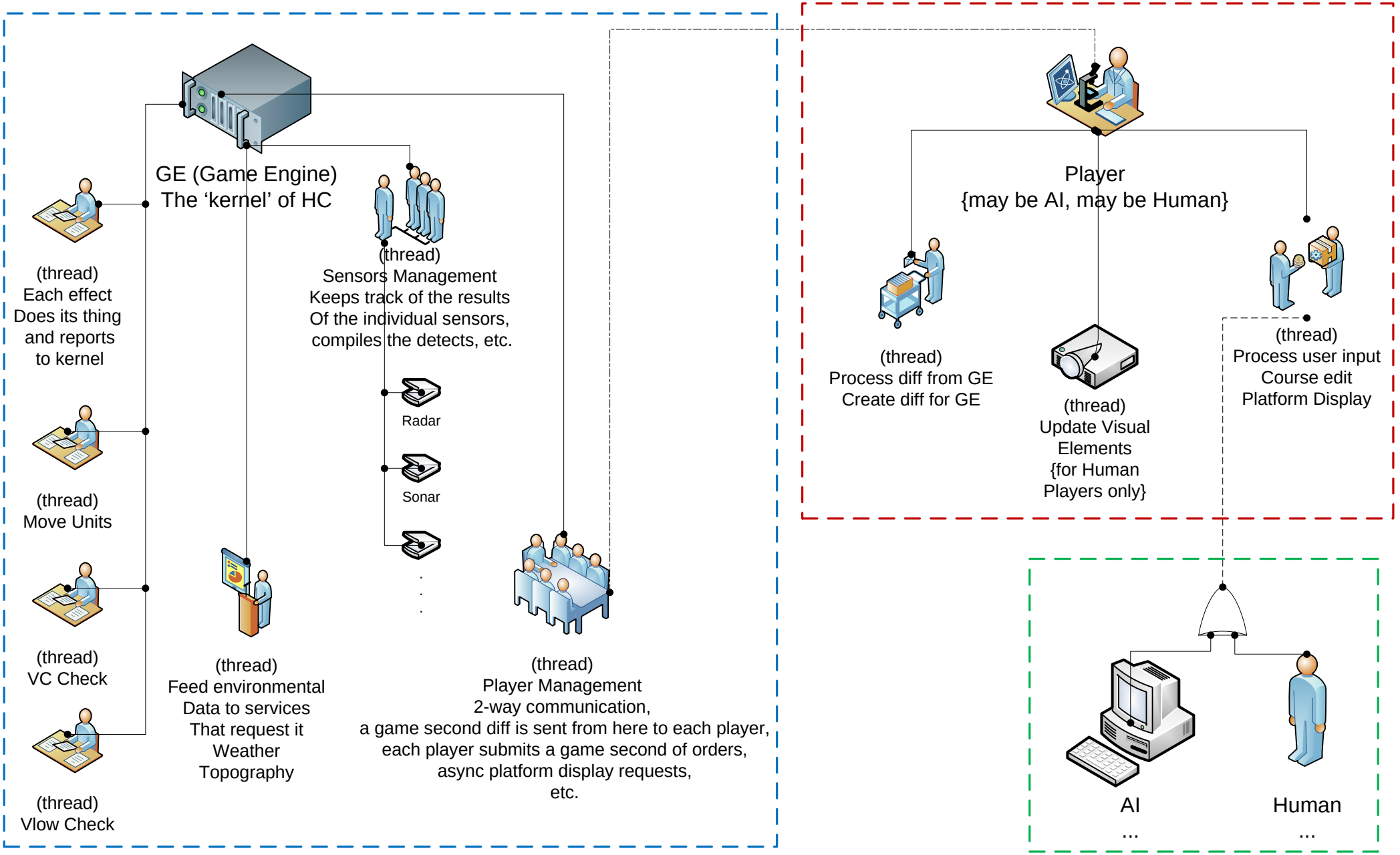
HC Restructure Idea Set 1

Tony Eischens revision: 2007/10/27

- *Central Ideas:*
- Separation of Game Engine and User Interface.
- Synchronization every game second. Do your game second worth of processing and report back to the kernel.
- Engine <-> UI protocol to be lite, i.e. sends difference sets, not full game state each second



HC Restructure Idea Set 1

Tony Eischens revision: 2007/10/27

See attached Visio Drawing for the *Central Ideas*.

Impetus:

The Harpoon Classic GE and SE are crafted in the C language, this makes coding a tedious process, much more so when considering the SE is a 16-bit application, and much more so in UI coding. A restructure brings an opportunity to give the game a decade old heart instead of a nearly two decade old heart. A second goal is to modularize the game as much as possible with the hope of one day being able to release as many of the interface specifications as possible to let those outside of AGSI take a stab at writing a radar model, making a new UI, writing a different AI, etc.

Benefits:

- Multiplayer is inherently enabled in the restructure. An AI player is treated the same way by the game engine as a human player.
- The GE becomes multithreaded, allowing multi-core processors to speed the game.
- Game components might use the IP protocol to communicate, meaning not only is multithreading used, but it would be possible to use multiple computers teamed to run one game.
    - Computer A might only run the Radar model
    - Computer B might Move Units and check VCs
    - Computer C might run the kernel
- Game could become somewhat platform ambivalent.
    - Enablers
        - Separation of Game Engine and user interface
        - IP protocol inter-component communication
    - In theory the Radar model might run on a Linux box while the Sonar model is on a Windows 2000 box, the player may be running the client on a Vista box.
    - Separating the UI from the GE makes cross-platform coding much easier!

A few examples of how a game might operate after such a restructure:

1. Tony decides to play 2nd War Between the States
    a. Scenario file defines each sensor model to use
        i. AGSI Radar 2007.001
        ii. AGSI Sonar 2007.002
        iii. AGSI Visual 2007.000
        iv. …
    b. Scenario file defines which database to use and/or contains relevant database entries for the scenario.
        i. Expect to be able to define multiple databases, i.e. AGSI Radar 2007.001 may require a database format 1.1 while AGSI Sonar 2007.002 may require a database format 1.2.
            1. This isn't just a negative, think about using a super whiz-bang radar model with continuously variable RCS values for a plane, you don't want to be stuck using the minimal radar descriptors in the default

database right? So you can extend beyond what AGSI defines, new sensor models, whatever database layout you need, etc.

c. Fire up the game user interface, written in oh, say Delphi this time. The UI offers to start a local game engine, Tony agrees, all GE components will run locally.
d. HTML Export DLL named Cost of Battle is enabled.
e. Scenario selected, Tony plays Blue, AI assigned to Red, AI will run on local computer as well.
f. Scenario starts, game plays out with with all the components running under the scenes.
    i. Tony pulls up a web browser and goes to the web page provided by the Cost of Battle Export DLL, wow, cool, the DLL is composing a cost of materials used by each side in real time.
    ii. Brad gets home from work, signs in to IRC, Tony asks if Brad would like to take over for the AI, he says sure.
        1. Save the game
        2. Load up the save, this time Brad replaces the AI
        3. Tony and Brad continue playing

2. Tony has created this huge scenario with 5,000 units, 3 sides
    a. It can't possibly be run on one 2007 era computer.
    b. Spread the workload
        i. GE Kernel runs on Ubuntu Linux box
        ii. Red AI Player runs on Windows 2000, talks to kernel over IP
        iii. Green AI Player runs on OS X, talks to kernel over IP
        iv. Radar runs on a Vista box
        v. Sonar runs on a XP box
        vi. Move units runs on another XP box
        vii. ESM runs on a RedHat Linux box
        viii. Sensors accumulator runs on Server 2003 box
        ix. Tony plays on a Vista notebook from home, all those roaring boxes are too loud at the office. Communication via IP back to kernel.
    c. Heh, look, that 5,000 unit scenario is now playable! Setting it up was a pain, but it runs, and could just as easily have 3 or 6 human players (multiple humans on some sides).
3. Rene really wants ANW with a nice interface so he writes his own sensor models and plugs them into HC
    a. Rene Radar 2007.010
        i. RCS of each platform defined in custom DB structure
            1. Top/Bottom/Front/Back/Right/Left
        ii. Radars defined in finer grained detail
            1. Search output power described as a function that decreases output power as angle from straight forward changes.
            2. Scan rate defined
            3. Elements in AESA radars treated individually
            4. Receive sensitivity given similarly detailed treatment
    b. Rene Sonar 2007.010

        i. Unique susceptibility rankings for each platform across multiple frequency bands.
        ii. Surface ducting
        iii. Deep sound channel
        iv. CZs more accurately modeled
        v. Multiple layers possible
    c. Rene lines up his custom database add-ons to feed the models, makes a scenario, defining the use of Rene Radar 2007.010 and Rene Sonar 2007.010.
    d. Fire up the GE, it reads the scenario and loads the proper sensor models and goes to town.
4. Tony wants to do a bunch of scenario testing
    a. Nobody said we need a Human Player, fire up two AI Players and a Human spectator Player. Or, skip the Human spectator all together and just gather the information via Export DLLs.
5. Sci-Fi day…
    a. Tony writes a new Effect, Laser Weapon
    b. Scenario says it needs the Laser Weapon effect so GE loads it up
    c. Cool, I can fire lasers, ah crap, so can the AI!

The AI Question:

- Isn't the AI horribly embedded in everything the game does, the drawing just yanks it out into its own place?!?
    - Well, that was one of the moments of clarity when drawing this out on a big piece of paper, the AI Player(as an analog to a Human Player) does very little in HC so extracting it isn't as big a deal as we all may have thought.
    - The AI only has two main actions
        - Planned Stuff
            - LR Patrols
            - Attacks
            - Ferrying
        - Off-the-Cuff Stuff
            - Attacks against detected targets
                - Separate formation patrols to attack
                - Launch aircraft to attack
                - Fire guns/missiles/torps/bombs
    - The AI does not have a strategy, a long term goal, etc.
        - Sure would be nice to give it those eh…
    - Another moment of clarity is that the Player 'object' (everything in the Red dashed box in the diagram) has all of the information the AI needs so it doesn't need to be treated as a special entity, it just sits in the place of a Human (green dashed box) and has the side benefit of skipping the UI processing.

Development Language(s):

- So Tony is a Delphi nut, one the VERY FEW Delphi nuts since nobody knows about Delphi… This plays a role in this section. What also plays a role is that modern

programmers easily switch languages and there is nothing unapproachable about Delphi for a C++ programmer.

- As noted before, HC the GE and SE are written in C. Orders Writer and BS Builder are mostly Delphi. The PE is Access 97, enabled by db_utils (C language).
- For Cross-Platform development there are many choices, RAD tools however are few and far between.
    - Candidates (rated A – F, A is easy/good, F is hard/bad)
        - Java – multi-platform capable {A}, UI design {C}, communications coding {C}.
        - C++ - multi-platform capable {B}, UI design {F}, communications coding {D}.
        - C#/VB.NET/etc – minimally multi-platform {F}, UI design {A}, communications coding {A}
        - Delphi – minimally multi-platform {D}, UI design {A}, communications coding {A}
            - Much Delphi code is directly compilable by Lazarus, giving a slight multi-platform capability. Delphi is solidly in the Win32 camp however, no 64-bit version even on the horizon.
            - http://www.codegear.com (see free Turbos as well as Delphi 2007)
        - Lazarus – multi-platform capable {B}. UI design {B}, communications coding {A}
            - Delphi Clone but cross platform. {B} ranking on cross-platform since you must compile a different executable for each platform, unlike say Java which compiles upon run.
            - Limited UI components available at this time.
            - Stable development platform for i386 Linux, OS X Intel and PowerPC, Windows.
            - Same communications components as Delphi (Indy suite)
            - Open source software (Free for commercial use, can use to develop commercial software)
            - http://www.lazarus.freepascal.org/
- Since I'm pretty handy with Delphi/Lazarus and feel that it is much easier to develop in than C++/Java, that's the route I'd take, more bang for each hour spent programming. I envision the Player UI being written in Delphi and the rest in Lazarus. The Delphi UI to take advantage of the wide range of components available (see www.tmssoftware.com for just a sampling).

How the various pieces talk to each other:

- TCP/IP – potentially XML messaging if it isn't too inefficient, otherwise a custom protocol on TCP.
    - GE kernel <-> Player(s)
        - Probably two connections per player, one for the regular per game second requests, one for the async requests like Platform Display information.
    - GE kernel <-> Sensors Manager
        - Allows the sensor data aggregator to potentially run on a separate machine since there could be a lot of processing.

    - Sensors Manager <-> Sensor
        - Each individual sensor shall be a TCP server (potentially providing a poor man's way to multi-thread).
        - Allows each sensor type to run on a different computer.
    - GE kernel <-> Effects
        - Mainly to ease addition of new Effects but also a hedge against certain effects taking a ton of time.
        - Perhaps an effects aggregator is needed like the sensors aggregator?
    - Other neat thing about the IP interfaces, they carry no language requirement, you could write a Radar model in Python, a user interface in PHP, IP connections are pretty universal and expand upon the whole modest approach at language neutrality in this restructure!
- DLL (shared object library, windows DLL, etc.)
    - Data export
- Other IPC (or just other threads of the kernel)
    - Compiling data for each game second.
- Quick legend, anything in the TCP/IP category can be spread across multiple machines, the other categories mean the stuff runs on the same machine as the GE kernel.

Effects explained (i.e. what is this Effects term you keep talking about?):

- Effects very roughly map to the various work units of the Harpoon 3 paper rules. The best explanation may be to list some Effects, each Effect gets called once per game second (it does not however have to do anything in each and every game second).
    - Move Units – pretty self explanatory, takes the course, speed, and location and calculates the next location.
    - Area Defense – Sets up area defense except point defense, launches SAMs, shoots guns.
    - Sub AI – Tells the sub how to maneuver, prosecute contacts (some AI mixed in here, good to remember that for separating AI)
    - Course Checks – are we at a waypoint, if we are, turn, speed up, etc.
    - VC Check – have victory conditions been achieved.
    - …

File Formats:

- One of the triggers for having this big dream of a restructure is the recognition that the next time we break the scenario file format it is going to be broken in many ways, no simple usage of spare bytes as was common in the past. If you are going to break the format, why not replace it wholesale with something more flexible?
- The current file formats are extremely tied to how the data is stored in the game, that makes any changes in the game reflect in the save game file and eventually scenario format.
- The way forward is via abstraction, the data as represented in the game may be entirely different from how it is stored in a scenario or saved game file.
    - Prime example and probably model to follow, XML scenario and save file formats.

    - Intermediate parser in the game converts the XML format to something the game can use.
- This will all apply to the database format as well and presents one of the larger challenges with mixing database material of multiple types for use in one scenario.
    - i.e. AGSI Radar Model uses one format data while TGP Sonar Model uses a non-standard set of data, merging will be ‘interesting’ at best.
- Things the Scenario file must dictate
    - Which sensor models and revisions are used
    - Which effects and revisions are used
    - Which databases and revisions are used
    - For all of the above, whether a newer revision of the same model can be substituted
    - There is no option to use older revisions of the same model, it just isn’t allowed

Detail on the ‘game second diffs’:

- At the GE a game second starts
    - The GE maintains the entire previous game second’s state
    - The GE maintains the entire current game second’s state
    - The GE maintains the entire next game second’s state
    - Cycle
        - The GE finds the differences between the current and previous game second states and sends them to each Player.
        - Players enter any commands for the game second, they are returned to the GE
        - GE applies players commands
        - GE executes Effects to complete the next game second state at which point it becomes the current game second, current game second becomes previous.
        - Repeat cycle

Async requests from Player to GE (a subset):

- Weather data at a certain point or between points
- Topographic data at a point or between points
- Platform display information
- Chat with other Players

Challenges:

- 200,000+ lines of code (200,247 in the quick count I just did) to translate into a different language.
- Marginal initial benefits, the game would pretty much play the same at the end of the conversion, somewhat slower and with a slightly better UI.
- Given the above, how to justify going farther than dreaming about this?
    - Builds a firm foundation for much more explosive and beneficial future improvements.

    - Potentially opens interfaces to public consumption to keep the game alive long after official development ends (look at this model and you'll see that much of the game can be replaced)
- Making the end result useable, multiple TCP servers spells firewall nightmares, running potentially tens of programs just to play a scenario.
- Making sure the file formats are flexible enough and linked in the proper places to allow the modularity to work.
- Creating a system (the hallowed HC Launcher program) to help distribute, download, consume the various pieces needed to run one of these scenarios (which may dictate custom database information, replacement sensor models, new/modified effects, etc.).
- What defines HC, and how do we not lose that?
    - This restructure envisions huge changes, would HC still be HC if the restructure happened?

What doesn't work or isn't defined:

- How do weather and topographic services work, are they also servers available to each sensor model, and each Effect (MoveUnits)?
- Is the UI protocol dynamic, can a user-added Effect such as Lasers feed new menu items to the UI and change the user experience in that manner, a new explosion graphic for instance?
- How is each sensor laid out, is it its own IP server that could in theory handle 10 different GEs each using a different database (i.e. Tony runs a Radar server for anyone to use and it may accommodate 5 scenarios running across different databases at any one time)?
- Is this restructure a property of Stratsims and application for HC is just one development of it (what this restructure is describing is almost entirely a generic game model in which you could plug Harpoon, or TacOps, or War at Sea…?

Well, I think that'll do for now, more ideas will surely come, until then I'm very interested in what anyone reading this has to say. Please respond at the forum here or e-mail if you are too shy for that or represent a really large contract to make this happen.

Sincerely,

Tony Eischens
sysop@tgp.net