NAME

kh − Kepler's Horizon game server: a two-player tactical game of interstellar combat, fleet command, and economic conquest

SYNOPSIS

kh [*OPTIONS*]
kh −−help
kh −−version
kh −−port *PORT* −−dbusr *USER* −−dbpass *PASS* −−dbname *DB* [−−dbhost *HOST*]
kh −−ai *PATH* [−−log *FILE*] [−−monitor]

DESCRIPTION

Kepler's Horizon is a deterministic, two-player tactical game of interstellar warfare. Players build individualized warships, navigate a hex-based star map connected by warplines, and resolve combat through strategic power allocation—without dice. The objective is to occupy enemy base star hexes and accumulate victory points.

The game supports single-player mode against an autonomous AI opponent implemented in Common Lisp via the Embedded Common Lisp (ECL) runtime. The AI plays by the same rules as human players, issuing the same commands through the same game engine.

This manual is the authoritative reference for the server, its command language, the AI agent architecture, the economic system, and the Milieu Codex of Stars.

A note on design philosophy. Kepler's Horizon is intentionally minimal. Strategic depth emerges from constrained movement, simultaneous decision-making in combat, and irreversible consequences. Equal inputs always yield equal outcomes. The rules favor clarity over complexity.

Syntax of the Manual

The game accepts text input from the user. The game is designed to treat text case-insensitive. The UI enforces this by making everything look upper case. Throughout this manual page where lower or mixed case appears, it is assured that the game itself is case-insensitive. The grammar is more important for command invocations. Case is irrelevant.

A Note About the Origin and Development

The game idea was a mixture of my own game developed back in 1984 and influenced heavily by other space-war games, such as WarpWar (1977). The implementation of the software of this game was designed and architected by a human, but some features were initially implemented by AI (Opus 4.6 via Anthropic/Claude).

The author makes no apologies about using AI to develop parts of the software, as this was helpful in speeding up the process. Regardless, all of the software had to be revised anyway. Further, the core engine of the parser (**lex** and **yacc**) was hand-written.

The *LISP* code used by the AI-Agency was mostly designed with AI support, but the ontology was designed and implemented by human work. Again, the author has no qualms about this because it is effectively the same as spending days researching. The reader may disagree.

After 45 years of writing software, the author can attest that fundamental breakthroughs in some kinds of software projects are greatly aided by AI-assisted engineering.

This is the way

OPTIONS

Generic Program Information

−−help Print a usage summary and exit.

−v, **−−version**
Print the version number and git SHA, then exit.

Database Connection

−H, **−−dbhost** *HOST*
MySQL server hostname. Default is **127.0.0.1**.

−u, **−−dbusr** *USER*
MySQL user name.

−d, **−−dbname** *DB*
MySQL database name (typically **khdb**).

−p, **−−dbpass** *PASS*
MySQL password.

Network

−P, **−−port** *PORT*
HTTP port for the REST API. Default is **8080**.

AI Configuration

−−ai *PATH*
Path to the directory containing AI DSL files (Common Lisp sources). Default is **dsl** relative to the working directory. The server loads twelve Lisp files from this directory on first single-player game start: *aa-macros.lisp*, *aa-theta.lisp*, *aa-entities.lisp*, *aa-util.lisp*, *aa-strategy.lisp*, *aa-goals.lisp*, *aa-rules.lisp*, *aa-economic.lisp*, *aa-build.lisp*, *aa-movement.lisp*, *aa-combat.lisp*, *aa-core.lisp*.

Logging

−L, **−−log** *FILE*
Write server log output to *FILE*.

Monitor Mode

−M, **−−monitor**
Enable terminal command input on the server console.

GETTING STARTED

Web Portal

Login or create an account at the web portal. The portal serves static HTML pages and communicates with the game server through a REST API.

Lobby

The lobby displays open game rooms. A player can create a new room (which selects a universe module) or join an existing room by taking seat A or seat B.

Starting a Game

When both seats are filled, either player may start the game. In single-player mode, an AI user occupies the second seat automatically.

Module Selection

The game host selects the universe module at room creation time. A **Module** defines a complete game universe: map topology, star systems, planets, species, resources, anomalies, and facilities. The game engine is module-agnostic; only content varies between modules.

The default module is **Kepler's Horizon**: 28 star systems across Alpha and Beta territories, with a Contested Reach between them.

MAP COMPONENTS

The game map is a hex grid overlaid with named star systems and warpline connections. Not every hex contains a star.

Star Hex

A map hex containing a star system (large or small). Star hexes are the nodes of the warpline graph. Ships must stop at enemy-occupied star hexes.

Base Star Hex

A star hex containing a large star. There are six base stars total, three per side (Alpha and Beta). Occupying enemy base stars earns victory points.

Space Hex

A hex with no star and no warpline. Ships may pass through freely.

Warp Line

A bi-directional edge connecting two star hexes. Traversing a warpline costs 1 PD regardless of distance. Warplines pass through intermediate hexes but those hexes are treated as space hexes for movement purposes.

SHIP TYPES AND ATTRIBUTES

Ship Types

WarpShip (W1−W9)

Capital ships capable of interstellar travel via warplines. WarpShips carry a Warp Generator (5 CR, mandatory) and may equip SystemShip Racks to carry escort vessels. Hull codes are W followed by a single digit: W1, W2, ... W9. A fleet may contain up to 9 WarpShips.

SystemShip (S01−S99)

Escort vessels that operate within a single star hex. SystemShips have no Warp Generator and cannot traverse warplines. They must be carried by WarpShips in SystemShip Racks. Hull codes are S followed by one or two digits: S1, S11, S25, etc.

Ship Attributes

Every ship has the following combat-relevant attributes, each costing Credit Points (CR) at construction time:

Attribute	Code	Cost	Description
Power/Drive	PD	1 CR/unit	Engine strength; powers movement and combat
Warp Generator	WG	5 CR	WarpShips only (automatic, mandatory)
Phasic	P	1 CR/unit	Energy weapon; damage = allocation + tech level
Shield	S	1 CR/unit	Energy shield; absorbs = allocation + tech level
Launcher	L	1 CR/launcher	Launches 1 torpedo per combat round
Torpedoes	T	1 CR/3	Projectile weapons; damage = 2 + tech level
SystemShip Hangar	H	1 CR/hangar	Carries 1 SystemShip per hangar (WarpShips only)

Additionally, ships may be equipped with economic hardware:

Equipment	Code	Cost	Purpose
Long Range Scanner	LRS	50 CR	Required for **extract scan** and **survey**

Equipment does not affect combat. A ship is destroyed only when all combat attributes (PD, P, S, L, T, H) are reduced to zero. The Warp Generator never takes damage.

Technology Levels

Ships retain their original tech level permanently. Tech level is determined by the game round at construction time:

Rounds	Tech Level
1−4	Level 0
5−8	Level 1
9−12	Level 2

13−16 Level 3
17+ +1 per 4 rounds

Tech level adds to:

- Phasic damage (when hitting)
- Torpedo damage (base 2 + tech level)
- Shield absorption (when powered)

A Tech Level 2 ship with Phasic 3 allocated to phasics deals 5 damage on a hit. A Tech Level 2 ship with Shield 2 allocated absorbs 4 hits.

GAME SEQUENCE

Each player-turn consists of the following phases, in order. The command**next** (alias **n**) adv ances to the next phase. The command **done** (alias **ZZ**) ends the current player's turn and passes initiative to the opponent.

1. **Count Victory Points.** At the start of each player-turn, the system awards 1 VP for each enemy base star hex occupied by your ships.
2. **Build Ships.** Recei ve Stellar Credits (CR) and spend Credit Points (CR) to construct new ships, repair damaged ships, resupply torpedoes, and manage equipment.
3. **Movement.** Move WarpShips along hex paths and warplines. Pick up and drop off SystemShips. Deploy newly built ships to your base star hexes.
4. **Resolve Combat.** At e very star hex where opposing ships coexist, combat is triggered. Players issue simultaneous orders, resolve fire, assign damage, and handle retreats.
5. **SystemShip Pick and Drop.** Free rearrangement of SystemShips at star hexes. No PD cost. Active player only.
6. **End of Turn.** Finalize the turn. Market prices update after both players complete a round.

COMMANDS

Commands are case-insensitive. Short aliases are shown after commas. All commands are entered in the console text field on the game page.

Session Commands

save *name*
: Save the current game state under *name*. The name must match the lexer pattern **[a−zA−Z][−a−zA−Z0−9]***: letters, digits, and dashes only. No underscores, dots, or spaces. Without arguments, shows usage.

load *name*
: Propose loading a previously saved game. This initiates a two-factor confirmation: the other player must type **accept** *name* to confirm. Without arguments, lists available saves.

accept *name*
: Accept a pending load request from the other player. The game state is restored from the saved snapshot.

reject *name*
: Reject a pending load request.

delete *name*
: Delete a saved game permanently.

quit Exit the current game. The game is auto-saved as **autosaved-YYYY-MM-DD-HH-MM**. Combat state is nullified (ships remain in position; combat will re-trigger on reload). The other player receives a notification and is redirected to the lobby. The AI thread's game context is zeroed so it naturally ceases operating on this game.

clear, **cls**
Clear the console display.

Turn Management

next, **n** Advance to the next game phase. Blocked if there are pending retreats or unresolved damage.

done, **ZZ**
Complete the current player's turn and pass initiative to the opponent. Blocked if there are pending retreats.

Information Commands

score Display victory points and credits for both players, plus current round and tech level and display current game state: whose turn, phase, round, credits, tech level.

fleet Display a manifest of your entire fleet: every ship with its hull code, name, location, and current attribute values (PD, P, S, L, T, H, equipment).

cargo *ship*
Display the cargo manifest for *ship*: quantities of each resource type in the hold, cargo capacity, and torpedo storage.

hex *location*
Show information about hex *location*: which ships occupy it, which star system (if any), active hex events, and warpline connections.

system *name*, **sy** *name*
Show information about star system *name*. Subcommands:

system *name* anomalies
List anomalies at the system.

system *name* facilities
List facilities at the system.

system *name* resources
List extractable resources at the system.

system *name* planets
List planets and moons in the system.

survey *name*, **sv** *name*
Survey a star system to increase your knowledge level (see MILIEU CODEX). Requires a ship with LRS at the target system. Each survey operation upgrades knowledge by one level. Without arguments, shows survey status.

crt Display the Combat Results Table.

help *topic*, **?** *topic*
Show help for a command or topic. Without arguments, lists all available topics. Topics: general, build, move, combat, economy, info, turn, ship, crt, torpedoes.

recap Display a summary of recent game events.

Build Commands

build, **bb**
Show current build state: active drafts and available CR.

build new *class name*, **bn** *class name*
Create a new ship draft. *class* is W (W arpShip) or S (SystemShip), optionally followed by 1−2 digits for a specific hull number (e.g., W4, S11). If no number is given, the game auto-assigns the next available hull code. *name* is the ship designation (e.g., Falcon, Avenger).

Examples:
> BN W FALCON
> BN S11 AVENGER

> BUILD NEW W4 DESTROYER

build drafts, **bd**
List all pending build drafts.

build drafts *ship*, **bd** *ship*
Show detailed attributes of a specific draft (by code or name).

build set *target attributes*, **bs** *target attributes*
Set attributes on a draft ship. *target* is ship code or name. Attribute syntax:

PD=N **or** D=**N**
Power/Drive (1 CR per unit)

P=N Phasic weapon (1 CR per unit)

S=N Shield defense (1 CR per unit)

L=N Torpedo launchers (1 CR per launcher)

T=N Torpedoes (1 CR per 3 torpedoes)

H=N SystemShip hangars (1 CR per hangar, WarpShips only)

Example: BS FALCON PD=7 P=3 S=2 L=1 T=6

build commit *target*, **BC** *target*
Finalize a draft and add the ship to your fleet. CR is deducted. The ship appears in the fleet but must be deployed before it can act.

build cancel *target*, **bx** *target*
Cancel a draft. No CR is deducted.

Deployment Commands

deploy *ship system*, **ds** *ship system*
Deploy a newly built ship to a base star system.

The first deployment by either player claims a side of the map. The three base stars on that side become your home bases. Subsequent deployments must go to one of your claimed base stars.

Movement Commands

move *ship dest* [*via...*], **m** *ship dest* [*via...*]
Move a WarpShip to *dest*, optionally through intermediate waypoints. Waypoints may be star system names or hex IDs (e.g., h1310). Movement costs 1 PD per hex or per warpline jump.

Critical rule: WarpShips are forced to stop at any star hex occupied by enemy ships. Combat is triggered at the end of the movement phase.

PD expended for movement is *not* permanently reduced. Full PD is available for combat power allocation.

Example: M W6 TAVON XYLEN
W6 moves to Tavon, then onward to Xylen.

pick *systemship warpship*, **pu** *systemship warpship*
Load a SystemShip onto a WarpShip's rack. Costs 1 PD during the movement phase. Ships must be at the same hex.

drop *systemship warpship*, **do** *systemship warpship*
Unload a SystemShip from a WarpShip. Costs 1 PD during the movement phase.

Combat Commands

combat, **cm**
Show active combat status at all contested hexes.

combat order *ship tactic target allocation*, **co ...**
Issue a combat order for one of your ships.

Tactics:

a (attack)
Aggressive offense. Best at 0 to +2 drive advantage.

d (dodge)
Defensive maneuver. Good at +1/+2 drive, hard to hit at 0/−1.

e (escape/retreat)
Attempt to flee combat. WarpShips only. SystemShips may never select escape.

Allocation: PD=N P=N S=N L=N T=N

Phasic fire example:
> CO W3 A S25 PD=2 P=3 S=2
W3 attacks S25: 2 drive, 3 phasic, 2 shield.

Torpedo fire example:
> CO S25 D W3 PD=4 L=1 T=3
S25 dodges while firing 1 torpedo at W3 with torpedo drive=3. Each**T=** parameter specifies that torpedo's drive power. If two torpedoes were fires, the player would need to use two Launchers: **L=2** and then assign *each* Torpedo power: **T=3 T=2** for power assignments 3 and 2 respectively.

combat commit, **cc**
Lock in all pending combat orders. Once both players commit, the combat round resolves automatically.

combat cancel, **cx**
Cancel all pending combat orders.

combat drafts, **cd**
Show your pending (uncommitted) combat orders.

combat apply *ship allocation*, **ca** *ship allocation*
Assign damage to a ship's attributes after combat resolution.

Example: CA W3 PD=2 P=1 S=2
Distributes 5 total damage: 2 to PD, 1 to Phasic, 2 to Shield. The total assigned must equal the damage dealt (or consume all remaining HP if the ship is destroyed).

retreat *ship hex*, **rt** *ship hex*
Retreat a ship from combat to an adjacent hex. Retreat destinations are validated against the hex adjacency graph.

Repair and Resupply Commands

repair *ship attr=N*, **rp** *ship attr=N*
Repair a damaged ship. 1 CR per attribute point restored. Ship must start the turn at a friendly base star hex (or at a REPAIR_DOCK or SHIPYARD facility). Cannot repair beyond original built capacity. One attribute per command invocation.

Example: rp W3 pd=2

resupply *ship quantity*, **rs** *ship quantity*
Resupply torpedoes at 1 CR per 3 torpedoes (up to original capacity). Ship must be at a friendly base star hex.

Economy Commands

extract scan, **ex scan**
Scan the current system for harvestable resources. Requires a ship with LRS (Long Range Scanner) at the system. Yield is modified by your knowledge level (see MILIEU CODEX).

extract *ship resource*, **ex** *ship resource*
Extract a resource from the current system into the ship's cargo hold. Multi-word resources use spaces: **ex W1 rare earth**.

market, **mk**
Display current market prices with trend indicators (RISING, STABLE, FALLING).

market *resource*, **mk** *resource*
Show price history for a specific resource.

trade list, **tr list**
Show base trade prices for all resources.

trade buy *resource quantity*, **tr buy ...**
Purchase resources at the current market price, deducting credits.

trade sell *resource quantity*, **tr sell ...**
Sell resources from cargo at 75% of market price.

trade transfer *from to resource qty*, **tr ts ...**
Move resources between ships at the same location.

fabricate list, **fab list**
Show available fabrication recipes and their material costs.

fabricate *recipe* [*quantity*], **fab** *recipe ...*
Manufacture items from raw materials in cargo.

outfit *ship* lrs, **out** *ship ...*
Install equipment on a ship.

outfit list, **out list**
Show available equipment and their credit costs.

salvage scan, **jk scan**
Scan for salvageable objects at the current hex. Discovery chance is modified by knowledge level.

salvage *ship*, **jk** *ship*
Conduct a salvage operation at the current hex. Hazard chance is modified by knowledge level.

Development Commands

gamedev, **gd**
Show current development override status.

gamedev *subcmd value*, **gd** *subcmd ...*
Set development overrides. Subcommands: env, combat, move, crt.

gamedev reset, **gd reset**
Clear all development overrides.

COMBAT RESOLUTION

Combat is the heart of Kepler's Horizon. It is entirely deterministic: given the same orders, the same results will always occur. There are no random rolls.

Combat Trigger

Combat is triggered when opposing ships occupy the same star hex at the end of the movement phase. The system creates a **combat_state** record for that hex and notifies both players.

Combat Stages

Combat at each hex proceeds through four stages in a cycle:

Stage	Name	Description
0	ORDERS	Both players issue orders for their ships
1	RESOLVE_READY	System resolves the round (automatic)
2	DAMAGE_PENDING	Players assign damage to their ships
3	RETREAT_PENDING	Stalemate: initiative player must withdraw

The Combat Results Table (CRT)

The CRT is the core lookup table that determines whether a weapon strikes its target. It is indexed by three values:

1. Firing ship's tactic (Attack, Dodge, or Retreat)
2. Target ship's tactic
3. Drive Difference = Firing Drive − Target Drive

Firing	Drive Diff	vs Attack	vs Dodge	vs Retreat
Attack	−3 or less	Miss	Miss	Escapes
Attack	−1, −2	Hit	Miss	Escapes
Attack	0, +1	Hit+2	Miss	Miss
Attack	+2	Hit+1	Hit+1	Miss
Attack	+3, +4	Miss	Hit	Hit
Attack	+5+	Miss	Miss	Miss
Dodge	−4 or less	Miss	Miss	Escapes
Dodge	−2, −3	Miss	Hit	Escapes
Dodge	0, −1	Hit	Hit	Escapes
Dodge	+1, +2	Hit	Miss	Escapes
Dodge	+3+	Miss	Miss	Escapes
Retreat	−2 or less	Miss	Miss	Escapes
Retreat	−1, 0	Hit	Miss	Escapes
Retreat	+1+	Miss	Miss	Escapes

Observe:

- Attack vs Attack at 0/+1 drive difference yields Hit+2—the deadliest intersection. Aggressive players who match drives annihilate each other.
- A drive difference of +5 or greater is always a miss—the attacker overshoots. This prevents dump-stat strategies.
- Dodge is safest at 0/−1 vs Attack (miss), but vulnerable at 0/−1 vs Dodge (hit).
- Retreat succeeds ("Escapes") against any tactic at favorable drive differentials, but fails if the enemy's drive advantage is too high.

Power Allocation

Each combat round, a player allocates their ship's PD to subsystems:

Drive (D)
: Maneuverability. Determines the row/column on the CRT. Higher drive = better chance to hit or dodge, but diverts power from weapons and shields.

Phasic (B)
: Weapon power. Cannot exceed the ship's built phasic rating. Damage on hit = phasic allocation + tech level.

Shield (S)
: Defensive shield. Cannot exceed the ship's built shield rating. Absorption = shield allocation + tech level.

Launchers (T)
: Number of torpedoes to fire. Each launcher requires 1 PD.

Constraint: D + P + S + L ≤ current PD.

Critical rule: Phasics/Shields and Torpedoes cannot be used in the same round. If you fire torpedoes, you cannot power phasics or shields.

Phasic Damage

When a phasic attack hits:

1. Damage = Phasic power allocation + Tech level
2. If target powered shields: absorption = Shield allocation + Tech level

3. Net damage = max(0, damage − absorption)

Torpedo Damage

Each torpedo is individually resolved:

- Torpedoes always use the **Attack** tactic on the CRT
- Each torpedo has its own drive setting (specified via **T=N**)
- Damage per hitting torpedo = 2 + Tech level
- Torpedoes are expended on firing (regardless of hit/miss)
- Each launcher fires at most 1 torpedo per round

Damage Assignment

After resolution (Stage 2), each player must assign their ship's damage to specific attributes using **combat apply**.

- Total assigned must equal damage dealt
- If damage exceeds remaining HP, the ship is destroyed (overkill accepted)
- Ship is destroyed when PD + P + S + L + T + H = 0
- The Warp Generator is never damaged

Retreat

A ship using the Escape tactic must receive "Escapes" results against *all* enemy ships firing at it. If any attacker rolls a Hit, the retreat fails and the ship remains in combat.

On successful retreat, the ship is flagged **escape_pending**. The player must issue**r etreat** *ship hex* to move the ship to an adjacent hex. The turn is blocked until all retreats are completed.

SystemShips may never retreat. They must be carried out by a WarpShip using the Pick/Drop mechanism.

Stalemate

If three consecutive combat rounds produce zero unabsorbed damage (stalemate_counter >= 3), the initiative player (who moved into the hex) must withdraw all their ships from the hex. The defender retains control.

Combat End Conditions

Combat at a hex ends when:

- All ships of one side are destroyed or retreated
- Stalemate forces the initiative player to withdraw

MOVEMENT RULES

1. WarpShips **must stop** at star hexes occupied by enemy ships.
2. WarpShips may move through space hexes with enemy ships.
3. Warplines are treated as space hexes for movement purposes. Entering a warpline at one end and exiting at the other costs 1 PD.
4. Ships with PD=0 cannot move.
5. PD used for movement is **not permanently expended**. Full PD is available for combat power allocation. This is a critical rule: movement does not weaken your ships for battle.
6. Cannot move onto enemy base star hex during turn 1.
7. Hex events (NAVIGATION_HAZARD) may increase movement cost through specific hexes.

The server uses BFS pathfinding to calculate minimum PD expenditure. Players may specify intermediate waypoints to force a specific route.

VICTORY CONDITIONS

Dominance Victory
Accumulate 3 victory points by occupying enemy base stars at the start of your turn. Each enemy base star hex with at least one of your ships earns 1 VP.

Elimination Victory
Destroy all enemy ships.

Draw Neither player has effective ships remaining.

THE ECONOMY

The economic layer adds resource management, trading, and manufacturing to the tactical combat gameplay.

Currency: Stellar Credits (CR)

All economic transactions use Stellar Credits. Players earn credits from trade, controlling TRADE_HUB facilities (+5 CR/turn), and selling extracted resources.

Resources

Eight types of raw materials exist in the universe. They are found in planets, moons, and asteroid belts, with varying abundance and extraction difficulty.

Resource	Abbrev	Base Price	Primary Uses
FERROUS	Fe	5 CR	Hull construction, fabrication
RARE_EARTH	Re	20 CR	Electronics, shields, tech
RADIOACTIVE	Rd	30 CR	Power, weapons, fabrication
CRYSTALLINE	Cr	25 CR	Phasics, warp drives, tech
VOLATILE	Vo	8 CR	Fuel, torpedoes, fabrication
WATER	H2O	3 CR	Life support
ORGANIC	Or	6 CR	Food, medicine
EXOTIC	Ex	100 CR	Special technology, advanced fabrication

Resource Abundance

Each deposit has an abundance rating that determines base yield:

Abundance	Base Yield
Rich	16 units
High	8 units
Moderate	4 units
Low	2 units
Trace	1 unit

Extraction difficulty further modifies yield:

Difficulty	Modifier
Easy	100%
Moderate	70%
Difficult	40%
Extreme	20%

Knowledge level at the system also modifies extraction yield (see MILIEU CODEX section).

Market Dynamics

The commodity market fluctuates based on supply and demand:

- Prices update each round after both players complete turns
- Buying increases demand, pushing prices up
- Selling increases supply, pushing prices down
- Prices are clamped to 50%−200% of base value
- Price trends are displayed as RISING, STABLE, or FALLING

Fabrication Recipes

Raw materials can be converted into ship upgrades and munitions:

Recipe	Output	Time	Material Cost
torpedoes	4 torpedoes	1 round	2 Fe, 1 Rd, 1 Vo
launchers	+1 launcher	3 rounds	5 Fe, 3 Re, 2 Cr
phasics	+1 phasic	3 rounds	8 Fe, 4 Re, 3 Cr
shields	+1 shield	3 rounds	6 Fe, 2 Re, 4 Cr
tech	+1 tech level	5 rounds	10 Re, 5 Cr, 2 Ex

Salvage

Wreckage and debris at hex locations can be salvaged for resources.

Discovery: Scanning (**salvage scan**) finds salvageable sites. Discovery chance is modified by knowledge level at the system. Discovered sites are remembered permanently.

Hazards: Each salvage operation carries a hazard chance (typically 5−25%). Hazard types include unstable reactors, booby traps, structural collapse, and hostile salvagers. Hazard damage is dealt to PD. If PD reaches 0, the ship is destroyed. Knowledge level reduces hazard chance.

Yields: Military wrecks yield torpedoes and rare electronics. Commercial wrecks yield common resources and credits. Ancient derelicts yield exotic materials. Debris fields yield ferrous and basic materials. Sites deplete after multiple operations.

Facilities

Facilities at star systems provide strategic advantages to the controlling player. Control is established by occupying a system with no enemy ships for 2 consecutive turns.

Facility	Effect
SHIPYARD	Build new ships, fabrication base
REPAIR_DOCK	Repair damaged ships away from home base
REFINERY	Fabrication bonuses
TRADE_HUB	+5 CR income per turn, trade access
FORTRESS	Combat defensive bonus

THE MILIEU CODEX

The **Milieu Codex of Stars** is the compendium of known space. It contains detailed information about each star system in the game universe: celestial bodies, planetary environments, native species, resources, anomalies, and strategic notes.

The Codex is not merely flavor text. **Knowledge level directly affects gameplay outcomes.** A commander who surveys a system before exploiting it will extract more resources, discover more salvage sites, and suffer fewer hazards than one who operates blind.

Knowledge Levels

Each player tracks knowledge independently per system. Knowledge starts at **Unknown** and is improved by surveying.

Level	Description	How Gained
Unknown	Name only; no actionable intelligence	Default
Charted	Basic characteristics; partial star data	First visit
Surveyed	Detailed analysis; full resource data	**survey** command
Intimate	Complete secrets; hidden sites revealed	**survey** (second pass)

Gameplay Impact of Knowledge

Extraction Yield Modifiers:

Level	Yield	Explanation
Unknown	25%	Prospectors operate blind
Charted	50%	Basic geological data improves targeting
Surveyed	100%	Precision extraction with detailed surveys

Intimate	100%	No additional bonus beyond Surveyed

Salvage Discovery Modifiers:

Level	Discovery Chance	Explanation
Unknown	50% of base	Searching blind
Charted	75% of base	General debris fields mapped
Surveyed	100% of base	Precise wreck catalogs available
Intimate	125% of base	Insider knowledge of concealed salvage

Salvage Hazard Modifiers:

Level	Hazard Modifier	Explanation
Unknown	+20%	Operating blind in debris fields
Charted	+10%	Partial hazard mapping
Surveyed	+0%	Standard risk assessment
Intimate	−10%	Know which wrecks are dangerous

Knowledge is **faction-wide**: all your ships benefit immediately when knowledge is gained at a system.

Territories

Systems are organized into three territories:

Western Alliance (Alpha Faction) — Home territory of the Terran alliance.

System	Hex	Star	Key Features
ARVEN	0307	G2V Yellow	Capital. Garden world Verdance. The Anvil shipyards.
BELIX	0606	Binary K1V+M4V	Ancient system. Haven cultural center. Whisper Gate anomaly.
CAYRU	0804	F8V Yellow-White	Young star. The Shatter belt, rich in metals.

Eastern Powers (Beta Faction) — Territory of the Zarekites and their allies.

System	Hex	Star	Key Features
ZAREK	2125	A3V White	Zarekite homeworld. Servitor shipworks. The First Library.
ASTREX	2223	G5V Yellow	Hanging Gardens. Commerce trade zone. Ishtar Gate anomaly.
BRION	2622	K5V Orange	Ancient depleted star. Dagan's Rest breadbasket.

Contested Reach — Disputed systems between the powers.

System	Hex	Star	Key Features
KORAL	1310	G8V	The Axis. 4 warpline connections. Axis Marker beacon.
QUELL	1616	M2V Red	Lithoid xenoforms. Keplerite exotic warp-crystals.
SYDRA	1719	K2V Orange	War-torn. The Graveyard and Scrapyard salvage sites.
VARYN	1922	G1V Yellow	The Remnant machine civilization.
WEXAR	2020	F7V	Kethrani aquatic traders. Water worlds.
XYLEN	2118	F5V	Nexus free port. Mercenary strongholds.
TAVON	1817	M1V Red	Vesh hive mind. The Song anomaly.

Xenoform Species

Species	Homeworld	Traits
Terran Human	Multiple	Adaptable, ambitious colonizers
Zarekite	ZAREK	Light-adapted, hierarchical, ancient tech
Lithoid	QUELL	Silicon-based, expert miners
Kethrani	WEXAR	Aquatic, skilled traders
Vesh	TAVON	Hive mind, subsonic communicators
The Remnant	VARYN	Ancient AI collective
Wanderers	Fleet-born	Nomadic traders, expert shiphandlers

Anomalies

Some systems contain unexplained phenomena with potential strategic implications:

Anomaly	System	Description
Whisper Gate	BELIX	Ancient structure; energy readings persist
Core Fragment	CAYRU	Exotic matter in The Shatter belt
First Library	ZAREK	Elder Race data storage
Ishtar Gate	ASTREX	Metal ring, 20m across, ancient texts mention it
Axis Marker	KORAL	Navigation beacon, warpline efficiency
Keplerite Vein	QUELL	Warp-enhancing exotic crystals
The Graveyard	SYDRA	Powered wreckage field, high salvage value
The Song	TAVON	Vesh vibrations that penetrate vacuum

Notable Resource Deposits

Resource	Location	Abundance
FERROUS	The Anvil (ARVEN)	Rich
FERROUS	The Shatter (CAYRU)	Rich
FERROUS	Servitor (ZAREK)	Rich
RARE_EARTH	Jewel (BELIX)	Rich
RARE_EARTH	The Shatter (CAYRU)	Rich
VOLATILE	Shepherd (ARVEN)	Rich
VOLATILE	Cloud Marshal (BELIX)	Rich
ORGANIC	Verdance (ARVEN)	Rich
ORGANIC	Hanging Gardens (ASTREX)	Rich
EXOTIC	Keplerite (QUELL)	Rich
EXOTIC	Gilded Throne ruins (ZAREK)	Rich
WATER	Verdance (ARVEN)	Abundant
WATER	Hanging Gardens (ASTREX)	Abundant
CRYSTALLINE	Glassine (XYLEN)	Rich

AI AGENT

Kepler's Horizon supports single-player mode through an autonomous AI opponent called the **Autonomy Agency** (Generation 4). The AI replaces one human player and executes commands through the same game engine, respecting all rules and constraints. It issues the same textual commands a human player would type, funneled through the same `yyparse()` grammar.

The AI thread runs independently of HTTP request handling. When a game ends (via **quit**, victory, or elimination), the AI's game context is zeroed—it is **never killed**. The thread stays alive and naturally ceases operating when it detects **m_game_id = 0** on its next gather cycle.

Architecture: Mealy State Machine

The AI operates as a closed-loop Mealy state machine with three phases per decision cycle:

1. GATHER (C++)
   Populate the **AASlate** from the database and StateMachine. The Slate is a complete snapshot of observable game state: fleet positions and attributes, combat theaters, economic data, territory control, BFS distance matrix, hex adjacency, warpline topology, codex knowledge levels, market prices, facility control, salvage sites, and persisted cross-turn metrics. Nothing is hidden from the AI that a human player could see.

2. CALCULATE (Lisp)
   Marshal the slate to Common Lisp via ECL (Embeddable Common Lisp) and call **(aa-calculate slate)**. The Lisp layer computes a strategic state once, then dispatches to a phase-specific decision pipeline that fires named rules, evaluates a goal graph, and returns a list of command strings plus optional metric write-backs.

3. RENDER (C++)
   Inject each resulting command into the game engine via **AICommandInjector**, exactly as if a human typed it. The `TaskRunner` thread executes the command through `yyparse()` and blocks until completion. The AI observes the result and feeds it back into the next GATHER cycle, making the loop adaptive: failed commands change the slate, and CALCULATE derives a different action on the

next iteration.

The cycle repeats (gather, recalculate, render) until the AI issues a terminal command (**NEXT** or **DONE**) to advance the phase or end the turn. Every decision is based on current game state, never stale assumptions.

A safety guard at the top of **gather()** checks **m_game_id <= 0** and sets **game_over = true**, causing the Mealy loop to exit gracefully without querying the database.

Generation 4 DSL Architecture

The Generation 4 AI is a declarative, rule-driven system built on three macro primitives defined in **aa-macros.lisp**:

defentity Declares an entity type (ship, combat, resource, facility, etc.) and auto-generates typed accessor functions. Entity fields support aliases, defaults, transforms, and derived computations.

define-strategy-rule
Registers a named decision rule with a phase tag, priority, guard predicate (**:when**), and action body (**:action**). Rules are dispatched by a rule engine that evaluates guards in priority order and fires the first match.

defgoal Declares a goal node in a dependency graph rooted at **:win-game**. Each goal carries a priority, a parent link (**:supported-by**), satisfaction and precondition predicates, an action function, and a resource-needs declaration. The goal graph is validated at load time to ensure all goals trace back to the root with no cycles.

All Lisp functions are pure: slate in, decisions out. No function mutates itself or carries hidden state. This makes every decision deterministic, debuggable, and auditable.

Strategy DSL Modules

The AI strategy is implemented across twelve Common Lisp modules, loaded in strict dependency order by the C++ runtime:

Order	Module	Purpose
1	aa-macros.lisp	DSL infrastructure: defentity, define-strategy-rule, defgoal
2	aa-theta.lisp	Parameter vector: all tunable constants (defparameter)
3	aa-entities.lisp	Entity ontology: ship, combat, resource, facility, etc.
4	aa-util.lisp	Slate accessors, ship finders, predicates, command helpers
5	aa-strategy.lisp	Strategic state computation: posture, theaters, force projection
6	aa-goals.lisp	Goal dependency graph and MPC evaluation engine
7	aa-rules.lisp	Named decision rules across all phases (build, move, combat, etc.)
8	aa-economic.lisp	Economic action generators: survey, extract, trade, fabricate
9	aa-build.lisp	Ship design, construction, repair, deployment
10	aa-movement.lisp	Movement routing, threat detection, target assignment
11	aa-combat.lisp	Combat triage, CRT-aware tactics, damage assignment
12	aa-core.lisp	Entry point and phase dispatcher

The Strategic State

At the start of each CALCULATE cycle, **compute-strategic-state** runs once and produces a strategy plist consumed by all downstream decisions. The strategic state comprises:

Victory State
Bases held, bases needed, score comparison, VP race projection, endgame detection (either side within one base of winning).

Fleet State
Ship counts by role, total combat attributes, weighted combat power (PD + 1.5B + 1.2S + 0.5T + 0.3M scaled by health ratio), and fleet composition analysis.

Enemy Profile
Enemy ship count, positions, maximum observed tech, torpedo capability, shield density, and estimated total CR investment. Cross-turn metrics track historical enemy positions and watermarks from

combat observations.

Hex Valuation
Each hex receives a strategic score: enemy base (100), own base (80), facility bonuses (shipyard 30, refinery 20, repair 15, trade 10), proximity to bases, warpline chokepoints (25), and VP-occupied bonus (40).

Force Projection
Undefended enemy bases, bases under threat, reinforcement availability, convergence points (hexes reachable by 2+ friendly ships), and coverage gaps (own bases outside friendly reach).

Enemy Intent
Estimated convergence target, threat velocity per base, inferred spending rate, and strategy classification (aggressive, defensive, tech-investing, balanced).

Temporal State
Turns until next tech level, game phase (early/mid/late), credit trend, power trend, fleet attrition rate, and 3-turn economic trajectory.

Posture
Result of firing posture rules (last-match-wins semantics). One of: aggressive, defensive, or balanced. Determines acceptable loss thresholds and build/movement directives.

Phase Dispatch
The entry point **(aa-calculate slate)** computes the strategic state, then dispatches to the current phase:

Build (Phase 0)
The rule engine fires build rules in priority order: commit pending drafts, set ship specifications from design templates, deploy completed ships, repair damaged ships at bases, evaluate economic goals, build defenders for unguarded bases, build new WarpShips. Falls through to **NEXT** when no rule produces an action.

Movement (Phase 1)
Checks threatened bases first (enemy within 3 hops) and dispatches defenders. Then moves idle ships toward enemy bases, spreading across targets to maximize VP coverage. Uses precomputed BFS distances from the slate.

Combat (Phase 2)
Triages all active theaters simultaneously. Each theater is assessed as fight, hold, or retreat based on force ratio and hex value. Ships needing retreat are handled first, then pending damage assignment, then combat orders, then commit.

Pick/Drop (Phase 3)
Rearranges SystemShips at contested or strategic hexes.

End Turn (Phase 4)
Issues **DONE** to pass initiative.

The Goal Graph
Economic decisions are driven by a directed acyclic graph of goals rooted at **:win-game**. Each goal declares a priority, a satisfaction predicate, a precondition predicate, and an action function. The MPC evaluator walks goals in priority order, finds the highest-priority goal that is both unsatisfied and actionable (precondition met), and executes its action.

Goals compose through dependency: a goal's action may require sub-goals to be satisfied first. The **:needed-resources** declaration prevents the sell-excess logic from liquidating cargo that serves an active goal.

The current goal graph:

Goal	Pri	Parent	Trigger
:maintain-torpedo-stock	1	:win-game	Fleet torpedoes below 50%
:resupply-torpedoes	2	:maintain-torpedo-stock	Ship on base with depleted torpedoes
:fabricate-materiel	3	:maintain-torpedo-stock	Ship at shipyard with ingredients
:acquire-resource	4	:fabricate-materiel	Missing fabrication ingredient
:survey-for-yield	5	:acquire-resource	System knowledge below Surveyed
:salvage-opportunity	6	:win-game	Ship at hex with salvageable wreckage
:sell-excess-cargo	8	:win-game	Cargo serving no active goal, above price floor

The Rule Engine

Strategy rules are registered via **define-strategy-rule** with a phase tag (**:build**, **:move**, **:combat**, **:triage**, **:tactics**, **:design**, **:posture**, **:pickdrop**), a priority (lower fires first), a guard predicate, and an action body.

Three dispatch modes:

fire-first-matching-rule
: Evaluates guards in priority order; fires the first that returns true. Used for combat orders and pick/drop.

fire-first-producing-rule
: Same, but the action must return a non-nil result. Used for build and movement where a rule may match but produce no command (e.g., build rule matches but player cannot afford the ship).

fire-posture-rules
: Evaluates all rules; last match wins. Used exclusively for posture computation where later rules override earlier ones.

Build Strategy

The AI selects from predefined ship templates defined in **aa-theta.lisp**:

Template	Attributes	Role
Brawler	PD=6 P=4 S=3	Main battle ship; default first build
Interceptor	PD=5 P=2 S=2	Fast pursuit; spreads across enemy bases
Torpedo Boat	PD=4 S=1 L=2 T=6	Alpha strike; counters heavy shields
Fortress	PD=8 P=6 S=5	SystemShip for base defense
Defender	PD=6 P=4 S=3	SystemShip garrison

Design selection is driven by design rules: the first build is always a Brawler; torpedo boats are chosen when the enemy fields heavy shields; interceptors fill the fleet for base coverage; defenders garrison unguarded home bases.

Build heuristics gate construction on CR availability, tech-level timing (the AI may delay a build if tech upgrades within 1−2 turns), and fleet size relative to the enemy.

Combat Strategy

Combat operates at two levels: theater triage and per-ship tactics.

Theater Triage. All active combats are assessed simultaneously before any orders are issued. Each theater receives a verdict (fight, hold, or retreat) based on force ratio, hex strategic value, and stalemate count. VP hexes are fought more aggressively; non-VP hexes with unfavorable force ratios are abandoned.

Tactics. Per-ship orders are generated by tactics rules:

- Focus fire: all ships in a theater target the same enemy (lowest total HP).
- CRT-aware power allocation caps drive differential at +4 to avoid the +5 automatic miss. Three allocation modes: standard attack (target diff +2), pursuit (+4), and counter-dodge (+3).
- Torpedo boats fire when they have stock; retreat when empty.
- Stalemate breaker: if the AI is the attacker and stalemate count reaches 2, it forces maximum damage output or retreats.

- Damage assignment sacrifices attributes in order: launchers, phasics, shields, PD (preserved last for retreat capability).

Economic Strategy

Economic actions are driven by the goal graph. The MPC evaluator selects the highest-priority unsatisfied goal and dispatches its action generator. Action generators map goals to concrete commands:

- **Resupply:** reload torpedoes at own base.
- **Fabricate:** convert raw materials to torpedoes at a shipyard (recipe: 2 Fe + 1 Rd + 1 Vo).
- **Acquire:** extract resources from a deposit or buy them at a trade hub.
- **Survey:** improve system knowledge to boost extraction yield.
- **Salvage:** recover materials from wreckage.
- **Sell:** liquidate excess cargo that serves no active goal, only if the market price exceeds 80% of base value.

The Theta Parameter Vector

All tunable numeric constants (thresholds, weights, fleet caps, template attributes) are declared as **defparameter** values in **aa-theta.lisp** and accessed via **(theta 'theta-xxx)**. An unknown key raises an error at load time, preventing silent misconfiguration. This design enables build-time parameter tuning from game telemetry without modifying strategy code.

Cross-Turn Memory

The AI persists deduced metrics to the **aa_metrics** database table via **(make-metric name value)** pseudo-commands emitted from Lisp. These metrics survive across decision cycles and are loaded back into the slate each GATHER.

Metric categories:

- Trend snapshots (credits, fleet power, economic trajectory)
- Enemy position history (per-ship hex snapshots)
- Enemy distance-to-base (closing rate inference)
- Combat watermarks (maximum observed PD, phasic, shield, launchers per enemy)
- Rack inference (enemy ship carried SystemShips)
- Per-theater engagement history (rounds fought, power committed)
- Enemy economic estimate (inferred total CR investment)

SYSTEMSHIP PICKUP AND DROP

During Movement Phase

- Costs 1 PD per SystemShip picked up or dropped
- Can be performed at any star hex during movement

During Combat

- Requires: Drive=0, Shield=0, Dodge or Escape tactic
- Only 1 SystemShip per combat round
- May fire Phasic (but not Torpedoes) while picking up/dropping
- Dropped SystemShips cannot fire that round
- Picked up SystemShips cannot fire but may power Shields

After Combat (Phase 5)

- Free rearrangement of SystemShips at star hexes
- No PD cost
- Active player only

BUILD POINT COSTS

Attribute	Cost
Power/Drive (PD)	1 CR per unit
Warp Generator (WG)	5 CR (WarpShips only, automatic)
Phasic (P)	1 CR per unit
Shield (S)	1 CR per unit
Launcher (L)	1 CR per launcher
Torpedoes (T)	1 CR per 3 torpedoes
SystemShip Hangar (H)	1 CR per hangar (WarpShips only)

CONSTRAINTS AND INVARIANTS

These are the fundamental laws of the game engine. They are never violated, and no command or sequence of commands can circumvent them:

- No randomness in combat—all outcomes are deterministic.
- No hidden information—all game state is public.
- Rules do not change mid-game.
- Equal inputs yield equal outcomes.
- Ships retain original tech level permanently.
- Warp Generators never take damage. A ship is destroyed when all attributes except WG reach 0.
- SystemShips cannot have Warp Generators or SystemShip Racks.
- WarpShips cannot be carried in SystemShip Racks.
- Phasics/Shields and Torpedoes cannot be used in the same combat round.
- SystemShips may never select the Escape tactic.
- PD used for movement is not permanently expended.
- Repairs cannot exceed original built capacity.
- Torpedo resupply cannot exceed original torpedo capacity.

EXAMPLES

Building a WarpShip

> **bn** W FALCON
> **bs** FALCON PD=5 P=3 S=2 L=1 T=3
> **bc** FALCON

Total cost: PD(5) + P(3) + S(2) + L(1) + T(1) + WG(5) = 17 CR. Tech level determined by current round.

Building a SystemShip

> **BN** S AVENGER
> **BS** AVENGER PD=1 L=1 T=6
> **BC** AVENGER

Total cost: PD(1) + L(1) + T(2) = 4 CR. No Warp Generator required.

Movement

> **M** W6 Tavon Xylen

W6 moves from current hex to Tavon via the nearest warpline path, then onward to Xylen. Total PD cost depends on path length (BFS-computed).

Combat Sequence

> **CO** W3 A S25 PD=2 P=3 S=2
> **CC**
(Wait for opponent to commit...)
(Resolution: W3 hits S25 for 3+tech damage)
> **CA** S25 PD=2 P=1

Torpedo Fire

> **CO** S25 D W3 PD=4 L=1 T=3
> **CC**

S25 dodges while firing 1 torpedo at W3. The torpedo uses Attack tactic with drive=3 on the Torpedo. Torpedo damage on hit = 2 + tech level.

Economic Workflow

> **out** W1 lrs
> **survey** ARVEN
> **extract scan**
> **ex** W1 ferrous
> **market**
> **trade sell** ferrous 5
> **fab** torpedoes W1

Save and Load

> **save** my-game
> **load** my-game
(Other player must type:)
> **accept** my-game

Quit

> **quit**
QUIT: Game auto-saved as 'autosaved-2026-02-13-14-30'.
Returning to lobby.
The game is auto-saved. Combat state is cleared. The other player is notified and redirected. The AI thread ceases operating on this game.

MODULES

Creating Custom Modules

A Module is a complete universe definition loaded from CSV files. All milieu data (star systems, planets, species, resources) is keyed by **module_id**.

Required files:

File	Content
star_systems.csv	Map topology (hex_id, name, base flags)
hexes.csv	Hex coordinates (q, r)
warplines.csv	Warpline connections (a_hex, b_hex)
warpline_hexes.csv	Intermediate warpline path hexes
planets.csv	Planetary bodies
species.csv	Xenoform species definitions
resources.csv	Extractable resource deposits
facilities.csv	Facilities and stations
anomalies.csv	System anomalies
salvageables.csv	Salvage sites and hazards
codex_rumors.csv	Knowledge and rumors per system

See the **ModuleAuthoring** wiki page for complete authoring instructions.

HEX EVENTS

Dynamic events affect specific hexes, modifying gameplay:

Event Type	Effect
NAVIGATION_HAZARD	+N PD cost to move through hex
COMBAT_INTERFERENCE	−N phasic damage in hex
SALVAGE_OPPORTUNITY	+25% salvage yields
EXTRACTION_BONUS	+N extraction yield

Events spawn and expire based on turn count. Use **hex** *location* to view active events.

MONITOR MODE

The server may accept commands from the terminal when started with **--monitor**. Each command must be prefixed with a player indicator. This feature is intended for development testing.

DATABASE

The game uses MySQL/MariaDB. The schema is defined in **site/db/Game.sql**. Module data is loaded from CSV files via **LoadModule.sql** scripts. Core data (help topics) is loaded via **site/db/core/Load.sql**.

Key tables: **games**, **ships**, **drafts**, **combat_state**, **combat_orders**, **pending_damage**, **star_systems**, **warplines**, **system_resources**, **codex_entries**, **market_prices**, **facility_control**, **hex_events**, **aa_metrics**, **saved_games**.

ADDING A GOAL TO THE AI AGENT

This section documents the procedure for adding a new goal to the Generation 4 AI Agent's goal graph. A goal is a declarative objective that the MPC evaluator can select, prioritize, and execute during the build phase of the AI's turn.

The example used throughout is a hypothetical goal called **:fortify-chokepoint**, whose purpose is to deploy a SystemShip defender at a warpline chokepoint hex when the AI controls that hex and it is ungarrisoned.

Step 1: Define the Goal (aa-goals.lisp)

Every goal is declared via the **defgoal** macro. The declaration establishes the goal's identity, its position in the dependency graph, and the predicates and action that govern it.

```
(defgoal :fortify-chokepoint
  :priority 7
  :supported-by :win-game
  :satisfied
    (null (find-unfortified-chokepoints slate))
  :precondition
    (and (>= (slate-credits slate) 200)
         (not (null (find-unfortified-chokepoints slate))))
  :action
    (issue-fortify-chokepoint slate goals)
  :needed-resources nil
  :doc "Build and deploy a defender at an ungarrisoned chokepoint.")
```

The fields:

:priority
: Integer. Lower numbers are higher priority. The evaluator walks goals in ascending priority order and selects the first that is unsatisfied and actionable. Choose a value that places the goal correctly relative to existing goals (see the goal table in the AI AGENT section).

:supported-by
: Keyword naming the parent goal. Every goal must trace to **:win-game** through its parent chain. The graph is validated at load time; a dangling parent is a fatal error.

:satisfied
: A form evaluated with the current slate bound. Returns true when the goal is already met and needs no action. The evaluator skips satisfied goals.

:precondition
: A form evaluated with the current slate bound. Returns true when the goal is actionable (i.e., the game state permits the goal's action to succeed). An unsatisfied goal with a false precondition is skipped; the evaluator moves to the next priority.

:action A form evaluated with both **slate** and **goals** bound. Must return a list of command plists (via **make-cmd**) or **nil**. Returning nil means the goal matched but could not produce a concrete command this cycle; the evaluator falls through.

:needed-resources
A form returning a list of resource-type strings (e.g., **'("FERROUS" "VOLATILE")**) that this goal's fulfillment consumes. The sell-excess logic checks this list before liquidating cargo. Use **nil** if the goal consumes no cargo.

Step 2: Implement the Finder (aa-util.lisp or aa-goals.lisp)
The satisfaction and precondition predicates typically call a finder function that locates the game-state entity the goal acts upon.

```
(defun find-unfortified-chokepoints (slate)
  "Return list of chokepoint hexes we control but have not garrisoned.
   A chokepoint is a hex with 3+ warpline connections bridging
   own-territory and enemy-territory."
  (let ((result nil))
    (dolist (hex (slate-chokepoint-hexes slate))
      (let ((garrison (count-if
                        (lambda (s)
                          (and (not (ship-warpship s))
                               (string= (ship-hex s) hex)))
                        (slate-own-ships slate))))
        (when (zerop garrison)
          (push hex result))))
    result))
```

Conventions for finder functions:

- Return the entity (ship, hex, system) or **nil**. For set-valued queries, return a list (empty list counts as nil for satisfaction tests).
- Accept only the slate as argument. Side-effect-free.
- Name the function **find-***what*-**for-***goal* or similar.

Step 3: Implement the Action Generator
The action function maps the goal to one or more concrete game commands. It returns a list of command plists or nil.

```
(defun issue-fortify-chokepoint (slate goals)
  "Build a defender SystemShip and deploy to the highest-value
   unfortified chokepoint.  Returns command list or nil."
  (declare (ignore goals))
  (let ((targets (find-unfortified-chokepoints slate)))
    (when targets
      (let ((best (first (sort targets #'>
                           :key (lambda (h)
                                  (compute-hex-value slate h))))))
        (format t "[LISP] fortify-chokepoint -> ~A~%" best)
        (list (make-cmd "BN" "S SENTINEL")
              (make-cmd "BS" "SENTINEL PD=6 P=4 S=3")
              (make-cmd "BC" "SENTINEL")
              (make-cmd "DS"
                (format nil "SENTINEL ~A" best)))))))
```

Conventions for action generators:

- Return **(list (make-cmd ...))** for success, **nil** for "cannot act this cycle."
- Use **(make-cmd "CMD" "ARGS")** to construct command plists. The C++ RENDER phase injects these as textual commands through `yyparse()`.
- Log the action via **format** to **t** for diagnostics.

- Multi-command sequences (build + set + commit + deploy) are legal; the RENDER phase executes them in order.

Step 4: Add Theta Parameters (aa-theta.lisp)

If the goal uses any tunable constants, declare them in the theta parameter vector rather than hardcoding them in the action function.

```
(defparameter *theta-fortify-min-warplines* 3
  "Minimum warpline connections for a hex to qualify as chokepoint.")

(defparameter *theta-fortify-max-defenders* 1
  "Maximum defender SystemShips to deploy per chokepoint.")
```

Reference them via **(theta 'theta-fortify-min-warplines)** in finder and action functions.

Step 5: Integrate with the Rule Engine (Optional)

Goals that operate during the build phase are automatically invoked by the existing economic-dispatch rule in **aa-rules.lisp**. No additional rule is needed unless the goal requires action during a different phase (movement, combat, pick/drop).

For non-build-phase goals, register a strategy rule:

```
(define-strategy-rule fortify-move-defender
  :phase :move
  :priority 250
  :when (and (find-unfortified-chokepoints slate)
             (find-idle-defender slate))
  :action (issue-move-defender-to-chokepoint slate strategy)
  :doc "Move idle defender toward unfortified chokepoint.")
```

Step 6: Validate the Goal Graph

The graph validator runs at load time. It verifies:

- Every goal traces to **:win-game** via **:supported-by** links.
- No cycles exist in the dependency chain.
- No duplicate **:objective** keywords.

A goal with a dangling parent (**:supported-by** naming a nonexistent goal) is a fatal load error. Test by restarting the server with the AI DSL path and checking the log for validation output.

Step 7: Metric Persistence (Optional)

If the goal benefits from cross-turn memory, emit metrics via **make-metric** in **compute-metrics-to-persist** (**aa-core.lisp**).

```
(make-metric "fortify-last-target" best-hex)
(make-metric "fortify-deployed-count" n)
```

These are intercepted by the C++ layer as pseudo-commands, written to the **aa_metrics** table, and reappear in the slate's **:metrics** alist on the next GATHER.

Summary: Files Modified

File	Change
aa-goals.lisp	defgoal declaration
aa-util.lisp or aa-goals.lisp	Finder function
aa-economic.lisp or new module	Action generator
aa-theta.lisp	Tunable parameters (if any)
aa-rules.lisp	Strategy rule (if non-build phase)
aa-core.lisp	Metric persistence (if cross-turn memory needed)

No C++ changes are required. The C++ GATHER/RENDER loop and the **defgoal** macro handle registration and dispatch automatically.

AI AGENT DESIGN NOTES

This section documents design considerations and future directions for the Autonomy Agency.

Closed-Loop Command Execution

The Mealy state machine is a closed loop: GATHER reads the world state after every RENDER, so a failed command changes the slate (or leaves it unchanged, which itself is information) and CALCULATE derives a different action on the next cycle.

For the loop to close tightly, the agent must distinguish between "the command succeeded and the world changed" and "the command failed and the world did not change." The TaskRunner carries both the result code and the error message from `invoke()`. When this information is conveyed back through the slate, the Lisp code can reason about failures as structured ontology facts rather than opaque strings:

- `:precondition-violated :credits-below-threshold`
- `:precondition-violated :hex-in-combat`
- `:precondition-violated :phase-mismatch`

Failure reasons scoped to the ontology dictionary guarantee that the Lisp code can handle every failure cause, because the set is finite and known at compile time. Failures become structured inputs to CALCULATE, not a special side channel.

Failure lifetime is phase-scoped: the failure list clears at each phase transition. Within a single phase, preconditions are stable (the world changes only through the agent's own actions). Combat is the exception, where state changes within a phase as orders resolve; combat failures may need round-scoped or stage-scoped clearing.

Latent Learning: The Diary Concept

The learning feedback loop separates into two timescales:

Runtime
: Static Lisp, fixed ontology, metrics memory. No self-modification. Deterministic and debuggable.

Build-time
: Ingest accumulated game diaries, produce updated Lisp parameters. The agent evolves through a controlled, auditable build step.

The diary extends the **aa_metrics** table to capture game-over summaries: ending round, winner, fleet composition at key moments, strategies attempted and when. Three translation approaches, ordered by complexity:

1. **Parameter Tables.** The diary feeds a statistical pass that emits a `defparameter` config file. Example: "average earliest viable torpedo build = round 5.2 across 47 games" becomes a theta value. No generated Lisp code. Simplest and likely captures 80% of the benefit.
2. **Rule Selection.** Multiple strate gy variants are written by hand (aggressive, turtle, torpedo-rush). Diary analysis determines which variants performed best per opponent profile. The build step emits a selection table.
3. **Lisp Code Generation.** An e xternal tool reads the diary, identifies patterns, and emits new Lisp functions. Requires validation against the ontology. This is a research direction more than an engineering task.

For approaches 1 and 2: a pre-build script queries the metrics database, runs the analysis, and writes a **.lisp** file of `defparameter` declarations. The normal build includes that file.

The power of the current architecture is that all Lisp functions are pure: slate in, decisions out. They carry no hidden state and do not mutate themselves. This makes them debuggable, predictable, and auditable. The diary/build-time approach preserves these properties by restricting evolution to a versioned build step.

BUGS

Report bugs to the Kepler's Horizon issue tracker.

FILES

Files that are deployed when installing the system:

site/web/
: Static HTML, CSS, JavaScript files served by the HTTP layer. These are deposited in the directory where your web server is accessing content.

site/db/Game.sql
: Complete database schema. Stored securely, used to initiate the database back-end.

site/db/core/help/
: Help topic CSV files for the in-game help system. Stored securely, used by the database initaliation setup.

site/db/modules/kh/
: Module data CSV files for the default Kepler's Horizon universe. Stored securely, used by the database initaliation setup.

dsl/ AI strategy DSL files (Common Lisp). A directory containing the run-time *LISP* code.

AUTHORS

Game design and implementation by sibomots.

ACKNOWLEDGEMENT

Game design is based upon the table-top (pencil and paper) game called **WarpWar** that was released in 1977. https://en.wikipedia.org/wiki/WarpWar Kepler's Horizon has extended the game much further, adding milieu concepts for market (trading, selling resources), resource extraction, exploration, fabrication, random space hazards, and an AI-Agent for single player mode.

FORMAT

This document is set in the style of traditional *UNIX* manual pages. It was prepared using *troff (equivalent)* with the **man**(7) macro package.

```
groff -t -man -Tpdf kh.6 > kh.pdf
```

Sectioning, typography, and layout follow established UNIX documentation conventions. No additional formatting mechanisms are employed.

COPYRIGHT