浮体式洋上風力施工プロセスモデル
詳細設計ドキュメント

港湾空港技術研究所 海洋利用研究グループ

2025 年 9 月 29 日

目次

1　はじめに

近年，地球温暖化による気候変動は世界的な課題となっており，我が国においては，2050 年までのカーボンニュートラル達成を宣言している．二酸化炭素排出量を実質ゼロへ抑えるには，発電・輸送・産業といった社会基盤全体のエネルギー転換が不可欠であり，再生可能エネルギーの大幅な導入が求められる．太陽光や陸上風力に加え，広大な海域を活用できる洋上風力発電は，安定した風況と高い設備利用率により次世代の主力電源として期待されている．特に水深が深い海域では従来展開されてきた着床式の風車を設置できないため，海上の浮体に風車を搭載する浮体式洋上風力発電（Floating Offshore Wind Turbine，以下 FOWT）が注目を集めている．

FOWT の導入拡大は温室効果ガス削減のみならず，エネルギー安全保障の観点からも重要である．日本は国土が狭く可採エネルギー資源に乏しいため，化石燃料の多くを輸入に頼っている．しかし，世界情勢の不確実性が高まる中で安定供給を確保するには，国内で生産可能な再生可能エネルギーの比率を高めることが不可欠である．また，海に囲まれた日本は沿岸域に広大な潜在的資源を持ち，FOWT の大規模展開は地域経済の活性化や関連産業の創出にもつながる．さらに，近年のカーボンプライシングや ESG 投資の潮流は，低炭素社会に適応した新しいビジネスモデルを後押ししており，FOWT への関心は産官学で急速に高まっている．

しかし，FOWT の建設には従来の陸上インフラと異なる技術的・運用的課題が存在する．巨大な浮体や風車部材を製造・輸送・組立・設置するプロセスは多岐にわたり，海象や気象条件に左右されやすい．港湾設備の制約や作業ウィンドウの短さが工程計画を複雑化し，コストとリスクの管理が難しい．さらには，複数の事業者や作業船，陸上の大型クレーン，高い地耐力を持った岸壁利用が同時に関与するため，全体を俯瞰しながら資源を最適に配分する調整メカニズムが不可欠である．これらの課題を解決するためには，実際の施工に先立って様々なシナリオを仮想的に検証し，ボトルネックや潜在的なリスクを把握することが求められる．

本シミュレーションは，こうした背景を踏まえ，FOWT 建設の各工程を仮想空間で再現する C++ 製のマルチエージェントシステムとして設計された．複数の仮想エージェント（例：基礎を組み立てるドックや資材を運ぶタグボート）が協調し，基礎の製作・輸送・設置といった作業を自律的に進める．各エージェントは港の配置や気象条件などの環境情報を参照しながら行動し，メッセージを介して作業状況を共有するため，工程間の待ち時間や混雑を定量的に評価できる．ユーザは入力パラメータを変更することで，例えばタグボートの台数や港のレイアウトがプロジェクト全体のスケジュールに与える影響を比較検討できる．

本プロジェクトの最終的なゴールは，施工プロセスをデジタル上で高精度に再現し，計画立案や最適化に寄与する「施工プロセスの計画最適化基盤」を提供することである．シミュレーションによって得られた知見は，建設現場の安全性向上やコスト削減，資材・人員配置の効率化，さらには将来の大規模施工の加速に向けた技術検証に役立つと期待される．また，本システムは C++ 初心者でも扱えるようコードを簡潔に保ち，ドキュメントを整備している．これにより，教育目的や研究開発においても柔軟に活用できる．

以上のように，カーボンニュートラル社会の実現に向けた FOWT の普及と，複雑な施工プロセスを可視化・評価するツールの必要性が，本シミュレーション開発の背景にある．仮想空間で多様なシナリオを試行し，得られたデータを基に現実のプロジェクトを改善することは，持続可能なエネルギーインフラの構築に大きく貢献するだろう．さらに，世界各国では 2050 年カーボンニュートラルに向けたロードマップが相次いで策定されており，欧州連合では洋上風力の導入目標を大幅に引き上げる政策が進行している．我が国でも「第 6 次エネルギー基本計画」において洋上風力が重要な位置づけを占め，2030 年代以降に大規模な商用化が始まると見込まれている．これまでに国内外で様々な実証プロジェクトが実施され，浮体式においてもセミサブ型，スパー型など多様な構造が提案されている．これらは環境条件や製造コスト，設置方法によって得手不得手が異なり，最適な方式を選定するためには周辺環境や供給体制を考慮した総合的な評価が必要である．

FOWT の建設は多くの工程が相互に依存している．例えば，基礎の製造スケジュールが遅延すれば，その後の曳航や

設置作業も連鎖的に遅れる．また，港湾のバース数が限られる場合には資材や完成基礎を仮置く場所が不足し，船舶の待機や渋滞が発生する．こうした状況では単一の工程だけを最適化しても全体の効率は向上せず，むしろボトルネックが別の場所へ移動するだけである．複数の工程を同時に評価し，システム全体の調和を図るためのシミュレーション技術が求められている．

デジタルトランスフォーメーションの観点からも，施工プロセスの仮想化は大きな意義を持つ．近年は建設業界でもデジタルツインや BIM（Building Information Modeling）を活用した高度な計画手法が普及しつつあり，仮想空間上で設備や工程を再現することで，実際の現場に先立って潜在的な問題を洗い出せる．本システムはその一環として，エージェントベースのシミュレーションを用いて施工中の資源の流れを可視化し，計画の柔軟な修正やリスク予測を可能にする．

エージェントベースアプローチの利点は，各要素を個別の主体として扱うことで複雑な相互作用を簡潔に表現できる点にある．ドック，タグボート，クレーン船，プレアンカーフリートや係留フリートといったエージェントはそれぞれ独自の状態と行動ルールを持ち，メッセージ通信によって協調する．本プロジェクトでは時間の進行を離散ステップで管理し，各ステップでエージェントが状態を更新する．これにより，工程の競合や待ち行列，資材の不足といった現象を自然に再現できる．

さらに，システムは拡張性を重視して設計されており，新たなエージェントや海域条件を追加することで，多様なシナリオを評価できる．例えば，洋上組立基地での施工や複数港湾による同時施工といったケースを模擬することも可能である．将来的には，気象データのリアルタイム連携や経済性評価モジュールとの結合など，より高度な分析に向けた拡張が期待される．こうした発展性は，研究コミュニティだけでなく産業界が直面する具体的な課題にも対応できる柔軟なツールとして，本システムを位置付ける．

最後に，本ドキュメントではシステムの設計思想と実装方法を順を追って説明する．初めて C++ を学ぶ読者でも理解できるよう専門用語には注釈を添え，コードの引用には行番号付きのリスティングを用いる．読み進めることで，FOWT 施工プロセスの概要からシミュレーションモデルの構築手法，結果の解釈に至るまで一連の流れを把握できるだろう．さらに，FOWT 建設プロジェクトでは多岐にわたるステークホルダーが関与する．国交省をはじめとする政府機関は海域利用の許認可や環境影響評価を担当し，自治体は港湾の整備や地域住民との合意形成を進める．電力事業者やゼネコンは設計・施工を担い，船舶運航会社や資材サプライヤーが物流を支える．これらの主体がタイミング良く連携しなければ，プロジェクトは遅延やコスト増に直結する．本シミュレーションでは，各主体の行動をエージェントとしてモデル化することで，情報共有の遅れや資源の競合が全体工程に与える影響を可視化できる．

また，気象条件の変動は海上工事の最大のリスク要因の一つである．強風や高波，台風の接近などによって作業が停止すると，船舶の待機費用や設備の維持費が増大する．本モデルでは今後，気象データを入力として与えることで，作業可能日数の推計や悪天候時の代替案検討など，より現実的なリスク分析へ発展させることができる．

こうした観点から，本システムは単なる学術的なモデルにとどまらず，政策立案や産業界の実務に資する実験基盤として位置付けられる．シミュレーションを通じて得られた成果が，2050 年カーボンニュートラルという国家目標の達成に向けた実行可能な戦略の策定を後押しすることを期待する．

加えて，FOWT 設備のライフサイクル全体を見れば，運用保守や撤去の段階でも環境負荷の低減とコスト効率化が重要なテーマとなる．長期にわたり海上に設置される基礎は，腐食や疲労による劣化が避けられず，定期点検や部材交換が欠かせない．シミュレーションで保守スケジュールを検討することで，ダウンタイムの最小化や予防保全の効果を評価できる．運転終了後の撤去計画についても，資材のリサイクルや海洋環境への影響評価を含めた総合的な検討が求められる．

シミュレーション結果は政策決定や企業の投資判断に重要なエビデンスを提供する．計画段階で複数のシナリオを比較し，最も費用対効果の高い戦略を選択できることは，公共事業の透明性向上にも寄与する．モデルの構造やパラメータを明示的に示すことで，結果の再現性と説明責任を担保している点も本プロジェクトの大きな特色である．

本シミュレーションの活用例として，港湾レイアウトの改善案の比較が挙げられる．例えば，バースの増設や倉庫配置

の変更が待機時間をどの程度短縮するか，浮体仮置きの場所などを事前に検証できる．計算結果は CSV 形式で出力され，Python スクリプトによる可視化や統計解析に利用できるため，意思決定者は数値に基づいた議論を行える．こうした定量的アプローチは，経験や勘に頼りがちな従来の計画手法からの脱却を促し，データ駆動型のプロジェクト管理へとつながる．このように，本システムはエネルギー政策・産業展開・教育活用の各側面から価値を提供するプラットフォームを目指している．

今後も実運用から得られる知見を反映し，精度と実用性を高めていく予定である．

1.1　モデルの概要

本モデルは，浮体基礎のライフサイクルに沿った施工プロセスを，イベント駆動型のエージェント相互作用として表現する．対象とする典型的なフローは以下の通りである：

1. ドックにおける浮体基礎の製作
2. ヤードでの仮置き・保管
3. 岸壁（クレーン）での組立作業
4. プレアンカーフリートによる事前アンカー敷設と係留試験
5. 曳航船＋台船による現場への輸送と係留フリートによる最終係留

図 1 に，各工程を連結するエージェント間のメッセージ遷移をシーケンス図として示す．

各工程は，港湾内の移動・バース占有・在庫物流と密接に結びついており，これらを単一の時空間モデル上で連鎖的に再現することで，全体プロセスのつながりを崩さない設計となっている．

1.2　アーキテクチャ

シミュレーションは C++ コアと Python 可視化群から構成される．C++ 側では，以下のようなエージェントが定義される：

- **FloatingFoundation**: 浮体基礎
- **Dock**: 製作ヤード
- **YardStorage**: 保管ヤード
- **QuayCrane**: 岸壁クレーン
- **TowFleet**: 曳航船隊
- **PreAnchorFleet**: 事前アンカー敷設・係留試験艦隊
- **MooringFleet**: 係留艦隊
- **MaterialStockAgent**: 資材・部材在庫管理

各エージェントは非同期メッセージングを担うイベントバスを介して疎結合に連携する．シミュレーションは `TIME_STEP_MIN`=6 分（`TIME_STEP_HOURS`=0.1 時間）刻みの離散時間で進行し，刻み幅は `cpp/include/constants.h` 内の定数を変更して調整できる．必要に応じてイベント発火により即時反応が起きる．状態遷移とキュー処理により，処理能力・待ち行列・優先度が自然に表現される．

1.2.1　エージェント詳細

以下に，主要エージェントの内部状態と責務，代表的な API を示す．

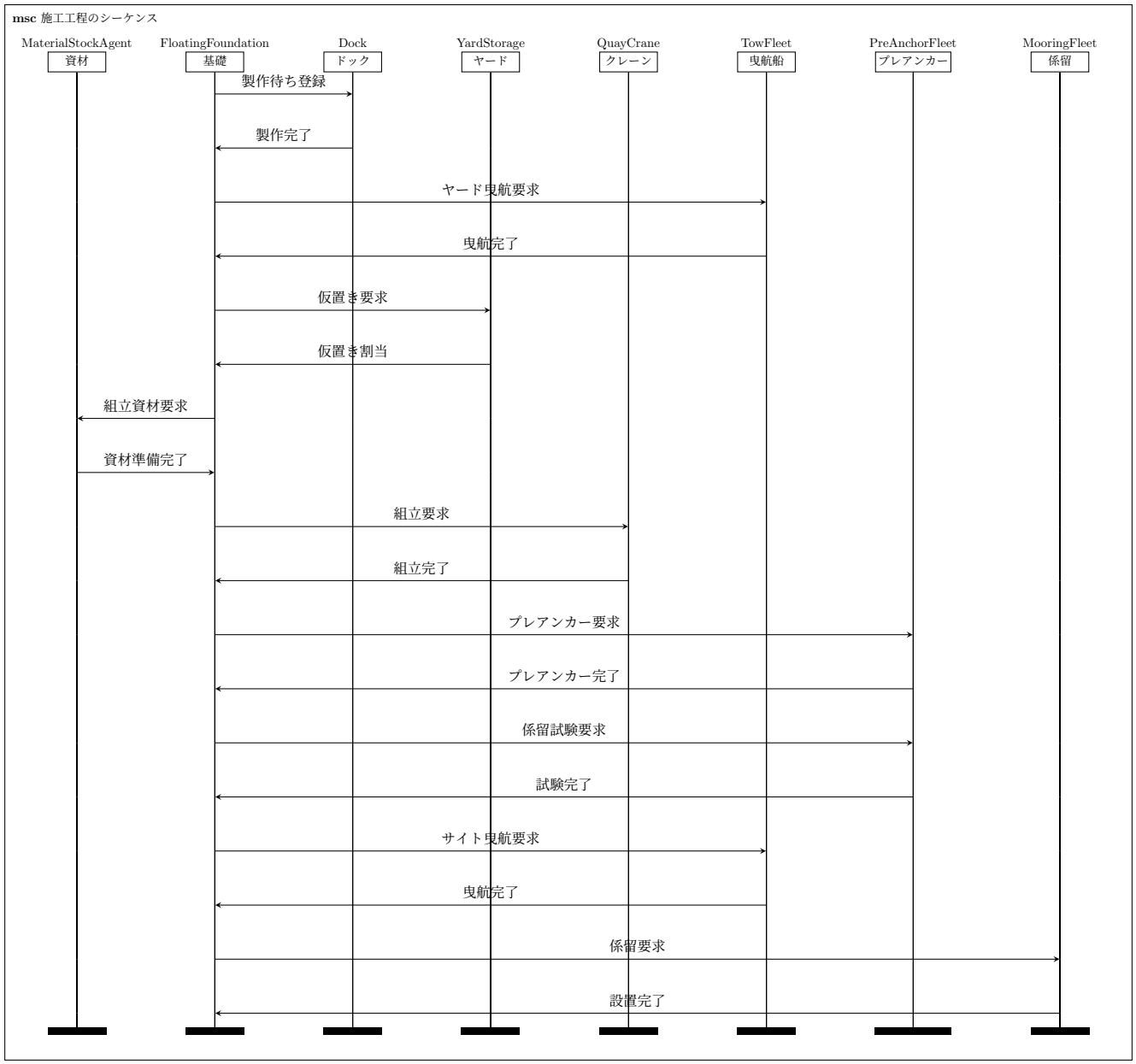

図 1　代表的な施工工程におけるエージェント間のシーケンス図

■ FloatingFoundation

基礎はまず None 状態で待機し，StorageSlotAssigned を受信すると Fabrication へ移る． BuildComplete 後に DockTow へ進み，TowComplete により YardStorage へ到着する． StorageSlotAssigned で AssemblyPending に遷移し，MaterialReady 受信をもって Assembly を開始する． AssemblyPending ではヤードスロットを保持したまま部材準備を待機し，MaterialReady を受け取るまで岸壁へ移動しない． YardStorage 段階で `StorageSlotAssigned` を受け取った時点で組立用資材を `MaterialStockAgent` へ要求し，ヤードを離れる前に部材手配を完了させる制御に統一された。[*1] 再度 TowComplete で Anchoring に入り，AnchorComplete を受け取ると Completed で終了す

[*1] `cpp/src/floating_foundation.cpp`

FloatingFoundation : Agent
status : std::string route_ : std::deque<Vec2> currentZone : std::string
step(dt) receive(msg) requestTow()

図 2　FloatingFoundation の簡易クラス図

状態	受信メッセージ	遷移先
None	StorageSlotAssigned	Fabrication
Fabrication	BuildComplete	DockTow
DockTow	TowComplete	YardStorage
YardStorage	StorageSlotAssigned	AssemblyPending
AssemblyPending	MaterialReady	Assembly
Assembly	AssemblyComplete	SiteTow
SiteTow	TowComplete	Anchoring
Anchoring	AnchorComplete	Completed

表 1　FoundationStage の状態遷移例

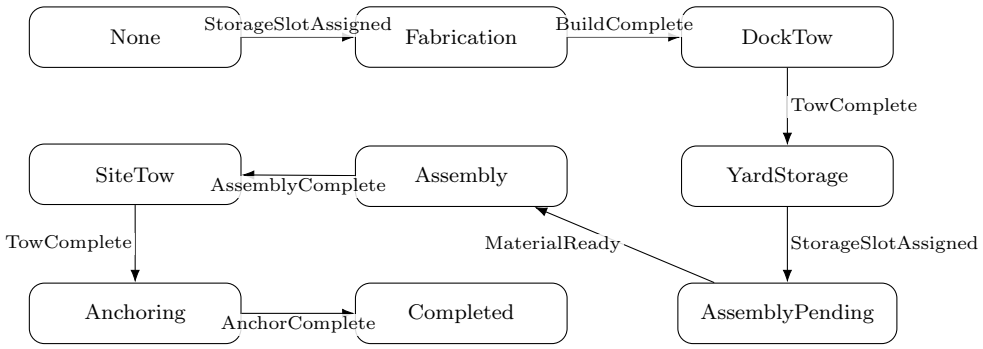

図 3　FoundationStage の状態遷移図

る． 列挙子は `FloatingFoundation::FoundationStage`[*2] に定義され，遷移イベントは `MessageType`[*3] を処理する `FloatingFoundation::receive`[*4] の実装に基づく．現在 `Anchored` 列挙子は未使用で，AnchorComplete で直接 `Completed` へ遷移する．

API 一覧:

- `void step(double dt)`

[*2] `cpp/include/floating_foundation.h`
[*3] `cpp/include/message.h`
[*4] `cpp/src/floating_foundation.cpp`

- `void receive(const Message&)`
- `void requestTow()`

■ Dock

Dock：Agent
capacity_：int processDays_：int waiting_：std::queue<AgentID> working_：std::vector<Task>
step(dt) receive(msg)

図 4　Dock の簡易クラス図

状態	イベント	遷移先
idle	JoinQueue	queued
queued	SlotOpen	fabricating
fabricating	BuildComplete	idle

表 2　Dock の状態遷移例

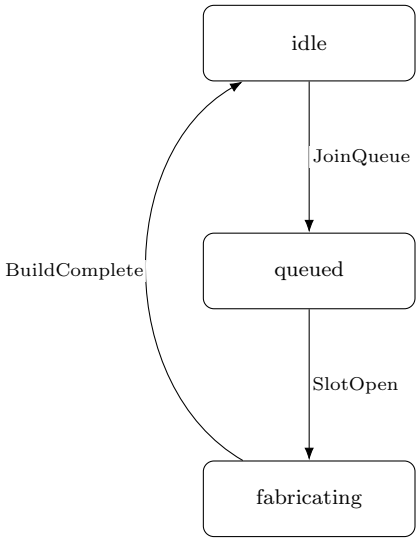

図 5　Dock の状態遷移図

Dock では基礎が JoinQueue で待機列に入り，SlotOpen が発生すると fabricating に進む．製作完了 (BuildComplete) で idle に戻り，次の基礎を受け入れる準備が整う．API 一覧:

- `void step(double dt)`
- `void receive(const Message&)`

■ YardStorage

YardStorage : Agent
capacity_ : int occupied_ : int queue_ : std::queue<AgentID>
step(dt) receive(msg) allocateSlot(id)

図 6　YardStorage の簡易クラス図

状態	イベント	遷移先
idle	Arrive	loading
loading	StoreComplete	idle
idle	RetrieveRequest	unloading
unloading	RetrieveComplete	idle

表 3　YardStorage の状態遷移例

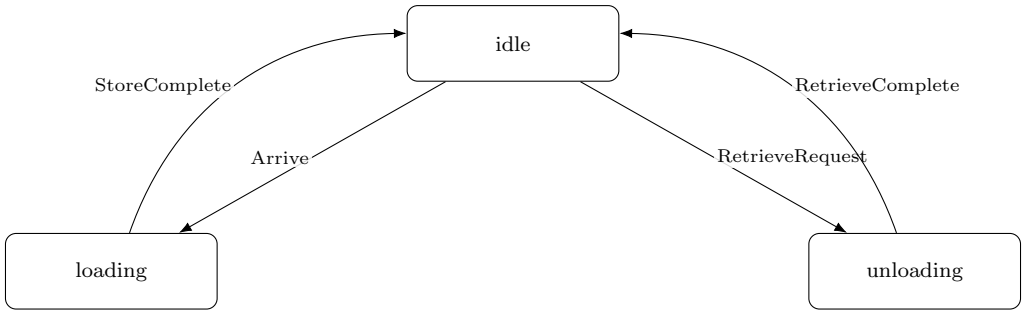

図 7　YardStorage の状態遷移図

基礎が到着 (Arrive) すると loading で格納作業を行い，StoreComplete で idle に戻る． 出庫要求 (RetrieveRequest) が来た場合は unloading で搬出し, RetrieveComplete 後に再び idle となる． 空きが生じた際は `step()` 内で `freeSlots = capacity_ - occupied_` を算出し，`i < freeSlots && !waiting_.empty()` の条件で待機列から基礎を取り出すため，割当数が空き容量および待機数を超えることはない．API 一覧:

- `void step(double dt)`
- `void receive(const Message&)`
- `void allocateSlot(AgentID)`

■ QuayCrane

API 一覧:

- `void step(double dt)`

QuayCrane : Agent
processDays_ : int currentId_ : AgentID
step(dt) receive(msg) requestMaterial()

図 8　QuayCrane の簡易クラス図

状態	イベント	遷移先
idle	StartAssembly	assembling
assembling	MaterialShortage	waitingMaterial
waitingMaterial	SupplyArrived	assembling
assembling	AssemblyComplete（資材不足なし）	idle

表 4　QuayCrane の状態遷移例（StartAssembly, MaterialShortage, SupplyArrived, AssemblyComplete）

- `void receive(const Message&)`
- `void requestMaterial()`

■ TowFleet

TowFleet : Agent
capacity_ : int queue_ : std::deque<TowReq> active_ : std::vector<ActiveTow>
registerFoundation(f) step(dt) receive(msg)

図 9　TowFleet の簡易クラス図

状態	イベント	遷移先
idle	TowRequest	towing
towing	ArriveSite	returning
returning	ReturnComplete	idle

表 5　TowFleet の状態遷移例

曳航艦隊は TowRequest を受けると towing となり，現場到着 (ArriveSite) 後は returning で帰港する． 復路が完了 (ReturnComplete) すると idle に戻り，次の依頼を待つ．API 一覧:

- `void registerFoundation(const std::shared_ptr<FloatingFoundation>&)`

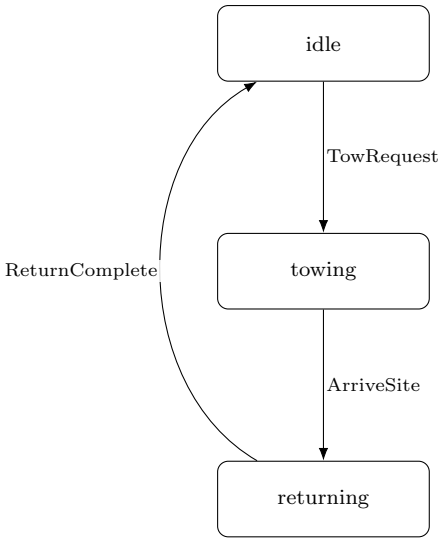

図 10　TowFleet の状態遷移図

- `void step(double dt)`
- `void receive(const Message&)`

■ PreAnchorFleet

PreAnchorFleet : Agent
capacity_ : int anchorsPerFoundation_ : int
step(dt) receive(msg)

図 11　PreAnchorFleet の簡易クラス図

状態	イベント	遷移先
idle	PreAnchorRequest	preAnchoring
preAnchoring	PreAnchorComplete	idle
idle	AnchorTestRequest	testing
testing	AnchorTestComplete	idle

表 6　PreAnchorFleet の状態遷移例

プレアンカー艦隊は PreAnchorRequest を受信すると preAnchoring でアンカー設置を行い，完了後に PreAnchorComplete を返す．続いて AnchorTestRequest で testing に移行し，AnchorTestComplete で idle に戻る．API 一覧:

- `void step(double dt)`
- `void receive(const Message&)`

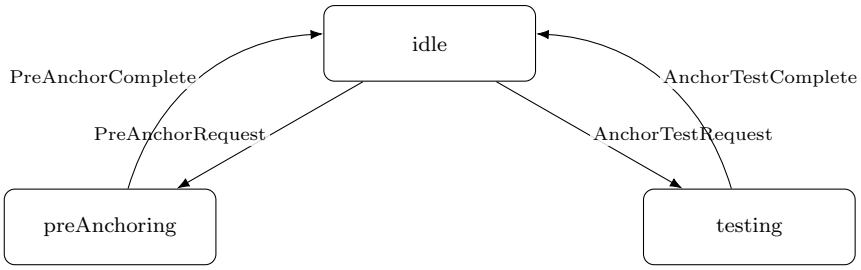

図 12　PreAnchorFleet の状態遷移図

■ MooringFleet

MooringFleet : Agent
capacity_ : int activeId_ : AgentID
step(dt) receive(msg)

図 13　MooringFleet の簡易クラス図

状態	イベント	遷移先
idle	MooringRequest	installing
installing	InstallComplete	idle

表 7　MooringFleet の状態遷移例

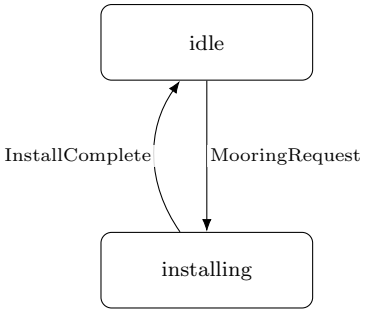

図 14　MooringFleet の状態遷移図

現場に曳航された基礎を受け取ると installing で係留作業を行い，作業完了 (InstallComplete) で idle に戻る．API 一覧:

- `void step(double dt)`
- `void receive(const Message&)`

■ MaterialStockAgent

MaterialStockAgent : Agent
inventory_ : `std::unordered_map<std::string,int>` reorderPoint_ : `int`

`step(dt)`
`receive(msg)`
`fulfillRequest(type)`

図 15　MaterialStockAgent の簡易クラス図

状態	イベント	遷移先
normal	StockLow	ordering
ordering	DeliveryArrived	normal
normal	IssueRequest	issuing
issuing	IssueComplete	normal

表 8　MaterialStockAgent の状態遷移例

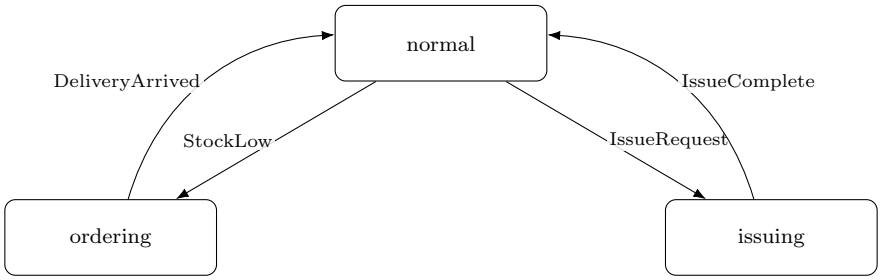

図 16　MaterialStockAgent の状態遷移図

API 一覧:

- `void step(double dt)`
- `void receive(const Message&)`
- `void fulfillRequest(const std::string&)`

環境（Environment）は港湾マップ JSON から構築され，ゾーン（Dock, Yard, Quay, Sea など），障害物，バース群，速度プロファイルを保持する．エージェントの移動は簡易 A探索により実現され，ゾーン種別に応じて移動速度が切り替わる．港内低速・外海高速といった現実的挙動が反映され，係留・離岸の所要時間もパラメータとして設定可能である．

資材在庫管理ポリシー（発注点方式）

本研究のシミュレーションでは，資材在庫の補充に確率的在庫管理モデルを採用している．資材 $i \in \{\text{Blade, Nacelle, Tower}\}$ について，以下のような管理基準を定める．

まず，「有効在庫量」（effective inventory level）を以下のように定義する：

$$E_i = S_i + O_i - B_i \tag{1}$$

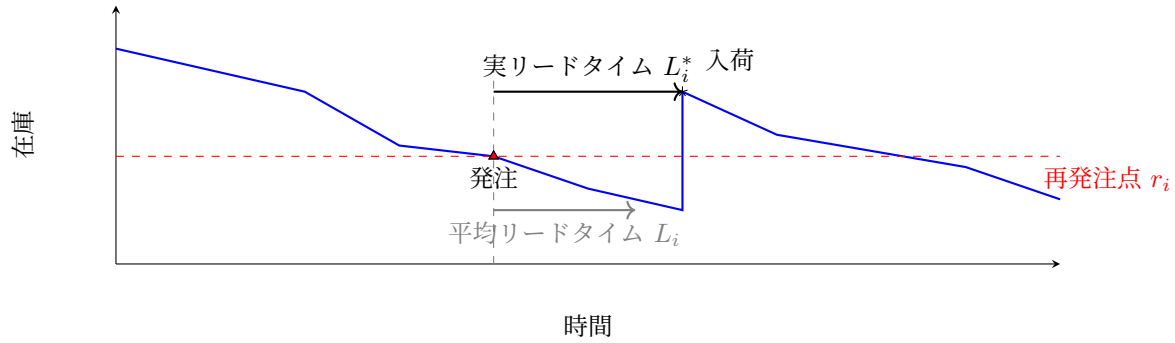

図 17　在庫水準とリードタイムの模式図．発注後も在庫は減少を続け，平均リードタイム L_i とは異なる実リードタイム L_i^* 経過後に入荷が反映される．

ここで，

- S_i：現在手元にある手持ち在庫量（現物在庫），
- O_i：既に発注済みであり，まだ納入されていない在庫（発注先行在庫／パイプライン在庫），
- B_i：引き当て待ちの要求（バックログ）によって将来必要と見込まれる不足量．

発注点（reorder point），これを発注を行う在庫残量のしきい値として，次式で与える：

$$r_i = \mu_i L_i + Z\,\sigma_i \sqrt{L_i} \tag{2}$$

ここで，

- μ_i：単位時間あたりの需要の平均，
- σ_i：同じく需要の標準偏差，
- L_i：平均リードタイム，
- Z：所望のサービスレベル（欠品許容率）に対応する安全係数．

発注の判断は，式 (1) で定義される有効在庫量 E_i が発注点 r_i 以下になったときに発注をかける方式である．すなわち，発注判定条件は

$$E_i \le r_i$$

となる．

シミュレーションでは，各資材について需要の平均 μ_i 及び標準偏差 σ_i を，在庫消費の時系列データ（過去の出庫・消費履歴）を用いて移動平均法で逐次推定する．一方，平均リードタイム L_i は定数として設定しておき，発注点 r_i の計算にはこの平均値を用いる．ただし，実際に発注を行う際のリードタイムは，揺らぎを含ませ，

$$L_i^* \sim \mathcal{N}(L_i,\ \sigma_{L_i}) \tag{3}$$

という正規分布からサンプリングされる値を用いる．例えば Blade 部材については平均 10 日，標準偏差 5 日を設定し，発注ごとに 8 日や 12 日といった揺らぎが実際のリードタイムとして発生するようにしている．これにより，発注点の計算に使われる平均リードタイム L_i と，実際の発注処理で用いられる実リードタイム L_i^* の間に自然なずれ・変動が生じる構造を持たせている（図 17）．

1.3　入力と出力

入力は以下の要素から構成される：

1. 港湾地物 JSON：ゾーン矩形，障害物，ターゲット座標，バース表，速度プロファイル
2. 施工パラメータ：浮体基礎製作ライン数，製作日数，曳航船数，アンカー能力，ヤード容量，クレーン本数，組立日数
3. 部材需要・BOM・初期在庫・発注ポリシー（リードタイム・バッチサイズ）
4. 乱数種とシナリオフラグ

出力はすべてのエージェントについて，位置，状態，ゾーン遷移，宛先などを含む CSV ログとイベント時系列ログである．Python スクリプト群により，以下の可視化と分析が生成可能である：

- 工程ガントチャート
- 装置稼働率・待機率
- 在庫水準と欠品時刻
- 港内軌跡アニメーション
- 基礎ごとのリードタイム分解（待ち／加工／移動）

主要な KPI として，総工期及び各基礎当たりサイクルタイム，資機材稼働率（稼働・アイドル・待ち行列状況），曳航総距離・時間，欠品発生回数を標準的に算出する．

1.4　モデル化上の着目点

本モデルの要点は，プロセス間のつながりを忠実に維持することである．ドックでの製作完了は曳航要求を発火し，TowFleet は能力制約下で優先度付きキューからジョブを取り出す．到着先のヤードに空きがなければ係留待ちが発生し，港内の占有・バースの空き具合により到着時刻が変動する．岸壁での組立は部材在庫に依存し，欠品時には MaterialStockAgent が発注・入庫イベントを管理する．入庫完了後に組立が再開される．最終的に，サイト曳航・係留が完了して初めて 1 基の「完成」がカウントされる．このように物流・移動・設備能力・在庫を一体で連鎖させることで，単独工程の最適化では見えない全体効果を炙り出すことができる．

1.5　研究利用のワークフロー

研究利用を想定した評価手順は以下の通りである：

1. 現況パラメータで基線ケースを実行し，KPI とボトルネックを可視化
2. 制約緩和（装置増設，バース再配置，速度プロファイル見直し）や運用ルール変更（曳航優先度，発注ポリシー）をシナリオとして投入
3. 感度分析により効果の立ち上がり，閾値等を把握
4. 実施工実績の指標（待機率，平均係留時間，日別完成数）と照合し，パラメータ同定・バリデーションを実施

現段階では気象・海象による作業中断は未導入であるが，天候カレンダーや確率的停止を組み込むシステム余地を備えており，将来的には波浪制約・作業ウィンドウ判定を導入可能である．

1.6　特徴と限界

本モデルの特徴は以下の通りである：

- 港湾地理と移動を明示したイベント駆動型結合モデル
- リソース制約・在庫・キューイングを同時に扱える拡張性
- 実務者に理解しやすい I/O（JSON + CSV）と再現性の高い可視化パイプライン

限界としては，細かな操船挙動等は簡略化しており，潮汐・交通管制・詳細な気象は現時点では考慮しない点が挙げられる．この点に関しては将来的な拡張課題として位置付けている．

2　ディレクトリ構成

リポジトリ直下にある主なフォルダと役割を以下に示す.

`cpp/`　C++ のソースコード一式．`include/` にヘッダ，`src/` に実装があり，`main.cpp` がエントリポイントである.
`data/`　港湾レイアウトなどシミュレーションに用いる入力データ（JSON 形式）.
`doc/`　技術ドキュメント．本ファイルもここに含まれる.
`python/`　ログ可視化や解析のための Python スクリプト.

2.1　開発環境入門

■ ディレクトリの見取り図

```
cpp/ ├──
  include/ ヘッダファイル ├──
  src/ ソースファイル └──
  main.cpp エントリポイント
data/
doc/
```

1 クラスにつき 1 ヘッダ + 1 ソースを用意するのが原則です.

■ Makefile の最小例

```
all: main.exe

main.exe: $(OBJS)
    $(CXX) $(CXXFLAGS) -o $@ $^

clean:
    rm -f $(OBJS) main.exe
```

上から順に「最終成果物」「その材料」「掃除の仕方」を示しています．`make` は **ルートディレクトリ**で実行し，生成された `cpp/main.exe` は `cd cpp` の後に実行します.

■ ビルドから実行まで

1. ルートディレクトリで `make` を実行する.
2. 生成された `cpp/main.exe` を確認する.
3. `cd cpp` のあと `./main.exe` でシミュレーションを開始.

誤って `cpp` ディレクトリで `make` すると依存ファイルが見つからず失敗するので注意.

3　主要エージェントとメッセージ

エージェント同士は直接関数を呼び出さず，**イベントバス** EventBus を介したメッセージ駆動型で連携する．送受信されるメッセージの種類をコード 3 に示す．

■ メッセージ種別（cpp/include/message.h）

```
11 // Enumerates all message categories exchanged among agents.
12 // New entries were added to model the full foundation workflow
13 // (build -> moor -> store -> transport -> assemble -> tow).
14
15 enum class MessageType {
16     TowRequest,
17     TowStart,
18     TowComplete,
19     MooringRequest,
20     InstallComplete,
21     PreAnchorRequest,
22     PreAnchorComplete,
23     AnchorTestRequest,
24     AnchorTestComplete,
25     DockSlotOpen,
26     JoinQueue,
27     MaterialRequest,
28     MaterialReady,
29     BuildComplete,
30     StorageRequest,
31     StorageSlotAssigned,
32     LeaveStorage,
33     AssemblyRequest,
34     AssemblyComplete,
35     ManufactureComplete,
36     CableInstallStart,
37     // extend as needed
38 };
```

11–13 行目　エージェント間でやり取りするメッセージ種別を概説するコメント．

15–37 行目　施工フローを網羅する MessageType 列挙体．曳航開始・完了や倉庫入出庫だけでなく，係留要求（MooringRequest），事前アンカー敷設（PreAnchorRequest/PreAnchorComplete），係留試験（AnchorTestRequest/AnchorTestComplete），ケーブル敷設開始通知（CableInstallStart）など，現行コードで利用するメッセージが全て定義されている．

主要なエージェントと役割を以下にまとめる．

MaterialStockAgent　部材倉庫．MaterialRequest を受け取ると在庫を確認して準備し，完了時に MaterialReady を返す．同一基礎からの重複要求は保留情報として記録され，二重出庫を防止する．

Dock　ドック（製造ヤード）．JoinQueue を受け付けて待機列を管理し，製作完了時には BuildComplete を送信，空きスロットが生じた際は DockSlotOpen をブロードキャストする．

YardStorage　ヤード仮置き場．StorageRequest に対して空きがあれば StorageSlotAssigned を返し，退去要求としての LeaveStorage も処理する．

QuayCrane　岸壁クレーン．AssemblyRequest を受けて組立作業を実施し，完了時に AssemblyComplete を通知する．

TowFleet　牽引艦隊．`TowRequest` を受け取ると曳航を開始し，`TowStart`/`TowComplete` で進捗を知らせる．

PreAnchorFleet　事前アンカー敷設と係留試験を担当し，`PreAnchorRequest`/`PreAnchorComplete` および `AnchorTestRequest`/`AnchorTestComplete` をやり取りする．

MooringFleet　現場での最終係留を担い，`MooringRequest` に応じて係留を実施して `InstallComplete` を返す．

CableLayFleet　洋上ケーブル敷設艦隊．節目ごとに `CableInstallStart` をブロードキャストし，敷設工程の開始を全体に知らせる．

FloatingFoundation　浮体基礎本体．各工程で関連エージェントへ要求を発行し，完了通知を受け取って自らの状態遷移を進める中心的存在である．

4　C++ 実装

4.1　ヘッダファイルの役割

- **agent.h**: すべてのエージェントの基底クラス．位置・速度・イベントバスへの送受信を提供する．
- **environment.h**: 港湾マップ，A* 経路探索，バース占有管理を担当．
- **dock.h**: 浮体基礎の製作スロットを管理し，Fabrication 完了を通知．
- **tow_fleet.h**: 牽引船隊．曳航要求キュー処理と走行シミュレーションを行う．
- **pre_anchor_fleet.h**: 事前アンカー艦隊．敷設および係留試験を担当する．
- **mooring_fleet.h**: 最終係留艦隊．係留ラインの設置と完了通知を行う．
- **cable_lay_fleet.h**: ケーブル敷設艦隊．輸送および敷設進捗を管理する．
- **storage_agent.h**: 仮置き場（ヤード等）の割当て．
- **quay_crane.h**: 岸壁クレーンによる組立作業．
- **material_stock_agent.h** と **material.h**: 部材在庫の発注・消費を管理．
- **event_bus.h**, **message.h**: 非同期メッセージング基盤とメッセージ型．
- **csv_logger.h**: 状態ログを CSV に書き出すユーティリティ．

4.2　主要クラスと関数

代表的な定義を以下に示す．

■ 基底エージェント（cpp/include/agent.h, cpp/src/agent.cpp）

```
class Agent {
protected:
    int id_;
    Vec2 pos_;
    Vec2 vel_;
    double speed_;
    double radius_;
    std::deque<Vec2> route_;
    std::string status_;
    Vec2 dest_;
    std::string dest_name_;
    Vec2 home_pos_;
    std::string home_zone_;
    std::string prevZone_;
    EventBus* bus_;
    static std::shared_ptr<Environment> env_;
    static EventLogger* eventLogger_;
    static SimTime currentTime_;
public:
    Agent(int id, const Vec2& pos = {}, double radius = 0.0);
    virtual ~Agent();
    void setEventBus(EventBus* bus);
    virtual void step(double dt) = 0;
    virtual void receive(const Message& msg) = 0;
    void planRoute(const Environment& env);
    void moveAlongRoute(double dt, const Environment& env);
protected:
    void send(AgentID receiver, MessageType type, Payload payload = {});
    std::string checkZoneArrival(const std::string& label);
```

```
30 };
```

各行の役割を以下に示す．

1 行目　クラス宣言の開始．

2 行目　保護されたメンバ宣言の開始．

3–18 行目　位置・速度・目的地などの状態変数と，イベントバスや共有環境を保持する静的メンバをまとめて定義している．

19–22 行目　コンストラクタ，デストラクタ，イベントバス接続関数．

23–24 行目　派生クラスで実装する純粋仮想関数．

25–26 行目　経路計画と移動処理のユーティリティ関数．

27–28 行目　メッセージ送信とゾーン到達判定の内部関数．

29–30 行目　クラス定義の終端．

以下に各メンバ変数および関数の役割をまとめる．

`id_`　エージェント固有の識別子．

`pos_`　現在位置を表す 2 次元ベクトル．

`vel_`　現在速度ベクトル．

`speed_`　移動速度の大きさ．

`radius_`　占有半径．

`route_`　これから辿る経路をキュー形式で保持する．

`status_`　状態を表す文字列．ログや可視化で利用される．

`dest_`　目的地の座標．

`dest_name_`　目的地ゾーンの名称．

`home_pos_`　基準位置．

`home_zone_`　基準ゾーン名．

`prevZone_`　直前に滞在していたゾーン．ゾーン出入り検知に利用される．

`bus_`　イベントバスへのポインタ．参照にすると必ず接続済みでなければならず空にできない．後から別のバスへ差し替えることもできない．ポインタなら null を入れて未接続を表せるため，起動順を気にせず後で接続でき，テストでもモックを差し込みやすい．さらに複数シナリオで別の EventBus を使い分けたい場合も代入だけで済む．参照型で未接続を表したいときはダミーの EventBus を用意する必要があり，コードが複雑になるのも避けられる．EventBus 自体の作成と破棄は別の場所で行い，ここでは所有権を持たない．

`env_`　すべてのエージェントで共有する環境オブジェクト．

`eventLogger_`　ゾーン到達などを記録するロガーへの静的ポインタ．

`currentTime_`　シミュレーション現在時刻を保持する静的変数．

`step`　毎ステップ呼ばれる純粋仮想関数．派生クラスで行動を定義する．

`receive`　メッセージ受信時に呼ばれる純粋仮想関数．

`planRoute`　A* 探索を用いて経路を計画する．

`moveAlongRoute`　計画済み経路に沿って移動処理を行う．

■ ドック（`cpp/include/dock.h`, `cpp/src/dock.cpp`）

```cpp
class Dock : public Agent {
    int capacity_;
    int processDays_;
    std::queue<AgentID> waiting_;
    struct Task { AgentID id; SimTime remainingMin; };
    std::vector<Task> working_;
    bool broadcastedOpen_;
public:
    Dock(AgentID id, int capacity, int processDays);
    void step(double dt) override;
    void receive(const Message& msg) override;
};
```

各行の意味は以下の通りである．

1 行目　エージェントを継承したドックの宣言．

2–7 行目　製作能力や処理日数，待機キューと進行中タスク，空き通知フラグなどの内部状態．

8 行目　公開メンバ領域の開始．

9 行目　コンストラクタで ID や能力値を設定する．

10 行目　Fabrication 進捗を更新し空きスロットを通知するステップ処理．

11 行目　`JoinQueue` などのメッセージを受信する処理．

12 行目　クラス定義の終端．

`override` は基底クラス `Agent` の仮想関数を上書きしていることをコンパイル時に保証する C++11 の機能である．ここでは `step` と `receive` に付されており，もし関数名や引数の型がひとつでも異なれば派生クラスの新しい関数として処理されず，直ちにコンパイルエラーとなる．このキーワードによってインタフェースの実装が静的に検証され，誤ったシグネチャが紛れ込む余地はない．

基底側では次のように純粋仮想関数として宣言され，派生クラスが必ず実装すべき契約となっている．

```cpp
class Agent {
public:
    virtual void step(double dt) = 0;
    virtual void receive(const Message& msg) = 0;
};

class YardStorage : public Agent {
public:
    void step(double dt) override { /* YardStorage 独自の処理 */ }
    void receive(const Message& msg) override { /* メッセージ受信処理 */ }
    // void step(int dt) override; // 引数を誤るとコンパイルエラー
};
```

`virtual ... = 0` が付いた関数は純粋仮想関数を意味し，`Agent` は抽象クラスとして直接は生成できない．派生側で `override` を指定することでシグネチャ一致が静的に検証され，契約を破る実装はビルド時点で退けられる．

■ 牽引船隊（cpp/include/tow_fleet.h, cpp/src/tow_fleet.cpp）

```cpp
class TowFleet : public Agent {
    int capacity_;
    struct TowReq { AgentID id; Vec2 target; int priority; };
    struct TowUnit {
```

```
5           AgentID id;
6           Vec2 pos;
7           std::vector<Vec2> route;
8           size_t idx;
9           double carry;
10          AgentID foundation;
11          double departRemaining;
12          double arriveRemaining;
13          bool waitingDestBerth;
14          std::string originZone;
15          std::string destZone;
16          std::vector<Vec2> returnRoute;
17          size_t returnIdx;
18          double returnCarry;
19          Vec2 returnGoal;
20          std::string status;
21          std::string destName;
22      };
23      std::deque<TowReq> queue_;
24      std::unordered_set<AgentID> queued_;
25      std::unordered_map<AgentID, std::shared_ptr<FloatingFoundation>> foundations_;
26      std::vector<TowUnit> units_;
27      int dockOccupied_;
28      int yardOccupied_;
29      TowPriorityMode mode_;
30  public:
31      TowFleet(AgentID id, int capacity);
32      void setHome(const Vec2& p, const std::string& zone);
33      void registerFoundation(const std::shared_ptr<FloatingFoundation>& f);
34      void step(double dt) override;
35      void receive(const Message& msg) override;
36      void logSamples(SampleLogger& logger, SimTime minute) const;
37  };
```

各行の意味は以下の通りである．

1 行目　牽引船隊クラスの宣言．

2 行目　同時に捌ける曳航ユニット数を表すメンバ．

3 行目　待機キューへ積む曳航要求の構造体．

4–20 行目　個々の牽引ユニット状態を保持する構造体．航路・進行位置・係留待ちフラグ・往復ルートなど，実装で参照される全属性を持つ．

21 行目　牽引ユニットの表示用目的地名．

22 行目　曳航要求キュー．

23 行目　二重登録を防ぐための待機集合．

24 行目　浮体基礎への参照を保持するマップ．

25 行目　稼働中ユニットの一覧．

26–27 行目　ドックおよびヤードで占有している基礎数のカウンタ．

28 行目　曳航優先度モード．

29 行目　公開メンバ領域の開始．

30 行目　コンストラクタで ID と能力値を設定する．

31 行目　帰還位置などの基準ゾーンを設定する補助関数．

32 **行目**　浮体基礎を登録するメソッド．共有ポインタを受け取って内部のマップへ記録する．
33 **行目**　各ステップで位置更新や進行管理を行う．
34 **行目**　メッセージ受信を処理する．
35 **行目**　各曳航ユニットの位置情報をロガーへ出力する補助関数．
36 **行目**　クラス定義の終端．

■ 浮体基礎（cpp/include/floating_foundation.h, cpp/src/floating_foundation.cpp）

```
enum class FoundationStage { None, Fabrication, DockTow, YardStorage,
    Assembly, SiteTowPending, SiteTow, Mooring, Moored, Completed };

class FloatingFoundation : public Agent {
    FoundationStage stage_;
    int remainingDays_;
    AgentID dock_;
    AgentID tow_;
    AgentID mooring_;
    AgentID preAnchorFleet_;
    AgentID yardStorage_;
    AgentID quay_;
    AgentID material_;
    Bom bom_;
    Vec2 siteTarget_;
    bool started_;
    bool yardSlot_;
    bool waitingStorageSlot_;
    bool waitingTowStart_;
    bool waitingTowComplete_;
    bool preAnchored_;
    bool anchorTested_;
    bool assemblyDone_;
public:
    FloatingFoundation(int id, AgentID dock, AgentID tow, AgentID mooring,
        AgentID preAnchorFleet, AgentID yardStorage, AgentID quay,
        AgentID material, const Vec2& siteTarget, Bom bom);
    void step(double dt) override;
    void receive(const Message& msg) override;
    void setStage(FoundationStage s, int duration);
    void progressOneDay();
    FoundationStage stage() const { return stage_; }
    bool isCompleted() const { return stage_ == FoundationStage::Completed; }
};
```

FoundationStage は enum class で定義された工程段階の列挙である．enum class は C++11 で導入された強い列挙型であり，列挙子が型のスコープ内に閉じるため名前空間を汚さない．また整数型への暗黙変換を行わず，異なる列挙体や数値との取り違えをコンパイル時に防げるため，状態遷移を安全に記述できる．例えば従来の enum のように整数として代入したり別の列挙と比較したりすることができないので，意図しない値が混入しても直ちにエラーで気付ける．小さなステージの違いも厳密に扱えるため，研究者が状態遷移を追う際の安心感が高まる．

浮体基礎は各工程を状態遷移で管理する．

1–2 **行目**　工程段階を列挙する FoundationStage 型．
4 **行目**　浮体基礎クラスの宣言．

5–21 行目　工程状態や関連エージェント ID，資材情報，工程フラグを保持するメンバ変数群．係留艦隊や事前アンカー艦隊の ID，アンカー試験済みフラグなど現行コードで追加された要素も含む．

22 行目　公開メンバ領域の開始．

23–25 行目　コンストラクタで関連エージェントや目標地点を設定する．

26 行目　ステップ処理．

27 行目　メッセージ受信処理．

28–29 行目　工程の明示更新や残日数進行を行う補助関数．

30–31 行目　現在ステージと施工完了判定のユーティリティ．

32 行目　クラス定義の終端．

■ main.cpp の骨格（cpp/main.cpp）

```
102     auto parseStatusInterval = [&](const std::string& value) {
103         try {
104             long long parsed = std::stoll(value);
105             if (parsed <= 0) {
106                 std::cerr << "Status interval must be a positive integer number of minutes." <<
                        std::endl;
107                 printUsage(argv[0]);
108                 std::exit(EXIT_FAILURE);
109             }
110             options.statusIntervalMinutes = static_cast<SimTime>(parsed);
111         } catch (const std::invalid_argument&) {
112             std::cerr << "Invalid status interval value '" << value << "'." << std::endl;
113             printUsage(argv[0]);
114             std::exit(EXIT_FAILURE);
115         } catch (const std::out_of_range&) {
116             std::cerr << "Status interval value out of range: '" << value << "'." << std::endl;
117             printUsage(argv[0]);
118             std::exit(EXIT_FAILURE);
119         }
120     };
121
122     for (int i = 1; i < argc; ++i) {
123         std::string arg(argv[i]);
124         if (arg == "--help" || arg == "-h") {
125             printUsage(argv[0]);
126             std::exit(EXIT_SUCCESS);
127         }
128         if (arg.rfind("--status-log-format=", 0) == 0) {
129             std::string value = normalize(arg.substr(std::strlen("--status-log-format=")));
130             if (value == "fleet") {
131                 options.logFormat = StatusLogFormat::Fleet;
132             } else if (value == "unit") {
133                 options.logFormat = StatusLogFormat::Unit;
134             } else {
135                 std::cerr << "Unknown status log format '" << value << "'." << std::endl;
136                 printUsage(argv[0]);
137                 std::exit(EXIT_FAILURE);
138             }
139             continue;
140         }
141         if (arg == "--status-log-format") {
142             if (i + 1 >= argc) {
```

```
143                 std::cerr << "Missing value for --status-log-format option." << std::endl;
144                 printUsage(argv[0]);
145                 std::exit(EXIT_FAILURE);
146             }
147             std::string value = normalize(argv[++i]);
148             if (value == "fleet") {
149                 options.logFormat = StatusLogFormat::Fleet;
150             } else if (value == "unit") {
151                 options.logFormat = StatusLogFormat::Unit;
152             } else {
153                 std::cerr << "Unknown status log format '" << value << "'." << std::endl;
154                 printUsage(argv[0]);
155                 std::exit(EXIT_FAILURE);
156             }
157             continue;
158         }
159 
160         if (arg.rfind("--status-clear=", 0) == 0) {
161             std::string value = normalize(arg.substr(std::strlen("--status-clear=")));
162             if (value == "on") {
163                 options.clearMode = StatusDisplay::ClearMode::On;
164             } else if (value == "off") {
165                 options.clearMode = StatusDisplay::ClearMode::Off;
166             } else {
167                 std::cerr << "Unknown status clear mode '" << value << "'." << std::endl;
168                 printUsage(argv[0]);
169                 std::exit(EXIT_FAILURE);
170             }
171             continue;
172         }
173         if (arg == "--status-clear") {
174             if (i + 1 >= argc) {
175                 std::cerr << "Missing value for --status-clear option." << std::endl;
176                 printUsage(argv[0]);
177                 std::exit(EXIT_FAILURE);
178             }
179             std::string value = normalize(argv[++i]);
180             if (value == "on") {
181                 options.clearMode = StatusDisplay::ClearMode::On;
182             } else if (value == "off") {
183                 options.clearMode = StatusDisplay::ClearMode::Off;
184             } else {
185                 std::cerr << "Unknown status clear mode '" << value << "'." << std::endl;
186                 printUsage(argv[0]);
187                 std::exit(EXIT_FAILURE);
188             }
189             continue;
190         }
191 
192         if (arg.rfind("--status-interval=", 0) == 0) {
193             parseStatusInterval(arg.substr(std::strlen("--status-interval=")));
194             continue;
195         }
196         if (arg == "--status-interval") {
197             if (i + 1 >= argc) {
198                 std::cerr << "Missing value for --status-interval option." << std::endl;
199                 printUsage(argv[0]);
200                 std::exit(EXIT_FAILURE);
201             }
```

```
            parseStatusInterval(argv[++i]);
            continue;
        }

        std::cerr << "Unknown argument '" << arg << "'." << std::endl;
        printUsage(argv[0]);
        std::exit(EXIT_FAILURE);
    }
    return options;
}

StatusIntervalResolution resolveStatusInterval(SimTime requested, SimTime stepMin)
{
    StatusIntervalResolution result{requested, {}};
```

102–118 行目　ドックと牽引艦隊を生成し，初期位置やホームゾーンを設定する．

120–145 行目　事前アンカー艦隊・係留艦隊・ケーブル敷設艦隊を初期化し，配置や目的地をログに出力する．

147–169 行目　ヤード仮置き場，岸壁クレーン，部材倉庫を生成してイベントバスへ接続する．

171–195 行目　浮体基礎を初期投入し，`PreAnchorRequest` の送信やトラッキング用ベクタへの登録を行う．

199–215 行目　イベント／サンプルロガーを準備し，型名の整形に利用するデマングル関数を定義する．

```
    if (result.value <= 0) {
        result.warnings.push_back(
            "Warning: status interval must be positive. Using step interval (" +
            std::to_string(stepMin) + " minutes).");
        result.value = stepMin;
    }

    if (result.value % stepMin != 0) {
        const SimTime original = result.value;
        const SimTime multiplier =
            std::max<SimTime>(SimTime{1}, (original + stepMin - 1) / stepMin);
        const SimTime adjusted = multiplier * stepMin;
        result.warnings.push_back(
            "Warning: status interval " + std::to_string(original) +
            " is not a multiple of the simulation step (" +
            std::to_string(stepMin) + " minutes). Adjusting to " +
            std::to_string(adjusted) + " minutes.");
        result.value = adjusted;
    }

    return result;
}

void waitForValidationReview(int seconds, char skipKey)
{
    if (seconds <= 0) {
        return;
    }

    std::cout.flush();
    const auto deadline = std::chrono::steady_clock::now() + std::chrono::seconds(seconds);

#ifdef _WIN32
    if (_isatty(_fileno(stdin)) == 0) {
```

```
250         std::this_thread::sleep_for(std::chrono::seconds(seconds));
251         return;
252     }
253 
254     while (std::chrono::steady_clock::now() < deadline) {
255         if (_kbhit()) {
256             int c = _getch();
257             if (c == skipKey) {
258                 while (_kbhit()) {
259                     int next = _getch();
260                     if (next == '\r' || next == '\n') {
261                         break;
262                     }
263                 }
264                 break;
265             }
266             if (c == '\r' || c == '\n') {
267                 break;
268             }
269         }
270         std::this_thread::sleep_for(std::chrono::milliseconds(50));
271     }
272 #else
273     if (!isatty(STDIN_FILENO)) {
274         std::this_thread::sleep_for(std::chrono::seconds(seconds));
275         return;
276     }
277 
278     termios original{};
279     if (tcgetattr(STDIN_FILENO, &original) != 0) {
280         std::this_thread::sleep_for(std::chrono::seconds(seconds));
281         return;
282     }
283 
284     termios modified = original;
285     modified.c_lflag &= static_cast<tcflag_t>(~(ICANON | ECHO));
286     if (tcsetattr(STDIN_FILENO, TCSANOW, &modified) != 0) {
287         std::this_thread::sleep_for(std::chrono::seconds(seconds));
288         return;
289     }
290 
291     const int originalFlags = fcntl(STDIN_FILENO, F_GETFL, 0);
292     if (originalFlags == -1) {
293         tcsetattr(STDIN_FILENO, TCSANOW, &original);
294         std::this_thread::sleep_for(std::chrono::seconds(seconds));
295         return;
296     }
297     fcntl(STDIN_FILENO, F_SETFL, originalFlags | O_NONBLOCK);
298 
299     bool stop = false;
300     while (!stop && std::chrono::steady_clock::now() < deadline) {
301         char c;
302         const ssize_t bytesRead = ::read(STDIN_FILENO, &c, 1);
303         if (bytesRead > 0) {
304             if (c == skipKey || c == '\n') {
305                 if (c != '\n') {
306                     while (::read(STDIN_FILENO, &c, 1) > 0 && c != '\n') {
307                     }
308                 }
```

216–238 **行目**　メインループの骨格．現在時刻を設定してメッセージを配送し，環境更新と各エージェントのステップ処理を行う．

239–255 **行目**　サンプルログ出力と曳航艦隊の個別ログを記録する部分．

256–275 **行目**　完了した基礎の検出・イベント出力・リストからの削除，ケーブル作業の完了判定を実施する．

276–297 **行目**　現在時刻や各艦隊の状況を整形し，`StatusDisplay` へ表示更新する．

298–308 **行目**　全基礎とケーブル作業が完了したらループを抜け，最終表示と標準出力の復元を行う．

4.3　通信基盤クラスの詳細

4.3.1　EventBus ヘッダ（cpp/include/event_bus.h）

```
1  #pragma once
2  #include <queue>
3  #include <memory>
4  #include <vector>
5  #include "message.h"
6
7  class Agent; // forward declaration
8  class Environment; // forward declaration
9
10 class EventBus {
11     struct ByDue {
12         bool operator()(const Message& a, const Message& b) const {
13             return a.due > b.due;
14         }
15     };
16
17     std::priority_queue<Message, std::vector<Message>, ByDue> calendar_;
18
19 public:
20     void post(const Message& msg);
21     void postAt(SimTime due, const Message& msg);
22     void postIn(SimTime delta, Message msg);
23     void deliverAll(const std::vector<std::shared_ptr<Agent>>& agents,
24                     Environment* env = nullptr);
25 };
```

1 **行目**　ヘッダ多重読み込み防止．

2–5 **行目**　標準ライブラリとメッセージ型のインクルード．

7–8 **行目**　`Agent` と `Environment` を前方宣言し，完全な定義を必要としないことを示す．

10 **行目**　イベント配送を司る `EventBus` クラスの宣言開始．

11–15 **行目**　メッセージの予定時刻でソートする比較関数オブジェクト．

17 **行目**　予定時刻付きメッセージを保持する優先度付きキュー．

19 **行目**　公開メソッドの開始．

20 **行目**　即時配送用の `post` 関数．

21 **行目**　絶対時刻を指定して投入する `postAt` 関数．

22 **行目**　相対時間を指定して投入する `postIn` 関数．

23–24 行目　キュー内のメッセージを時刻順に配送する `deliverAll` 関数.

25 行目　クラス定義の終端.

4.3.2　EventBus 実装（cpp/src/event_bus.cpp）

```
#include "event_bus.h"
#include "agent.h"
#include "environment.h"

void EventBus::post(const Message& msg) {
    postAt(Agent::currentTime(), msg);
}

void EventBus::postAt(SimTime due, const Message& msg) {
    Message copy = msg;
    copy.due = due;
    calendar_.push(copy);
}

void EventBus::postIn(SimTime delta, Message msg) {
    postAt(Agent::currentTime() + delta, msg);
}

void EventBus::deliverAll(const std::vector<std::shared_ptr<Agent>>& agents,
                          Environment* env) {
    const SimTime currentTime = Agent::currentTime();
    while (!calendar_.empty()) {
        const Message& next = calendar_.top();
        if (next.due > currentTime) {
            break;
        }
        Message m = next;
        calendar_.pop();
        Agent::setCurrentTime(m.due);
        if (m.receiver == SYSTEM_ID) {
            if (env) env->handleMessage(m);
        } else if (m.receiver == BROADCAST_ID) {
            for (auto& a : agents) {
                if (a->id() != m.sender) {
                    a->receive(m);
                }
            }
        } else {
            for (auto& a : agents) {
                if (a->id() == m.receiver) {
                    a->receive(m);
                    break;
                }
            }
        }
    }
}
```

1–3 行目　必要なヘッダをインクルードする.

5–7 行目　`post` は現在時刻で `postAt` を呼び出す薄いラッパーである.

9–13 行目　postAt は実行時刻を付与して優先度付きキューに積む．

15–16 行目　postIn は相対時間を絶対時刻に換算し postAt を利用する．

19–42 行目　deliverAll はキューが空になるまで時刻順にメッセージを取り出し，受信者に応じて環境または各エージェントへ配送する．処理のたびに現在時刻を更新する．

4.3.3　Message ヘッダ（cpp/include/message.h）

```cpp
#pragma once
#include <variant>
#include <string>
#include <utility>
#include "vec2.h"
#include "constants.h"

constexpr AgentID BROADCAST_ID = -1;
constexpr AgentID SYSTEM_ID = -2;

// Enumerates all message categories exchanged among agents.
// New entries were added to model the full foundation workflow
// (build -> moor -> store -> transport -> assemble -> tow).

enum class MessageType {
    TowRequest,
    TowStart,
    TowComplete,
    MooringRequest,
    InstallComplete,
    PreAnchorRequest,
    PreAnchorComplete,
    AnchorTestRequest,
    AnchorTestComplete,
    DockSlotOpen,
    JoinQueue,
    MaterialRequest,
    MaterialReady,
    BuildComplete,
    StorageRequest,
    StorageSlotAssigned,
    LeaveStorage,
    AssemblyRequest,
    AssemblyComplete,
    ManufactureComplete,
    CableInstallStart,
    // extend as needed
};

struct Bom {
    int blade;
    int nacelle;
    int tower;
};

using Payload = std::variant<std::monostate, int, double, std::string, Vec2, Bom>;

struct Message {
    SimTime due;
```

```
50      AgentID sender;
51      AgentID receiver;
52      MessageType type;
53      Payload payload;
54
55      Message() = default;
56      Message(AgentID s, AgentID r, MessageType t, Payload p)
57          : due(0), sender(s), receiver(r), type(t), payload(std::move(p)) {}
58      Message(SimTime d, AgentID s, AgentID r, MessageType t, Payload p)
59          : due(d), sender(s), receiver(r), type(t), payload(std::move(p)) {}
60  };
```

1 行目　ヘッダ多重読み込み防止.

2–6 行目　メッセージで利用する型と定数のインクルード.

8–9 行目　特殊な受信者を表す定数.

BROADCAST_ID は全エージェント宛，SYSTEM_ID はシステムからの通知を示す.

11–13 行目　メッセージ種別を網羅するコメント.

15–37 行目　施工フローを網羅する MessageType 列挙体.

40–44 行目　部材の必要数量を表す Bom 構造体.

46 行目　メッセージに添付可能なデータ型を std::variant で表現した Payload 型.

48–59 行目　配送予定時刻と送受信者などを保持する Message 構造体とコンストラクタ群.

4.3.4　Message 実装

Message はデータ保持のみを目的とした単純な構造体であり，挙動を伴う関数を持たない．そのため別途 message.cpp は存在せず，ヘッダのみで完結している.

4.3.5　Agent ヘッダ（cpp/include/agent.h）

```
1   #pragma once
2
3   #include <deque>
4   #include <string>
5   #include <memory>
6   #include <vector>
7   #include "vec2.h"
8   #include "message.h"
9   #include "constants.h"
10
11  class EventBus;
12  class Environment;
13  class EventLogger;
14
15  struct UnitStatus {
16      std::string type;       ///< human-readable unit type label
17      AgentID id;             ///< identifier for the unit (may differ from fleet id)
18      Vec2 position;          ///< current position of the unit
19      std::string destination; ///< textual representation of current destination
20      std::string status;     ///< activity/state description
21  };
22
23  class Agent {
```

```
protected:
    int id_;
    Vec2 pos_;
    Vec2 vel_;
    double speed_;
    double radius_;
    std::deque<Vec2> route_;
    std::string status_;
    std::string segment_;
    Vec2 dest_;
    std::string dest_name_;
    Vec2 home_pos_;
    std::string home_zone_;
    std::string prevZone_;
    EventBus* bus_;
    static std::shared_ptr<Environment> env_;
    static EventLogger* eventLogger_;
    static SimTime currentTime_;

public:
    Agent(int id, const Vec2& pos = {}, double radius = 0.0);
    virtual ~Agent();
    void setEventBus(EventBus* bus);

    virtual void step(double dt) = 0; // dt in hours
    virtual void receive(const Message& msg) = 0;

    int id() const;
    const Vec2& position() const;
    const Vec2& velocity() const;
    double speed() const;
    double radius() const;
    const std::string& status() const;
    const std::string& segment() const;
    const std::deque<Vec2>& route() const;
    const Vec2& destination() const;
    const std::string& destinationName() const;
    const Vec2& homePosition() const;
    const std::string& homeZone() const;

    void setPosition(const Vec2& p);
    void setVelocity(const Vec2& v);
    void setSpeed(double s);
    void setRadius(double r);
    void setStatus(const std::string& s);
    void setSegment(const std::string& s);
    void setRoute(const std::deque<Vec2>& route);
    void setDestination(const Vec2& d);
    void setDestinationName(const std::string& n);
    void setHome(const Vec2& p, const std::string& zone);

    static void setEnvironment(const std::shared_ptr<Environment>& env);
    static std::shared_ptr<Environment> environment();
    static void setEventLogger(EventLogger* logger);
    static EventLogger* eventLogger();
    static void setCurrentTime(SimTime t);
    static SimTime currentTime();

    /// Provide unit-level status information for status displays or logging.
```

```
83      /// The default implementation treats the agent itself as a single unit.
84      /// Fleets managing multiple physical units override this to expose each
85      /// unit individually.
86      virtual std::vector<UnitStatus> unitStatuses() const;
87
88      // Route planning and movement helpers
89      void planRoute(const Environment& env);
90      void moveAlongRoute(double dt, const Environment& env);
91
92  protected:
93      void send(AgentID receiver, MessageType type, Payload payload = {});
94      std::string checkZoneArrival(const std::string& label);
95  };
```

1 行目　ヘッダ多重読み込み防止.

3–8 行目　標準ライブラリと自前のヘッダのインクルード.

10–12 行目　後方依存を避けるための前方宣言.

14 行目　すべてのエージェントが継承する基底クラスの宣言.

15–31 行目　位置・速度・目的地などの状態と，イベントバス・環境・イベントログ・現在時刻の静的メンバ.

33–36 行目　コンストラクタと仮想デストラクタ，イベントバス設定関数．仮想デストラクタを備えることで，‘Agent*‘ 経由で ‘Dock‘ など派生オブジェクトを ‘delete‘ しても派生クラスの解体処理が実行される．もし仮想でなければ ‘ Agent‘ だけが呼ばれ，‘Dock‘ が確保した作業バッファや開いたファイルハンドルは解放されずにメモリやリソースが漏れる．‘Agent* p = new Dock(); delete p;‘ の例のように基底型ポインタで扱う場面でも安全に後始末を連鎖させるための保険である．さらに派生側で ‘ Dock() override‘ と書けば，‘ Agent()‘ と一致しないシグネチャをコンパイラが即座に検出し，誤ったデストラクタの定義を未然に防げる．例えば ‘ Dock(int)‘ のように引数を誤って付けた場合でも，‘override‘ がなければ独立の関数として静かに受け入れられるが，指定しておけばその時点でエラーとなり事故を未然に遮断する.

38–39 行目　派生クラスで実装すべき純粋仮想関数.

`step` は時間発展,

`receive` はメッセージ受信時の処理.

41–51 行目　各種ゲッター．状態を参照するためのインターフェース.

53–61 行目　対応するセッター群．位置や目的地などを設定する.

63–68 行目　環境・イベントロガー・現在時刻を共有する静的関数群.

70–72 行目　経路計画と移動処理の補助関数.

74–76 行目　メッセージ送信とゾーン到達判定の内部ユーティリティ.

77 行目　クラス定義の終了.

4.3.6　Agent 実装（cpp/src/agent.cpp）

```
1  #include "agent.h"
2  #include "event_bus.h"
3  #include "environment.h"
4  #include "constants.h"
5  #include "csv_logger.h"
6  #include "route_utils.h"
7  #include <iostream>
8  #include <cassert>
```

```cpp
#include <sstream>
#include <typeinfo>

std::shared_ptr<Environment> Agent::env_ = nullptr;
EventLogger* Agent::eventLogger_ = nullptr;
SimTime Agent::currentTime_ = 0;

Agent::Agent(int id, const Vec2& pos, double radius)
    : id_(id), pos_(pos), vel_(0.0, 0.0), speed_(0.0), radius_(radius),
      status_("idle"), segment_(""), dest_(pos), dest_name_(""), home_pos_(pos), home_zone_(""),
      prevZone_(""), bus_(nullptr) {}

Agent::~Agent() = default;

void Agent::setEventBus(EventBus* bus) { bus_ = bus; }

int Agent::id() const { return id_; }
const Vec2& Agent::position() const { return pos_; }
const Vec2& Agent::velocity() const { return vel_; }
double Agent::speed() const { return speed_; }
double Agent::radius() const { return radius_; }
const std::string& Agent::status() const { return status_; }
const std::string& Agent::segment() const { return segment_; }
const std::deque<Vec2>& Agent::route() const { return route_; }
const Vec2& Agent::destination() const { return dest_; }
const std::string& Agent::destinationName() const { return dest_name_; }
const Vec2& Agent::homePosition() const { return home_pos_; }
const std::string& Agent::homeZone() const { return home_zone_; }

void Agent::setPosition(const Vec2& p) { pos_ = p; }
void Agent::setVelocity(const Vec2& v) { vel_ = v; }
void Agent::setSpeed(double s) { speed_ = s; }
void Agent::setRadius(double r) { radius_ = r; }
void Agent::setStatus(const std::string& s) { status_ = s; }
void Agent::setSegment(const std::string& s) { segment_ = s; }
void Agent::setRoute(const std::deque<Vec2>& route) { route_ = route; }
void Agent::setDestination(const Vec2& d) { dest_ = d; }
void Agent::setDestinationName(const std::string& n) { dest_name_ = n; }
void Agent::setHome(const Vec2& p, const std::string& zone) { home_pos_ = p; home_zone_ = zone; }

void Agent::setEnvironment(const std::shared_ptr<Environment>& env) { env_ = env; }
std::shared_ptr<Environment> Agent::environment() { return env_; }

void Agent::setEventLogger(EventLogger* logger) { eventLogger_ = logger; }
EventLogger* Agent::eventLogger() { return eventLogger_; }

void Agent::setCurrentTime(SimTime t) { currentTime_ = t; }
SimTime Agent::currentTime() { return currentTime_; }

std::vector<UnitStatus> Agent::unitStatuses() const
{
    UnitStatus unit;
    unit.type = typeid(*this).name();
    unit.id = id_;
    unit.position = pos_;
    if (!dest_name_.empty()) {
        unit.destination = dest_name_;
    } else {
        std::ostringstream oss;
```

```
68          oss << '(' << dest_.x << ',' << dest_.y << ')';
69          unit.destination = oss.str();
70      }
71      unit.status = status_;
72      return {unit};
73  }
74
75  void Agent::send(AgentID receiver, MessageType type, Payload payload) {
76      if (bus_) {
77          bus_->post({id_, receiver, type, payload});
78      }
79  }
80
81  void Agent::planRoute(const Environment& env) {
82      route_ = env.findPath(pos_, dest_);
83  }
84
85  void Agent::moveAlongRoute(double dt, const Environment& env) {
86      (void)env;
87      if (route_.size() < 2) {
88          setVelocity({0.0, 0.0});
89          if (route_.empty()) setStatus("idle");
90          return;
91      }
92
93      double stepDist = speed_ * dt;
94      Vec2 startPos = pos_;
95      std::vector<Vec2> poly(route_.begin(), route_.end());
96      size_t idx = 0;
97      double carry = 0.0;
98      advanceAlongRoute(pos_, poly, idx, carry, stepDist);
99      Vec2 delta = pos_ - startPos;
100 #ifndef NDEBUG
101     double maxDist = stepDist * (1.0 + SPEED_EPS);
102     assert(delta.length() <= maxDist + 1e-9);
103 #endif
104     setVelocity(delta * (1.0 / dt));
105     for (size_t i = 0; i < idx; ++i) {
106         route_.pop_front();
107     }
108     if (route_.size() <= 1) {
109         route_.clear();
110         setStatus("arrived");
111     } else {
112         route_.front() = pos_;
113         setStatus("moving");
114     }
115 }
116
117 std::string Agent::checkZoneArrival(const std::string& label) {
118     if (!env_) return "";
119     std::string currZone = env_->zoneName(pos_.x, pos_.y);
120     if (currZone != prevZone_ && !currZone.empty()) {
121         if (eventLogger_) {
122             eventLogger_->log(currentTime_, label + "#" + std::to_string(id_),
123                               "zone_arrival", currZone);
124         }
125         prevZone_ = currZone;
126         return currZone;
```

```
127     }
128     return "";
129 }
```

1–8 行目　ヘッダや定数，デバッグ用の標準出力を利用するためのインクルード．

10–12 行目　環境ポインタ，イベントロガー，現在時刻の静的メンバ初期化．

14–17 行目　コンストラクタでは各種メンバを初期化し，初期状態を設定する．

19 行目　デストラクタはデフォルト実装．

21 行目　イベントバスのポインタを設定するメンバ関数．

23–33 行目　各種ゲッターで状態を参照する．

35–43 行目　セッター群で状態を更新する．

45–52 行目　環境オブジェクトやイベントログ，現在時刻の静的セッター／ゲッター．

54–58 行目　`send` はイベントバスにメッセージを投げるユーティリティ．

60–61 行目　`planRoute` は環境に経路探索を依頼して移動経路を設定する．

64–94 行目　`moveAlongRoute` は経路に沿って移動し，障害物がある場合は回避を試みる．最終的に速度と位置，状態を更新する．

96–107 行目　`checkZoneArrival` はゾーンの出入りを検知し，初めて入ったゾーンをログ出力する．

■ 定数定義（cpp/include/constants.h）[*5]

```
1  #pragma once
2
3  #include <cstdint>
4  #include <cmath>
5
6  // Time conversions
7  using SimTime = int64_t; // minutes
8  using AgentID = int;
9  constexpr int HOURS_PER_DAY = 24;
10 constexpr double TIME_STEP_HOURS = 0.1;
11 inline SimTime hours_to_min(double h) {
12     return static_cast<SimTime>(std::llround(h * 60));
13 }
14
15 // Movement parameters
16 constexpr double BASE_SPEED = 1.0;            // units moved per hour
17 constexpr double MOVE_THRESHOLD = 0.1;        // minimum distance to advance route
18 constexpr double SNAP_DISTANCE = 1e-3;        // snapping distance when near waypoint
19 constexpr double SPEED_EPS = 1e-6;            // allowable relative overshoot per step
20
21 // Reserved agent identifiers
22 constexpr AgentID DOCK_ID = 10;
23 constexpr AgentID TOW_FLEET_ID = 15;
24 constexpr AgentID MOORING_FLEET_ID = 20;
25 constexpr AgentID PRE_ANCHOR_FLEET_ID = 21;
26 constexpr AgentID CABLE_LAY_FLEET_ID = 25;
27 constexpr AgentID YARD_STORAGE_ID = 31;
```

[*5] 2025 年 8 月 4 日の定数整理（commit `d415c0113cffafc3703fea388b8d2729f4d583ab`）において，時間上限を表していた `MAXHOUR` や刻み幅を示す `TIME_STEP` など単位が曖昧な名称を廃し，上限を日数で管理する `MAX_SIM_DAYS` と時間刻みを明示する `TIME_STEP_HOURS` へ改称した．これにより設定値の意味が一貫し，長期シミュレーションでの単位混同やオーバーフローの危険を回避できる．

```
28 constexpr AgentID QUAY_CRANE_ID = 50;
29 constexpr AgentID MATERIAL_STOCK_ID = 60;
30
31 // Simulation configuration
32 constexpr int NUM_FOUNDATIONS = 10;
33 constexpr int FOUNDATION_MAKE_LINES = 1;
34 constexpr int FOUNDATION_MAKE_DAYS = 20;
35 constexpr int INITIAL_DOCK_STOCK = 3;
36 constexpr int YARD_STORAGE_CAPACITY = 5;
37 constexpr int TOW_CAPACITY = 3;
38 constexpr int MOORING_CAPACITY = 2;
39 constexpr int PREANCHOR_CAPACITY = 1;
40 constexpr int ANCHORS_PER_FOUNDATION = 3;
41 constexpr int PREANCHOR_DAYS_PER_ANCHOR = 1;
42 constexpr int ANCHOR_TEST_DAYS = 1;
43
44 // Task durations
45 constexpr int MAX_SIM_DAYS = 10000;
46
47 constexpr int ASSEMBLY_LINES = 3;
48 constexpr int ASSEMBLY_DAYS = 3;
49 constexpr int MOORING_DAYS_PER_LINE = 2;
50
51 // Towing parameters
52 constexpr double TOW_SPEED_SEA = 1.0;        // speed in open sea
53 constexpr double TOW_SPEED_PORT = 0.5;       // speed within port zones
54 constexpr double MOORING_HOURS = 2.0;        // time for mooring/unmooring
55 constexpr int DOCK_BERTHS = 2;               // available berths at the dock
56 constexpr int YARD_BERTHS = 4;               // available berths at the yard
57
58 // Cable installation parameters
59 constexpr int CABLE_EXPORT_COUNT = 1;
60 constexpr int CABLE_ARRAY_COUNT = 9;
61 constexpr double CABLE_EXPORT_LENGTH_KM = 5.0;
62 constexpr double CABLE_ARRAY_LENGTH_KM = 1.5;
63 constexpr double CABLE_LAY_SPEED_KM_PER_DAY = 10.0;
64 constexpr double CABLE_RISER_CONNECT_DAYS = 0.5;
65 constexpr double CABLE_EMBED_LENGTH_M = 1000.0;
66 constexpr double CABLE_EMBED_SPEED_M_PER_DAY = 200.0;
67 constexpr int CABLE_LAY_SHIP_COUNT = 1;
68
69 enum class CableMode { Scalar, Route };
70 constexpr CableMode CABLE_MODE = CableMode::Route;
71 constexpr const char* CABLE_ROUTES_FILE = "../data/maps/port_map_detailed.json";
72
73 enum class TowPriorityMode { FIFO, Score };
74 constexpr TowPriorityMode TOW_PRIORITY_MODE = TowPriorityMode::FIFO;
75
76 constexpr int BLADE_MAX = 5;
77 constexpr int NACELLE_MAX = 5;
78 constexpr int TOWER_MAX = 5;
79
80 constexpr int IMPORT_BATCH_BLADE = 4;
81 constexpr int IMPORT_BATCH_NACELLE = 4;
82 constexpr int IMPORT_BATCH_TOWER = 4;
83 constexpr bool USE_RANDOM_IMPORT_LEAD = true;
84 constexpr unsigned int IMPORT_RNG_SEED = 42;
85
86 constexpr int IMPORT_LEAD_BLADE = 10;
```

```
constexpr int IMPORT_LEAD_NACELLE = 5;
constexpr int IMPORT_LEAD_TOWER = 30;
constexpr int IMPORT_LEAD_DEV_BLADE = 5;
constexpr int IMPORT_LEAD_DEV_NACELLE = 5;
constexpr int IMPORT_LEAD_DEV_TOWER = 5;

enum class ReorderPolicy { Basic, ReorderPointSystem };
constexpr ReorderPolicy REORDER_POLICY = ReorderPolicy::ReorderPointSystem;
constexpr double SAFETY_Z = 1.65;
constexpr int DEMAND_MA_HOURS = 168;
constexpr int MAX_PIPELINE_ORDERS = 3;
```

1 行目　`#pragma once` により重複インクルードを防ぐ．

3–4 行目　時間計算に必要な標準ヘッダを読み込む．

6–13 行目　シミュレーション時間に関する基本型と変換関数を定義する．

15–19 行目　移動に関するパラメータを設定する（スナップ距離や許容誤差を含む）．

21–28 行目　各エージェントに割り当てられた固定 ID を定義する．

30–41 行目　基礎数や製作ライン数，能力値などシミュレーション設定値をまとめる．

43–49 行目　タスク所要日数と組立・係留に関する定数群．

51–56 行目　曳航速度や係留時間など移動関連パラメータ．

58–67 行目　ケーブル敷設に関するパラメータを定義する．

69–71 行目　ケーブル敷設モードとルート定義ファイルを指定する．

73–74 行目　曳航要求の優先モードを列挙型と定数で指定する．

76–78 行目　各部材の最大在庫量．

80–84 行目　輸入バッチサイズとリードタイム乱数設定．

86–91 行目　部材ごとのリードタイム平均とばらつき．

93–97 行目　発注ポリシーや安全在庫係数など在庫制御用定数．

■ CSV ロガー ヘッダ（cpp/include/csv_logger.h）

```
#ifndef CSV_LOGGER_H
#define CSV_LOGGER_H

#include <string>
#include <vector>
#include <fstream>

class CSVLogger {
public:
    CSVLogger(const std::string& path, const std::vector<std::string>& header);
    void appendRow(const std::vector<std::string>& row);
    void appendRow(const std::vector<int>& row);
private:
    std::ofstream ofs_;
};

class EventLogger {
public:
    EventLogger(const std::string& path);
    void log(int time, const std::string& entity,
```

```
21              const std::string& event, const std::string& detail);
22 private:
23     CSVLogger logger_;
24 };
25
26 class SampleLogger {
27 public:
28     SampleLogger(const std::string& path);
29     void log(const std::vector<std::string>& row);
30 private:
31     CSVLogger logger_;
32 };
33
34 #endif // CSV_LOGGER_H
```

1–2 行目　伝統的なヘッダガードで多重定義を防止.

4–6 行目　文字列・配列・ファイル出力に必要な標準ヘッダをインクルード.

8–15 行目　CSV ロガー本体の宣言．ファイルオープンと行追加メソッドを提供する.

17–22 行目　イベントログ用の薄いラッパークラスを宣言し，内部で CSVLogger を保持する.

24–29 行目　サンプリングログ用のラッパークラスを宣言し，CSV 書き出しを再利用する.

31 行目　ヘッダガード終端.

■ CSV ロガー 実装（cpp/src/csv_logger.cpp）

```
1  #include "csv_logger.h"
2  #include <sstream>
3
4  CSVLogger::CSVLogger(const std::string& path, const std::vector<std::string>& header)
5      : ofs_(path, std::ios::trunc)
6  {
7      if (ofs_.is_open()) {
8          for (size_t i = 0; i < header.size(); ++i) {
9              ofs_ << header[i];
10             if (i + 1 < header.size()) ofs_ << ',';
11         }
12         ofs_ << '\n';
13     }
14 }
15
16 void CSVLogger::appendRow(const std::vector<std::string>& row)
17 {
18     for (size_t i = 0; i < row.size(); ++i) {
19         ofs_ << row[i];
20         if (i + 1 < row.size()) ofs_ << ',';
21     }
22     ofs_ << '\n';
23     ofs_.flush();
24 }
25
26 EventLogger::EventLogger(const std::string& path)
27     : logger_(path, {"time", "entity", "event", "detail"})
28 {
29 }
30
```

```cpp
void EventLogger::log(int time, const std::string& entity,
                      const std::string& event, const std::string& detail)
{
    logger_.appendRow({std::to_string(time), entity, event, detail});
}

SampleLogger::SampleLogger(const std::string& path)
    : logger_(path, {"time", "id", "type", "x", "y", "zone", "status", "destinationName"})
{
}

void SampleLogger::log(const std::vector<std::string>& row)
{
    logger_.appendRow(row);
}

void CSVLogger::appendRow(const std::vector<int>& row)
{
    for (size_t i = 0; i < row.size(); ++i) {
        ofs_ << row[i];
        if (i + 1 < row.size()) ofs_ << ',';
    }
    ofs_ << '\n';
    ofs_.flush();
}
```

1–2 行目　実装で必要なヘッダをインクルード．

4–13 行目　コンストラクタはファイルを開き，ヘッダ行を書き込む．既存ファイルは切り詰められる．

16–24 行目　文字列ベクタを受け取り，カンマ区切りで書き込む．行末でフラッシュ．

26–33 行目　イベントロガーは列名を固定し，appendRow 経由で 1 行追記するヘルパー．

35–41 行目　サンプルロガーは座標や状態の列名を設定し，CSV 書き出しを委譲する．

43–50 行目　整数ベクタ版も同様に処理し，数値をそのまま出力する．

■ 環境クラス ヘッダ（cpp/include/environment.h）

```cpp
#pragma once
#include <string>
#include <vector>
#include <deque>
#include <queue>
#include <memory>
#include <unordered_map>
#include <cmath>
#include "vec2.h"
#include "message.h"

class EventBus;
class FloatingFoundation;
class YardStorage;
class Agent;

struct Rect {
    double x;
    double y;
```

```
    double w;
    double h;
};

struct Zone : Rect {
    std::string name;
    std::string type;
};

struct Target {
    std::string name;
    Vec2 pos;
};

struct Berth {
    std::string id;
    std::string quay;
    Vec2 pos;
    double depth;
    bool occupied{false};
};

struct Obstacle {
    double cx;
    double cy;
    double w;
    double h;
    double angle;
    bool contains(double x, double y) const {
        double rad = -angle * M_PI / 180.0;
        double s = std::sin(rad);
        double c = std::cos(rad);
        double dx = x - cx;
        double dy = y - cy;
        double xPrime = c * dx - s * dy;
        double yPrime = s * dx + c * dy;
        return std::abs(xPrime) <= w / 2.0 && std::abs(yPrime) <= h / 2.0;
    }
};

class Environment {
public:
    double width{0.0};
    double height{0.0};
    std::vector<Zone> zones;
    std::vector<Obstacle> obstacles;
    std::vector<Target> targets;
    std::unordered_map<std::string, double> speedProfiles;
    std::unordered_map<std::string, std::vector<Berth>> berths;

    bool isOccupied(double x, double y) const;
    bool isTraversable(double x, double y) const;
    bool isInsideZone(const std::string& name, double x, double y) const;
    std::string zoneName(double x, double y) const;
    std::string zoneType(const std::string& name) const;
    bool isWaterType(const std::string& type) const;
    std::string firstZoneNameOfType(const std::string& type) const;
    Vec2 nearestZoneCenterOfType(const Vec2& from, const std::string& type) const;
    Vec2 snapToWaterTarget(const Vec2& rawTarget, std::string* snappedZoneOut) const;
```

```
79      double speedForZone(const std::string& zoneNameOrType) const;
80      std::vector<Vec2> getTargets() const;
81
82      std::vector<int> freeBerths(const std::string& quayName) const;
83      bool acquireBerth(const std::string& quayName, int berthIndex);
84      void releaseBerth(const std::string& quayName, int berthIndex);
85      int pickAnyFreeBerth(const std::string& quayName) const;
86
87      // Find a path using a simple grid-based A* algorithm.
88      std::deque<Vec2> findPath(const Vec2& start, const Vec2& goal, double step = 1.0) const;
89
90      static Environment loadFromJson(const std::string& path);
91
92      // inventory management
93      std::queue<AgentID> dockInventory;
94      std::vector<int> manufacturing;
95      int nextFoundationId{100};
96      int totalFoundations{0};
97      int foundationsCreated{0};
98
99      Vec2 zoneCenter(const std::string& name) const;
100     void handleMessage(const Message& msg);
101     void step(EventBus& bus,
102               std::vector<std::shared_ptr<Agent>>& agents,
103               std::vector<std::shared_ptr<FloatingFoundation>>& foundations,
104               std::shared_ptr<YardStorage> yardStorage,
105               AgentID dockId, AgentID towId, AgentID anchorId,
106               AgentID quayId, AgentID materialId);
107 };
```

1–10 **行目**　基本ヘッダと位置ベクトル，メッセージ定義を読み込む．

12–15 **行目**　他クラスの前方宣言で循環参照を回避．

17–57 **行目**　矩形やゾーン，障害物など環境を構成する構造体を定義．

59 **行目以降**　Environment クラス本体．港湾マップや障害物，速度プロファイルなどを保持し，経路探索やバース管理，在庫処理の関数を提供する．

■ 環境クラス 実装（cpp/src/environment.cpp）

```
1  #include "environment.h"
2  #include <fstream>
3  #include <queue>
4  #include <unordered_map>
5  #include <cmath>
6  #include <nlohmann/json.hpp>
7  #include <algorithm>
8  #include <iostream>
9  #include "constants.h"
10 #include "floating_foundation.h"
11 #include "storage_agent.h"
12 #include "event_bus.h"
13
14 using json = nlohmann::json;
15
16 namespace {
17     bool pointInRect(double x, double y, const Rect& r) {
```

```
        return x >= r.x && x <= r.x + r.w && y >= r.y && y <= r.y + r.h;
    }
}

bool Environment::isOccupied(double x, double y) const {
    if (x < 0 || y < 0 || x > width || y > height) return true;
    for (const auto& o : obstacles) {
        if (o.contains(x, y)) return true;
    }
    return false;
}

bool Environment::isTraversable(double x, double y) const {
    if (isOccupied(x, y)) return false;
    std::string zn = zoneName(x, y);
    if (!isWaterType(zoneType(zn))) return false;
    return true;
}

bool Environment::isInsideZone(const std::string& name, double x, double y) const {
    for (const auto& z : zones) {
        if (z.name == name && pointInRect(x, y, z)) return true;
    }
    return false;
}

std::string Environment::zoneName(double x, double y) const {
    for (const auto& z : zones) {
        if (pointInRect(x, y, z)) return z.name;
    }
    return "";
}

std::string Environment::zoneType(const std::string& name) const {
    for (const auto& z : zones) {
        if (z.name == name) return z.type;
    }
    return "";
}

bool Environment::isWaterType(const std::string& t) const {
    return (t == "sea" || t == "channel" || t == "basin" || t == "dock");
}

std::string Environment::firstZoneNameOfType(const std::string& type) const {
    for (const auto& z : zones) if (z.type == type) return z.name;
    return "";
}

Vec2 Environment::nearestZoneCenterOfType(const Vec2& from, const std::string& type) const {
    double best = 1e300;
    Vec2 bestPos{};
    for (const auto& z : zones) if (z.type == type) {
        Vec2 c{z.x + z.w / 2.0, z.y + z.h / 2.0};
        Vec2 diff = c - from;
        double d2 = diff.x * diff.x + diff.y * diff.y;
        if (d2 < best) {
            best = d2;
            bestPos = c;
```

```
        }
    }
    return bestPos;
}

Vec2 Environment::snapToWaterTarget(const Vec2& raw, std::string* zoneOut) const {
    std::string zn = zoneName(raw.x, raw.y);
    std::string zt = zoneType(zn);
    if (isWaterType(zt)) {
        if (zoneOut) *zoneOut = zn;
        return raw;
    }
    static const char* order[] = {"dock", "basin", "channel", "sea"};
    for (const char* t : order) {
        Vec2 c = nearestZoneCenterOfType(raw, t);
        if ((c.x != 0.0 || c.y != 0.0) || !firstZoneNameOfType(t).empty()) {
            if (zoneOut) *zoneOut = firstZoneNameOfType(t);
            return c;
        }
    }
    if (zoneOut) *zoneOut = zn;
    return raw;
}

double Environment::speedForZone(const std::string& zoneNameOrType) const {
    auto it = speedProfiles.find(zoneNameOrType);
    if (it != speedProfiles.end()) return it->second;

    std::string type = zoneType(zoneNameOrType);
    if (!type.empty()) {
        auto tIt = speedProfiles.find(type);
        if (tIt != speedProfiles.end()) return tIt->second;
        if (type == "sea" || type == "Sea") return TOW_SPEED_SEA;
    } else if (zoneNameOrType == "sea" || zoneNameOrType == "Sea") {
        return TOW_SPEED_SEA;
    }
    return TOW_SPEED_PORT;
}

std::vector<Vec2> Environment::getTargets() const {
    std::vector<Vec2> out;
    for (const auto& t : targets) {
        out.push_back(t.pos);
    }
    return out;
}

std::deque<Vec2> Environment::findPath(const Vec2& start, const Vec2& goal, double step) const {
    struct Node {
        int x;
        int y;
        double f;
    };
    auto cmp = [](const Node& a, const Node& b) { return a.f > b.f; };
    std::priority_queue<Node, std::vector<Node>, decltype(cmp)> open(cmp);

    auto hash = [](int x, int y) {
        return (static_cast<long long>(x) << 32) ^ static_cast<unsigned long long>(y);
    };
```

```
    std::unordered_map<long long, Vec2> came;
    std::unordered_map<long long, double> g;

    auto sx = static_cast<int>(std::round(start.x / step));
    auto sy = static_cast<int>(std::round(start.y / step));
    auto gx = static_cast<int>(std::round(goal.x / step));
    auto gy = static_cast<int>(std::round(goal.y / step));

    auto h = [&](int x, int y) {
        return std::hypot(x - gx, y - gy);
    };

    open.push({sx, sy, h(sx, sy)});
    g[hash(sx, sy)] = 0.0;

    const int dirs[4][2] = {{1,0},{-1,0},{0,1},{0,-1}};

    while (!open.empty()) {
        Node cur = open.top();
        open.pop();
        if (cur.x == gx && cur.y == gy) {
            std::deque<Vec2> path;
            long long key = hash(cur.x, cur.y);
            while (came.count(key)) {
                Vec2 p = came[key];
                path.push_front(p);
                key = hash(static_cast<int>(std::round(p.x / step)),
                           static_cast<int>(std::round(p.y / step)));
            }
            path.push_front(Vec2{sx * step, sy * step});
            path.push_back(Vec2{gx * step, gy * step});
            return path;
        }

        for (auto& d : dirs) {
            int nx = cur.x + d[0];
            int ny = cur.y + d[1];
            Vec2 world{nx * step, ny * step};
            if (!isTraversable(world.x, world.y)) continue;
            double tentative = g[hash(cur.x, cur.y)] + 1.0;
            long long key = hash(nx, ny);
            if (!g.count(key) || tentative < g[key]) {
                g[key] = tentative;
                came[key] = Vec2{cur.x * step, cur.y * step};
                double f = tentative + h(nx, ny);
                open.push({nx, ny, f});
            }
        }
    }
    return {};
}

Environment Environment::loadFromJson(const std::string& path) {
    Environment env;
    std::ifstream ifs(path);
    if (!ifs.is_open()) return env;
    json j;
    ifs >> j;
    env.width = j.value("width", 0.0);
```

```
    env.height = j.value("height", 0.0);
    for (const auto& z : j["zones"]) {
        Zone zone;
        zone.name = z.value("name", "");
        zone.type = z.value("type", "");
        zone.x = z.value("x", 0.0);
        zone.y = z.value("y", 0.0);
        zone.w = z.value("w", 0.0);
        zone.h = z.value("h", 0.0);
        env.zones.push_back(zone);
    }
    for (const auto& o : j["obstacles"]) {
        Obstacle obs{};
        obs.w = o.value("w", 0.0);
        obs.h = o.value("h", 0.0);
        if (o.contains("cx")) {
            obs.cx = o.value("cx", 0.0);
            obs.cy = o.value("cy", 0.0);
        } else {
            double x = o.value("x", 0.0);
            double y = o.value("y", 0.0);
            obs.cx = x + obs.w / 2.0;
            obs.cy = y + obs.h / 2.0;
        }
        obs.angle = o.value("angle", 0.0);
        env.obstacles.push_back(obs);
    }
    for (const auto& t : j["targets"]) {
        Target target;
        target.name = t.value("name", "");
        target.pos.x = t.value("x", 0.0);
        target.pos.y = t.value("y", 0.0);
        env.targets.push_back(target);
    }
    if (j.contains("berths")) {
        std::string firstId;
        std::string lastId;
        for (const auto& b : j["berths"]) {
            Berth berth;
            berth.id = b.value("id", "");
            berth.quay = b.value("quay", "");
            berth.pos.x = b.value("x", 0.0);
            berth.pos.y = b.value("y", 0.0);
            berth.depth = b.value("depth_m", 0.0);
            berth.occupied = false;
            env.berths[berth.quay].push_back(berth);
            if (firstId.empty()) firstId = berth.id;
            lastId = berth.id;
        }
        if (!firstId.empty()) {
            std::cout << "Berths loaded: " << firstId << ".." << lastId << std::endl;
        }
    }
    if (j.contains("speed_profiles")) {
        const auto& sp = j["speed_profiles"];
        if (sp.contains("fallback") || sp.contains("override_by_zone")) {
            if (sp.contains("fallback")) {
                for (auto it = sp["fallback"].begin(); it != sp["fallback"].end(); ++it) {
                    env.speedProfiles[it.key()] = it.value().get<double>();
```

```
                }
            }
            if (sp.contains("override_by_zone")) {
                for (auto it = sp["override_by_zone"].begin(); it != sp["override_by_zone"].end();
                     ++it) {
                    env.speedProfiles[it.key()] = it.value().get<double>();
                }
            }
        } else {
            for (auto it = sp.begin(); it != sp.end(); ++it) {
                env.speedProfiles[it.key()] = it.value().get<double>();
            }
        }
    }
    if (env.targets.size() < static_cast<size_t>(NUM_FOUNDATIONS)) {
        auto it = std::find_if(env.zones.begin(), env.zones.end(), [](const Zone& z) {
            return z.type == "sea" || z.name == "Sea";
        });
        if (it != env.zones.end()) {
            size_t existing = env.targets.size();
            for (int i = existing; i < NUM_FOUNDATIONS; ++i) {
                double spacing = it->w / (NUM_FOUNDATIONS + 1);
                double x = it->x + spacing * (i + 1);
                double y = it->y + it->h / 2.0;
                Target gen;
                gen.name = "Site" + std::to_string(i + 1);
                gen.pos = {x, y};
                env.targets.push_back(gen);
            }
        }
    }
    return env;
}

std::vector<int> Environment::freeBerths(const std::string& quayName) const {
    std::vector<int> result;
    auto it = berths.find(quayName);
    if (it == berths.end()) return result;
    const auto& list = it->second;
    for (size_t i = 0; i < list.size(); ++i) {
        if (!list[i].occupied) result.push_back(static_cast<int>(i));
    }
    return result;
}

bool Environment::acquireBerth(const std::string& quayName, int berthIndex) {
    auto it = berths.find(quayName);
    if (it == berths.end()) return false;
    auto& list = it->second;
    if (berthIndex < 0 || berthIndex >= static_cast<int>(list.size())) return false;
    if (list[berthIndex].occupied) return false;
    list[berthIndex].occupied = true;
    return true;
}

void Environment::releaseBerth(const std::string& quayName, int berthIndex) {
    auto it = berths.find(quayName);
    if (it == berths.end()) return;
    auto& list = it->second;
```

```
    if (berthIndex < 0 || berthIndex >= static_cast<int>(list.size())) return;
    list[berthIndex].occupied = false;
}

int Environment::pickAnyFreeBerth(const std::string& quayName) const {
    auto it = berths.find(quayName);
    if (it == berths.end()) return -1;
    const auto& list = it->second;
    for (size_t i = 0; i < list.size(); ++i) {
        if (!list[i].occupied) return static_cast<int>(i);
    }
    return -1;
}

Vec2 Environment::zoneCenter(const std::string& name) const {
    for (const auto& z : zones) {
        if (z.name == name) {
            return Vec2{z.x + z.w / 2.0, z.y + z.h / 2.0};
        }
    }
    return Vec2{};
}

void Environment::handleMessage(const Message& msg) {
    if (msg.type == MessageType::ManufactureComplete) {
        int id = std::get<int>(msg.payload);
        dockInventory.push(id);
    }
}

void Environment::step(EventBus& bus,
                       std::vector<std::shared_ptr<Agent>>& agents,
                       std::vector<std::shared_ptr<FloatingFoundation>>& foundations,
                       std::shared_ptr<YardStorage> yardStorage,
                       AgentID dockId, AgentID towId, AgentID anchorId,
                       AgentID quayId, AgentID materialId) {
    if (!dockInventory.empty()) {
        dockInventory.pop();
        if (foundationsCreated + static_cast<int>(manufacturing.size()) < totalFoundations) {
            manufacturing.push_back(FOUNDATION_MAKE_DAYS * HOURS_PER_DAY);
        }
    }
    for (auto it = manufacturing.begin(); it != manufacturing.end();) {
        *it -= TIME_STEP_HOURS;
        if (*it <= 0) {
            if (foundationsCreated < totalFoundations) {
                Vec2 site = targets[foundationsCreated].pos;
                Bom bom{1, 1, 1};
                auto f = std::make_shared<FloatingFoundation>(nextFoundationId++, dockId, towId,
                    anchorId,
                                                              yardStorage->id(), quayId,
                                                                  materialId, site, bom);
                std::string dz = firstZoneNameOfType("dock");
                if (dz.empty()) dz = firstZoneNameOfType("basin");
                Vec2 dpos = zoneCenter(dz);
                f->setPosition(dpos);
                f->setEventBus(&bus);
                f->setHome(dpos, dz);
                f->setDestinationName(dz);
```

```
369                 f->setSpeed(BASE_SPEED);
370                 foundations.push_back(f);
371                 agents.push_back(f);
372                 bus.post({SYSTEM_ID, SYSTEM_ID, MessageType::ManufactureComplete, f->id()});
373                 ++foundationsCreated;
374             }
375             it = manufacturing.erase(it);
376         } else {
377             ++it;
378         }
379     }
380 }
```

1–12 行目　必要な標準・プロジェクトヘッダを読み込む.

14 行目　JSON パーサの型エイリアス.

16–20 行目　無名名前空間で矩形判定の補助関数を定義.

22–114 行目　ゾーン判定や速度取得など環境の基本機能.

116–172 行目　A*探索による経路計算を実装.

174–284 行目　JSON ファイルから環境情報を読み込み，ゾーンやターゲットを構築.

287–324 行目　バース管理の関数群.

326–333 行目　指定ゾーンの中心座標を計算.

335–340 行目　製作完了メッセージを受けて在庫キューを更新.

342 行目以降　製作中の基礎を進め，新しい基礎を生成するステップ処理.

■ ドッククラス ヘッダ（cpp/include/dock.h）

```
1  #pragma once
2  #include <queue>
3  #include <vector>
4  #include "agent.h"
5
6  // Dock agent managing fabrication slots and queueing.
7  // Responsibility moved from main.cpp: queuing foundations,
8  // tracking fabrication progress, broadcasting slot availability,
9  // and notifying foundations when fabrication completes.
10
11 class Dock : public Agent {
12     int capacity_;
13     int processDays_;
14     std::queue<AgentID> waiting_;           // foundations waiting for a slot
15     struct Task { AgentID id; int remainingHours; }; // remaining fabrication hours
16     std::vector<Task> working_;             // foundations currently in fabrication
17     bool broadcastedOpen_;                  // avoid spamming DockSlotOpen
18 public:
19     Dock(AgentID id, int capacity, int processDays);
20     void step(double dt) override;          // advance fabrication and broadcast
21     void receive(const Message& msg) override; // handle JoinQueue requests
22 };
```

1 行目　`#pragma once` により再インクルードを防止.

2–4 行目　標準ライブラリと基底クラスのヘッダをインクルード.

6–9 行目　ドックエージェントの責務をコメントで説明.

11 行目　‘Dock‘ クラスの宣言．‘Agent‘ を継承する.

12–17 行目　製作ライン数 ‘capacity_‘ や待機キュー ‘waiting_‘ など内部状態を定義.

19 行目　コンストラクタ宣言.

20 行目　シミュレーションを 1 ステップ進める ‘step‘ メソッド.

21 行目　メッセージ処理を行う ‘receive‘ メソッド.

■ ドッククラス 実装（cpp/src/dock.cpp）

```cpp
#include "dock.h"
#include "agent_logger.h"
#include "constants.h"
#include <sstream>

Dock::Dock(AgentID id, int capacity, int processDays)
    : Agent(id), capacity_(capacity), processDays_(processDays),
      broadcastedOpen_(false) {}

void Dock::receive(const Message& msg) {
    if (msg.type == MessageType::JoinQueue) {
        waiting_.push(msg.sender);
        LOG_AGENT("Dock", "receive: JoinQueue id=" << msg.sender
                  << " waiting=" << waiting_.size());
    }
}

void Dock::step(double dt) {
    moveAlongRoute(dt, *Agent::environment());
    checkZoneArrival("Dock");
    auto formatWorking = [&]() {
        std::ostringstream oss;
        oss << "[";
        bool first = true;
        for (const auto& t : working_) {
            if (!first) oss << ", ";
            oss << "{id:" << t.id << ", remaining_hours:" << t.remainingHours << "}";
            first = false;
        }
        oss << "]";
        return oss.str();
    };

    // move queued foundations into open slots
    while (!waiting_.empty() && static_cast<int>(working_.size()) < capacity_) {
        AgentID f = waiting_.front();
        waiting_.pop();
        working_.push_back({f, processDays_ * HOURS_PER_DAY});
        broadcastedOpen_ = false; // slot now filled
        LOG_AGENT("Dock", "start: id=" << f << " slots=" << working_.size()
                  << "/" << capacity_ << " working=" << formatWorking());
    }

    AgentID lastComplete = -1;
    // advance fabrication tasks
    for (auto it = working_.begin(); it != working_.end(); ) {
        it->remainingHours -= static_cast<int>(dt);
```

```
48          if (it->remainingHours <= 0) {
49              send(it->id, MessageType::BuildComplete);
50              lastComplete = it->id;
51              it = working_.erase(it);
52              broadcastedOpen_ = false; // freed slot -> allow next broadcast
53          } else {
54              ++it;
55          }
56      }
57
58      LOG_AGENT("Dock", "progress: waiting=" << waiting_.size()
59                << ", working=" << working_.size() << ' ' << formatWorking()
60                << ", last_complete=" << lastComplete);
61
62      // broadcast availability only once when a new slot becomes free
63      int openSlots = capacity_ - static_cast<int>(working_.size());
64      if (openSlots > 0 && !broadcastedOpen_) {
65          send(BROADCAST_ID, MessageType::DockSlotOpen);
66          broadcastedOpen_ = true;
67          LOG_AGENT("Dock", "broadcast: DockSlotOpen open_slots=" << openSlots);
68      }
69  }
```

1–4 行目　必要なヘッダをインクルード.

6–8 行目　コンストラクタで基底クラスとメンバを初期化し通知フラグをリセット.

10–16 行目　‘receive‘ は ‘JoinQueue‘ 要求を待機キューに追加しログ出力.

18–20 行目　ルート移動とゾーン到達判定を実行.

21–32 行目　ラムダ ‘formatWorking‘ で作業中タスクの表示文字列を生成.

34–42 行目　空きスロットがある限り待機キューからタスクを開始.

44–56 行目　作業残時間を減らし完了時は ‘BuildComplete‘ を送信.

58–60 行目　現在の待機・作業状況をログ出力.

62–68 行目　新たに空きスロットが出来たときだけ ‘DockSlotOpen‘ をブロードキャスト.

■ 牽引船隊 ヘッダ（cpp/include/tow_fleet.h）

```
1   #pragma once
2   #include <deque>
3   #include <vector>
4   #include <unordered_map>
5   #include <unordered_set>
6   #include <memory>
7   #include <string>
8   #include "agent.h"
9   #include "vec2.h"
10  #include "floating_foundation.h"
11  #include "constants.h"
12
13  class TowFleet : public Agent {
14      int capacity_;
15      struct TowReq { AgentID id; Vec2 target; int priority; };
16      struct ActiveTow {
17          AgentID id;
18          std::deque<Vec2> route;
```

```
        Vec2 target;
        int returnBuffer;
        std::string originZone;
        std::string destZone;
        double departRemaining;
        double arriveRemaining;
        bool waitingDestBerth;
        Vec2 tugPos;
    };
    struct Returning { SimTime remainingMin; };
    std::deque<TowReq> queue_;
    std::unordered_set<AgentID> queued_;
    std::vector<ActiveTow> active_;
    std::vector<Returning> returning_;
    std::unordered_map<AgentID, std::shared_ptr<FloatingFoundation>> foundations_;
    int dockOccupied_;
    int yardOccupied_;
    TowPriorityMode mode_;
public:
    TowFleet(AgentID id, int capacity);
    void registerFoundation(const std::shared_ptr<FloatingFoundation>& f);
    void step(double dt) override;
    void receive(const Message& msg) override;
};
```

1 行目　`#pragma once` で多重定義を防止.

2–11 行目　デックやコンテナ操作，浮体基礎参照のためのヘッダをインクルード.

13 行目　Agent を継承する曳航船隊クラスの宣言.

14–27 行目　処理能力と曳航要求・進行中曳航を管理する内部構造体.

28–36 行目　待機キューや戻り航行などの内部状態と関連マップ.

38–41 行目　コンストラクタ，基礎登録，ステップ更新，メッセージ受信関数.

■ 牽引船隊 実装（cpp/src/tow_fleet.cpp）

```
#include "tow_fleet.h"
#include "agent_logger.h"
#include <algorithm>
#include <sstream>
#include <cmath>
#include "environment.h"

TowFleet::TowFleet(AgentID id, int capacity)
    : Agent(id), capacity_(capacity), dockOccupied_(0), yardOccupied_(0),
      mode_(TOW_PRIORITY_MODE) {}

void TowFleet::registerFoundation(const std::shared_ptr<FloatingFoundation>& f) {
    foundations_[f->id()] = f;
}

void TowFleet::receive(const Message& msg) {
    if (msg.type == MessageType::TowRequest) {
        Vec2 tgt = std::get<Vec2>(msg.payload);
        bool active = std::any_of(active_.begin(), active_.end(), [&](const ActiveTow& t) {
            return t.id == msg.sender;
```

```
        });
        if (active || queued_.count(msg.sender)) {
            LOG_AGENT("TowFleet", "TowFleet ignoring duplicate TowRequest from " << msg.sender);
            return;
        }
        int priority = -static_cast<int>(msg.sender);
        queue_.push_back({msg.sender, tgt, priority});
        queued_.insert(msg.sender);
        std::ostringstream ids;
        for (size_t i = 0; i < queue_.size(); ++i) {
            if (i) ids << ',';
            ids << queue_[i].id;
        }
        LOG_AGENT("TowFleet", "TowFleet received TowRequest from " << msg.sender
                  << " to (" << tgt.x << ", " << tgt.y << ") queue="
                  << queue_.size() << " order=[" << ids.str() << "]");
    }
}

void TowFleet::step(double dt) {
    std::ostringstream buf;
    for (size_t i = 0; i < returning_.size(); ++i) {
        if (i) buf << ',';
        buf << returning_[i].remainingMin / 60 << "h (" << returning_[i].remainingMin / (
            HOURS_PER_DAY * 60) << "d)";
    }
    std::ostringstream qids;
    for (size_t i = 0; i < queue_.size(); ++i) {
        if (i) qids << ',';
        qids << queue_[i].id;
    }
    LOG_AGENT("TowFleet", "TowFleet step: queue=" << queue_.size()
              << " active=" << active_.size()
              << " returning=" << returning_.size()
              << " remaining=[" << buf.str() << "]"
              << " q=[" << qids.str() << "]"
              << " capacity=" << capacity_
              << " dock=" << dockOccupied_ << "/" << DOCK_BERTHS
              << " yard=" << yardOccupied_ << "/" << YARD_BERTHS);
    for (auto it = returning_.begin(); it != returning_.end();) {
        it->remainingMin -= hours_to_min(dt);
        if (it->remainingMin <= 0) {
            it = returning_.erase(it);
        } else {
            ++it;
        }
    }

    size_t used = active_.size() + returning_.size();
    auto env = Agent::environment();
    auto computeTime = [&](const std::deque<Vec2>& path) {
        double t = 0.0;
        for (size_t i = 1; i < path.size(); ++i) {
            Vec2 a = path[i-1];
            Vec2 b = path[i];
            double len = (b - a).length();
            std::string zone = env->zoneName(a.x, a.y);
            double sp = env->speedForZone(zone); // zone-dependent speed
            t += len / sp;
```

```
        }
        return t;
    };

    while (used < static_cast<size_t>(capacity_) && !queue_.empty()) {
        TowReq req;
        if (mode_ == TowPriorityMode::FIFO) {
            req = queue_.front();
            queue_.pop_front();
        } else {
            auto best = std::max_element(queue_.begin(), queue_.end(),
                                         [](const TowReq& a, const TowReq& b){ return a.priority <
                                              b.priority; });
            req = *best;
            queue_.erase(best);
        }
        auto it = foundations_.find(req.id);
        if (it == foundations_.end()) continue;
        auto f = it->second;
        std::string requestedZone = env->zoneName(req.target.x, req.target.y);
        std::string navZone = requestedZone;
        Vec2 navTarget = env->snapToWaterTarget(req.target, &navZone);
        std::deque<Vec2> path = env->findPath(f->position(), navTarget);
        if (path.empty()) {
            LOG_AGENT("TowFleet", "no path: rawZone=" << requestedZone << " navZone=" << navZone
                      << " from=(" << f->position().x << "," << f->position().y
                      << ") to=(" << navTarget.x << "," << navTarget.y << ")");
            queue_.push_back(req);
            break; // avoid tight loop on unreached targets
        }
        std::string originZone = env->zoneName(f->position().x, f->position().y);
        std::string originType = env->zoneType(originZone);
        bool berthFree = true;
        if (originType == "dock" && dockOccupied_ >= DOCK_BERTHS) berthFree = false;
        if (originType == "yard" && yardOccupied_ >= YARD_BERTHS) berthFree = false;
        if (!berthFree) {
            queue_.push_back(req);
            break;
        }
        if (originType == "dock") ++dockOccupied_;
        else if (originType == "yard") ++yardOccupied_;
        double routeTime = computeTime(path);
        int buffer = static_cast<int>(std::ceil(routeTime + 2 * MOORING_HOURS));
        f->setRoute(path);
        f->setDestination(navTarget);
        f->setDestinationName(!requestedZone.empty() ? requestedZone : navZone);
        send(req.id, MessageType::TowStart);
        active_.push_back({req.id, path, navTarget, buffer, originZone, navZone, MOORING_HOURS,
            0.0, false, f->position()});
        queued_.erase(req.id);
        std::ostringstream leftIds;
        for (size_t i = 0; i < queue_.size(); ++i) {
            if (i) leftIds << ',';
            leftIds << queue_[i].id;
        }
        LOG_AGENT("TowFleet", "TowFleet starting tow for foundation " << req.id
                  << " from " << originZone << " to zone " << navZone
                  << " (buffer " << buffer << "h), remaining capacity "
                  << (capacity_ - static_cast<int>(active_.size())
```

```
                    - static_cast<int>(returning_.size()))
                << ", queue left " << queue_.size() << " [" << leftIds.str() << "]");
        ++used;
    }
    if (used >= static_cast<size_t>(capacity_) && !queue_.empty()) {
        std::ostringstream leftIds;
        for (size_t i = 0; i < queue_.size(); ++i) {
            if (i) leftIds << ',';
            leftIds << queue_[i].id;
        }
        LOG_AGENT("TowFleet", "TowFleet at capacity, remaining queue " << queue_.size()
                << " [" << leftIds.str() << "]");
    }

    for (auto it = active_.begin(); it != active_.end();) {
        auto f = foundations_[it->id];
        if (it->departRemaining > 0.0) {
            it->departRemaining -= dt;
            f->setVelocity({0.0, 0.0});
            f->setStatus("departing");
            if (it->departRemaining <= 0.0) {
                std::string oType = env->zoneType(it->originZone);
                if (oType == "dock") --dockOccupied_;
                else if (oType == "yard") --yardOccupied_;
            }
            it->tugPos = f->position();
            ++it;
            continue;
        }
        if (!it->route.empty()) {
            Vec2 pos = f->position();
            Vec2 target = it->route.front();
            Vec2 toTarget = target - pos;
            double dist = toTarget.length();
            std::string zone = env->zoneName(pos.x, pos.y);
            double sp = env->speedForZone(zone);
            double stepDist = sp * dt;
            if (dist <= stepDist) {
                Vec2 next = target;
                Vec2 vel = (next - pos) * (1.0 / dt);
                f->setVelocity(vel);
                f->setPosition(next);
                f->setStatus("moving");
                it->route.pop_front();
                LOG_AGENT("TowFleet", "snap_to_waypoint id=" << it->id
                          << " d=" << dist << " step=" << stepDist);
                it->tugPos = f->position();
                ++it;
                continue;
            }
            Vec2 dir = toTarget.normalized();
            Vec2 next = pos + dir * stepDist;
            if (env->isOccupied(next.x, next.y)) {
                Vec2 perp(-dir.y, dir.x);
                Vec2 alt = pos + perp * stepDist;
                if (!env->isOccupied(alt.x, alt.y)) {
                    next = alt;
                } else {
                    alt = pos - perp * stepDist;
```

```
                    if (!env->isOccupied(alt.x, alt.y)) {
                        next = alt;
                    } else {
                        f->setVelocity({0.0, 0.0});
                        f->setStatus("blocked");
                        LOG_AGENT("TowFleet", "TowFleet foundation " << it->id
                                  << " blocked at (" << pos.x << "," << pos.y << ")");
                        ++it;
                        continue;
                    }
                }
            }
            Vec2 vel = (next - pos) * (1.0 / dt);
            f->setVelocity(vel);
            f->setPosition(next);
            f->setStatus("moving");
            it->tugPos = next;
            ++it;
        } else {
            it->arriveRemaining -= dt;
            if (it->arriveRemaining <= 0.0) {
                send(it->id, MessageType::TowComplete);
                returning_.push_back({hours_to_min(it->returnBuffer)});
                if (env->zoneType(it->originZone) == "dock") {
                    send(it->id, MessageType::LeaveDock);
                }
                it = active_.erase(it);
            } else {
                ++it;
            }
        }
    }
}
```

1–6 行目　必要なヘッダを読み込む.

8–10 行目　コンストラクタで能力と占有状況を初期化.

12–14 行目　`registerFoundation` では曳航艦隊が担当する浮体基礎を管理リスト `foundations_`に登録し，ID をキーに共有ポインタを格納することで，後続の曳航要求メッセージに即応できる状態を整える．登録済みか否かの判別が高速に行えるため，重複要求の排除や進行中曳航との照合が容易となる．ここで ID と基礎ポインタの対応をハッシュマップで保持することにより，各メッセージ処理時に線形探索を行う必要がなく，艦隊の状態保持とリソース管理が一貫する．処理内容は以下の擬似コードに集約される．

```
foundations_.emplace(f->id(), f);
```

16–37 行目　`receive` は曳航要求をキューに登録し重複を除外.

40–58 行目　`step` 冒頭で戻航や待機の状況をログ出力.

59–66 行目　戻航中タスクの残時間を更新.

68–139 行目　空き能力があればキューから曳航を開始し経路と所要時間を計算.

140–148 行目　能力不足時は残キューをログに記録.

150 行目以降　進行中曳航を前進させ位置と状態を更新し，完了時には ‘TowComplete‘ を送信.

■ 浮体基礎クラス ヘッダ（cpp/include/floating_foundation.h）

```
#pragma once
#include <deque>
#include "agent.h"

// Floating foundation agent that now manages its own queueing and towing.
// Responsibility moved from main.cpp: joining dock queues, requesting tows,
// and tracking tow progress until completion.

// Stages for the simplified foundation workflow.
enum class FoundationStage {
    None,
    Fabrication,
    DockTow,
    YardStorage,
    Assembly,
    SiteTow,
    Anchoring,
    Anchored,
    Completed
};

class FloatingFoundation : public Agent {
    FoundationStage stage_;
    int remainingDays_;
    AgentID dock_;
    AgentID tow_;
    AgentID anchor_;
    AgentID yardStorage_;
    AgentID quay_;
    AgentID material_;
    Bom bom_;
    Vec2 siteTarget_;
    bool started_;
    bool yardSlot_; // track yard storage assignment
    bool waitingStorageSlot_;
    bool waitingTowStart_;
    bool waitingTowComplete_;
public:
    FloatingFoundation(int id, AgentID dock, AgentID tow, AgentID anchor,
                       AgentID yardStorage, AgentID quay,
                       AgentID material, const Vec2& siteTarget,
                       Bom bom);
    void step(double dt) override;      // send next request based on current stage
    void receive(const Message& msg) override; // react to completion messages

    // legacy helpers kept for compatibility
    void setStage(FoundationStage s, int duration);
    void progressOneDay();
    FoundationStage stage() const { return stage_; }
    bool isCompleted() const { return stage_ == FoundationStage::Completed; }
};
```

1 行目　#pragma once により再インクルードを防止.

2–3 行目　必要な標準ヘッダと基底クラスを読み込む.

5–7 行目　クラスの責務をコメントで説明.

9–20 行目　基礎の工程を表す ‘FoundationStage‘ 列挙体を定義.

22–37 行目　‘FloatingFoundation‘ クラスのメンバ変数．関連エージェント ID や状態管理フラグを保持.

38–44 行目　コンストラクタと仮想関数 ‘step‘・‘receive‘ を宣言.

47–52 行目　過去互換用の補助関数とアクセス関数.

■ 浮体基礎クラス 実装（cpp/src/floating_foundation.cpp）

```cpp
#include "floating_foundation.h"
#include "agent_logger.h"
#include "environment.h"

namespace {
    const char* stageName(FoundationStage s) {
        switch (s) {
        case FoundationStage::None: return "None";
        case FoundationStage::Fabrication: return "Fabrication";
        case FoundationStage::DockTow: return "DockTow";
        case FoundationStage::YardStorage: return "YardStorage";
        case FoundationStage::Assembly: return "Assembly";
        case FoundationStage::SiteTow: return "SiteTow";
        case FoundationStage::Anchoring: return "Anchoring";
        case FoundationStage::Anchored: return "Anchored";
        case FoundationStage::Completed: return "Completed";
        }
        return "Unknown";
    }
}

FloatingFoundation::FloatingFoundation(int id, AgentID dock, AgentID tow, AgentID anchor,
                                       AgentID yardStorage, AgentID quay,
                                       AgentID material, const Vec2& siteTarget,
                                       Bom bom)
    : Agent(id), stage_(FoundationStage::None), remainingDays_(0),
      dock_(dock), tow_(tow), anchor_(anchor), yardStorage_(yardStorage),
      quay_(quay), material_(material), bom_(bom), siteTarget_(siteTarget),
      started_(false), yardSlot_(false),
      waitingStorageSlot_(false), waitingTowStart_(false), waitingTowComplete_(false),
      preAnchored_(false), anchorTested_(false), assemblyDone_(false),
      waitingAssemblyMaterials_(false) {}

void FloatingFoundation::receive(const Message& msg) {
    if (stage_ == FoundationStage::Completed) return;
    LOG_AGENT("FloatingFoundation", "Foundation " << id() << " received message " << static_cast<
        int>(msg.type)
              << " at stage " << stageName(stage_) << " from " << msg.sender);
    switch (msg.type) {
    case MessageType::MaterialReady:
        LOG_AGENT("FloatingFoundation", "Foundation " << id() << " materials ready at stage "
                  << stageName(stage_));
        if (stage_ == FoundationStage::AssemblyPending) {
            waitingAssemblyMaterials_ = false;
            send(yardStorage_, MessageType::LeaveStorage);
            LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent LeaveStorage to yard")
                ;
            yardSlot_ = false;
```

```
            send(quay_, MessageType::AssemblyRequest);
            LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent AssemblyRequest to quay");
            setStage(FoundationStage::Assembly, 0);
        } else {
            LOG_AGENT("FloatingFoundation", "WARNING: Foundation " << id()
                      << " received MaterialReady in unexpected stage " << stageName(stage_));
        }
        break;
    case MessageType::BuildComplete: {
        LOG_AGENT("FloatingFoundation", "Foundation " << id() << " build complete");
        Vec2 yard = Agent::environment()->zoneCenter("Yard");
        send(tow_, MessageType::TowRequest, yard);
        LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent TowRequest to yard");
        waitingTowStart_ = true;
        setStage(FoundationStage::DockTow, 0);
        setStatus("idle");
        break;
    }
    case MessageType::StorageSlotAssigned:
        if (msg.sender == yardStorage_) {
            LOG_AGENT("FloatingFoundation", "Foundation " << id() << " yard storage slot assigned at stage "
                      << stageName(stage_));
            if (!waitingStorageSlot_) {
                LOG_AGENT("FloatingFoundation", "WARNING: Foundation " << id() << " received StorageSlotAssigned without pending request");
            }
            waitingStorageSlot_ = false;
            if (stage_ == FoundationStage::YardStorage) {
                if (!waitingAssemblyMaterials_) {
                    send(material_, MessageType::MaterialRequest, bom_);
                    LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent MaterialRequest for assembly");
                    waitingAssemblyMaterials_ = true;
                    setStage(FoundationStage::AssemblyPending, 0);
                } else {
                    LOG_AGENT("FloatingFoundation", "WARNING: Foundation " << id()
                              << " already waiting for assembly materials");
                }
            } else {
                send(yardStorage_, MessageType::LeaveStorage);
                LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent LeaveStorage to yard");
                yardSlot_ = false;
                send(dock_, MessageType::JoinQueue);
                LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent JoinQueue to dock");
                setStage(FoundationStage::Fabrication, 0);
                started_ = true;
            }
        }
        break;
    case MessageType::AssemblyComplete:
        LOG_AGENT("FloatingFoundation", "Foundation " << id() << " assembly complete");
        send(tow_, MessageType::TowRequest, siteTarget_);
        LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent TowRequest to site");
        waitingTowStart_ = true;
        setStage(FoundationStage::SiteTow, 0);
```

```
100         setStatus("idle");
101         break;
102     case MessageType::TowStart:
103         LOG_AGENT("FloatingFoundation", "Foundation " << id() << " tow started");
104         if (!waitingTowStart_) {
105             LOG_AGENT("FloatingFoundation", "WARNING: Foundation " << id() << " TowStart
                    unexpected");
106         }
107         waitingTowStart_ = false;
108         waitingTowComplete_ = true;
109         setStatus("moving");
110         break;
111     case MessageType::TowComplete:
112         if (!waitingTowComplete_) {
113             LOG_AGENT("FloatingFoundation", "WARNING: Foundation " << id() << " TowComplete
                    unexpected");
114         }
115         waitingTowComplete_ = false;
116         if (stage_ == FoundationStage::DockTow) {
117             setStatus("idle");
118             send(yardStorage_, MessageType::StorageRequest);
119             LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent StorageRequest after
                    DockTow");
120             yardSlot_ = true;
121             waitingStorageSlot_ = true;
122             waitingAssemblyMaterials_ = false;
123             setStage(FoundationStage::YardStorage, 0);
124             LOG_AGENT("FloatingFoundation", "Foundation " << id() << " completed dock tow,
                    requesting yard storage");
125         } else if (stage_ == FoundationStage::SiteTow) {
126             setStatus("idle");
127             send(anchor_, MessageType::AnchorRequest);
128             LOG_AGENT("FloatingFoundation", "Foundation " << id() << " sent AnchorRequest");
129             setStage(FoundationStage::Anchoring, 0);
130             LOG_AGENT("FloatingFoundation", "Foundation " << id() << " arrived at site, requesting
                     anchor");
131         }
132         break;
133     case MessageType::AnchorComplete:
134         LOG_AGENT("FloatingFoundation", "Foundation " << id() << " anchor complete");
135         setStatus("idle");
136         setStage(FoundationStage::Completed, 0);
137         break;
138     default:
139         break;
140     }
141 }
142 void FloatingFoundation::step(double dt) {
143     (void)dt;
144     if (stage_ == FoundationStage::Completed) return;
145     LOG_AGENT("FloatingFoundation", "Foundation " << id() << " stepping at stage " << stageName(
            stage_)
146               << ", yardSlot=" << yardSlot_ << " status=" << status()
147               << " waitingStorageSlot=" << waitingStorageSlot_
148               << " waitingTowStart=" << waitingTowStart_
149               << " waitingTowComplete=" << waitingTowComplete_);
150     if (!yardSlot_ && stage_ == FoundationStage::None && !waitingStorageSlot_) {
151         LOG_AGENT("FloatingFoundation", "Foundation " << id() << " requesting yard storage");
152         send(yardStorage_, MessageType::StorageRequest);
```

```
153         waitingStorageSlot_ = true;
154         yardSlot_ = true;
155     } else if (!yardSlot_ && stage_ == FoundationStage::None && waitingStorageSlot_) {
156         LOG_AGENT("FloatingFoundation", "Foundation " << id() << " WARNING: waiting for yard
                storage assignment");
157     }
158 }
159
160 void FloatingFoundation::setStage(FoundationStage s, int duration) {
161     LOG_AGENT("FloatingFoundation", "Foundation " << id() << " stage change " << stageName(stage_)
162               << " -> " << stageName(s));
163     stage_ = s;
164     remainingDays_ = duration;
165 }
166
167 void FloatingFoundation::progressOneDay() {
168     if (remainingDays_ > 0) --remainingDays_;
169 }
```

1–3 行目　ヘッダと環境クラスをインクルード.

5–20 行目　無名名前空間で列挙値を文字列に変換する補助関数を定義.

22–30 行目　コンストラクタで関連エージェント ID や状態フラグを初期化.

33–124 行目　‘receive‘ 関数で各種メッセージに応じて次工程を指示.

126–142 行目　‘step‘ は必要に応じて保管要求を送るなど工程を進める.

144–149 行目　‘setStage‘ で工程と残日数を更新.

151–152 行目　‘progressOneDay‘ は残日数を 1 日減少させる補助関数.

■ 係留フリート ヘッダ（cpp/include/mooring_fleet.h）

```
1  #pragma once
2  #include <queue>
3  #include <unordered_map>
4  #include <vector>
5  #include "agent.h"
6  #include "constants.h"
7
8  class MooringFleet : public Agent {
9      int capacity_;
10     int linesPerFoundation_;
11     int daysPerLine_;
12     struct Task { AgentID id; SimTime remainingMin; };
13     std::queue<AgentID> mooringQueue_;
14     std::vector<Task> working_;
15     std::unordered_map<AgentID, int> remainingLines_;
16 public:
17     MooringFleet(AgentID id, int capacity, int linesPerFoundation, int daysPerLine);
18     void step(double dt) override;
19     void receive(const Message& msg) override;
20 };
```

1 行目　`#pragma once` で多重定義を防止.

2–6 行目　キューや連想配列（好きな添字・キーをつけることのできる配列のこと）など必要なヘッダを読み込む.

8–14 行目　‘MooringFleet‘ の内部状態．係留待ちキュー ‘mooringQueue_‘ と進行中タスク ‘working_‘ を保持する．
16–18 行目　コンストラクタと仮想関数 ‘step‘・‘receive‘ の宣言．

■ 係留フリート 実装（cpp/src/mooring_fleet.cpp）

```cpp
#include "mooring_fleet.h"
#include "agent_logger.h"
#include "constants.h"

MooringFleet::MooringFleet(AgentID id, int capacity, int linesPerFoundation, int daysPerLine)
    : Agent(id), capacity_(capacity), linesPerFoundation_(linesPerFoundation), daysPerLine_(
        daysPerLine) {}

void MooringFleet::receive(const Message& msg) {
    if (msg.type == MessageType::MooringRequest) {
        LOG_AGENT("MooringFleet", "received MooringRequest from " << msg.sender
                  << ", queue size " << mooringQueue_.size() + 1);
        mooringQueue_.push(msg.sender);
        remainingLines_[msg.sender] = linesPerFoundation_;
    }
}

void MooringFleet::step(double dt) {
    LOG_AGENT("MooringFleet", "step: queue=" << mooringQueue_.size()
              << " working=" << working_.size());
    size_t used = working_.size();
    while (used < static_cast<size_t>(capacity_) && !mooringQueue_.empty()) {
        AgentID id = mooringQueue_.front();
        mooringQueue_.pop();
        LOG_AGENT("MooringFleet", "starting mooring line for " << id);
        working_.push_back({id, hours_to_min(daysPerLine_ * HOURS_PER_DAY)});
        --remainingLines_[id];
        if (remainingLines_[id] > 0) {
            mooringQueue_.push(id);
        }
        ++used;
    }
    for (auto it = working_.begin(); it != working_.end();) {
        it->remainingMin -= hours_to_min(dt);
        if (it->remainingMin <= 0) {
            AgentID fid = it->id;
            LOG_AGENT("MooringFleet", "completed mooring line for " << fid);
            it = working_.erase(it);
            bool stillWorking = false;
            for (const auto& w : working_) {
                if (w.id == fid) { stillWorking = true; break; }
            }
            if (!stillWorking && remainingLines_[fid] == 0) {
                send(fid, MessageType::InstallComplete);
                send(CABLE_LAY_FLEET_ID, MessageType::InstallComplete, fid);
                remainingLines_.erase(fid);
            }
        } else {
            ++it;
        }
    }
}
```

1–3 行目　必要なヘッダをインクルード.

5–6 行目　コンストラクタで処理能力や工程時間を設定.

8–15 行目　‘receive‘ は ‘MooringRequest‘ をキューに登録し残り係留本数を記録する.

17–38 行目　‘step‘ はキューからタスクを割り当て進捗を更新し，全本数完了時に ‘InstallComplete‘ を送信する.

■ プレアンカーフリート（cpp/pre_anchor_fleet.*）

役割　風車到着前にアンカーを敷設し, 把駐力試験まで担当する艦隊. ‘PreAnchorRequest‘ を受信するとサイト毎に複数本のアンカー設置タスクを順次処理し, 完了後に ‘PreAnchorComplete‘ を通知する. 続いて ‘AnchorTestRequest‘ を受けると ‘ANCHOR_TEST_DAYS‘ だけ試験を行い, ‘AnchorTestComplete‘ を返して係留準備完了を知らせる.

読む順番

本章のコードを追う際は次のヘッダを順に確認すると理解しやすい.

1. `message.h`
2. `event_bus.h`
3. `agent.h`
4. 各エージェント固有のヘッダ（`dock.h` や `tow_fleet.h` など）

5　Python 可視化

シミュレーションで生成された logs/run.log と logs/agent_log.csv を python/ ディレクトリ内の各種スクリプトで解析する.

- **python/animate_agents.py**：ログからエージェント軌跡をアニメーション化.
- **python/visualize_logs.py**：静的軌跡図や動画を生成.
- **python/gantt_chart.py**：浮体基礎の工程をガントチャート化.
- **python/equipment_usage_visualize.py**：ドックやクレーン稼働率を時系列表示.
- **python/stock_level_visualize.py**：部材在庫と欠品時刻を可視化.

利用例：

```
$ python python/animate_agents.py --csv logs/samples.csv \
    --map data/maps/port_map_detailed.json --save logs/anim.gif

$ python python/gantt_chart.py --csv logs/samples.csv \
    --out logs/gantt_chart.png
```

6　実行方法

6.1　ビルド

```
$ cd cpp
$ make
```

main.exe が生成される．

6.2　シミュレーションの起動

```
$ ./main.exe
```

起動直後にマップ JSON を読み込み，MapValidator が必須項目の有無や型の不整合を検査する。致命的なエラーが検出された場合は色付きのレポートを出力したうえでシミュレーションを中止するが，berths や water_types が欠落しているなど後方互換性のために既定値へフォールバックできる項目については警告（Warning）を表示したまま続行する。検査結果は標準出力に表示され，10 秒間は画面を保持して内容を確認でき，途中で Enter または s キーを押すと即座に先へ進める。[*6] ../logs/run.log と ../logs/agent_log.csv が出力される．

6.3　可視化

出力されたログを前述の Python スクリプトで解析・図化する．結果は logs/ 以下に PNG/GIF などで保存される．

[*6] cpp/src/map_validator.cpp, cpp/main.cpp

7　今後の拡張予定

本シミュレータは，エージェント間のメッセージ駆動と EventBus による時刻管理を備え，港湾における洋上風車施工プロセスを再現している．今後は，**複雑系としての挙動と不確実性の連関**を解析できる研究基盤へ発展させるため，以下の拡張を段階的に進める．

7.1　研究課題に直結する機能拡張（複雑系・不確実性）

- **工程遅延と適応ロジック**: 資材搬入遅延，作業員／船舶割当の競合，気象による作業停止などを事象として導入し，各エージェントが待機・再スケジューリング・別作業への振替を**動的に**判断する方策（優先度制御，ルールベース，簡易強化学習の policy 差替え）を比較可能にする．
- **仮置き場配置のシナリオ分析**: 港内・湾内・外洋に複数の仮置き候補を用意し，距離・波浪暴露・錨地制約をパラメタライズした**ロケーション・ネットワーク**上で輸送・待機時間を再現．配置選択の違いが *makespan* や**待ち行列長**に与える影響を比較する．
- **確率的要素とモンテカルロ解析**: 曳航速度の揺らぎ（波浪・潮流による確率分布），悪天候の作業ウィンドウ，設備故障確率を導入．多数試行から工期・コスト・稼働率の分布と 95% 信頼区間を推計し，遅延要因の寄与（後述の感度分析）を可視化する．
- **パラメータ可変化と感度分析**: constants.h の固定値を外部設定（YAML/JSON）へ移行し，**一括シナリオ実行**で全要素の**ローカル感度**（% 変化に対する KPI 変動）や**分散ベース感度**（Sobol 指標の算出）を支援する．
- **外部データとキャリブレーション**: AIS 等の近似データや公開報告書のサイクルタイムを用いて，速度・待機・気象停止モデルを**ベイズ推定**／**最小二乗**で粗校正．将来の実施工データと突合できるよう，同一フォーマットのログ／ CSV を出力する．

7.2　アーキテクチャ拡張（MAS らしさとスケーラビリティ）

- **Coordinator エージェント**: 現在 main.cpp が担う集中管理を，専任の Coordinator に委譲．開始・中断・再開・優先度変更等を**メッセージ**で行い，分散的に進行管理できるようにする．
- **InformationAgent（環境）**： 天候・海象・港湾運用情報を定期配信する InformationAgent を新設．TowFleet/MooringFleet は**受信情報に基づく可否判定**と**作業ウィンドウ**の再計画を行う．
- **AnalysisAgent（集計・共有）**：各エージェントの完了イベントをサブスクライブし，待機時間，キュー長，稼働率，同時使用率など KPI をオンラインで集計．閾値超過時に Coordinator へ**ボトルネック警告**を通知．
- **MonitoringAgent（異常検知）**：進捗逸脱，長期アイドル，デッドロックの兆候を監視し，回復アクション（優先度付け替え，迂回，再割当）を提案する．
- **NegotiationAgent（交渉）**：曳航要求競合などのリソース割付を，**入札**／**オークション**／ *Contract-Net* 等の簡易プロトコルで決定．ルールの交換容易性（policy 差替え）を確保．
- **多港湾・多艦隊シナリオ**: 複数港／複数船隊／複数案件の並行施工を想定．資材，クレーン，曳航・錨泊船の**グローバル最適割当（ヒューリスティクス**／ *MIP* **近似）**を，シミュレーション・イン・ザ・ループで評価．

7.3　計測・可視化（ボトルネック特定のための計装）

- **KPI 定義**: makespan，各資源の稼働率（busy/available），平均／最大待ち行列長，タスク遅れ（tardiness），航走距離・時間，気象停止時間，港内滞留時間，**クリティカルパス頻度**．

- **ログ拡充**: すべての `TaskStart/End`，`QueueEnter/Leave`，`ResourceAcquire/Release` を CSV 出力（`time`，`agent`，`event`，`resource`，`id`，`payload`）．乱数シード，設定ファイル，git コミット ID を**ラン** *ID* に紐づけ保存．
- **自動プロット**: 実行後にガントチャート，稼働率ヒートマップ，キュー長の時系列，Sankey 的フロー図を生成．上位 N ボトルネックを**寄与度**とともに報告．

7.4　再現性・共同利用性

- **設定の外部化と署名**: `YAML/JSON` で全シナリオを管理し，実行ごとに設定のハッシュと git コミット ID を結果に埋め込む．乱数シードを明示管理．
- **一括実験ランナー**: `--sweep file.yaml` でパラメータ探索（格子・LHS・ベイズ最適化）を実行，CSV を自動集約．
- **Python 連携**: Pybind11 で `run(config)` を公開．Jupyter で統計解析／可視化，Monte Carlo の並列実行を容易化．
- **配布形態**: Docker/conda 環境を用意し，第三者がクローン即実行できるテンプレートを提供（CI で検証）．

7.5　実装イメージ（抜粋）

■ 外部設定の例

```
{
"seed": 42,
"weather": {"pause_prob": 0.08, "window_hours": [8, 18]},
"tow": {"speed_mean": 6.0, "speed_sigma": 1.2},
"laydown": [{"name": "HarborA", "pos":[0,0]}, {"name":"BayX","pos":[12,7]}],
"policy": {"dispatch":"priority", "tie_break":"shortest-travel"}
}
```

■ Monte Carlo ドライバ（擬似コード）

```
for (int i = 0; i < trials; ++i) {
auto cfg = load_config(base_cfg);
cfg.seed = base_seed + i;
auto sim = Simulator(cfg);
sim.run();
kpi_logger.flush(i, cfg, sim.metrics()); // makespan, utilizations, queues...
}
```

7.6　評価設計（論文化の観点）

- **研究仮説**:
  1. 適応ロジック導入により，搬入遅延があっても makespan の分散が減少する．
  2. 仮置き場の地理的選択は，待機時間と曳航負荷のトレードオフをもたらす（最適点が存在）．
- **実験計画**: ベースライン（現行ロジック）vs. 提案（適応・交渉・配置最適化）を同一乱数系列で比較．効果量と信頼区間を報告．
- **感度・寄与分析**: 主要パラメータの Sobol 指標／部分依存プロットで KPI への寄与を可視化．どの不確実性が支

配的かを明示.

- **再現性記述**: 設定ファイル・シード・コミット ID・Docker タグを付記し，公開可能な近似データで**再現可能パッケージ**を添付.

7.7 導入順のロードマップ

1. **計装と外部設定**: ログ拡充，KPI 計測，設定外部化，シード管理.
2. **不確実性の導入**: 速度揺らぎ，気象停止，工程遅延の確率モデル（最初は簡素な分布）.
3. **適応・交渉ロジック**: `Coordinator`/`Information`/`Analysis`/`Negotiation` の最小実装.
4. **仮置き場シナリオと一括実験**: 複数配置案の比較，Monte Carlo 運用.
5. **Python 連携と可視化**: Pybind11，Jupyter テンプレ，ボトルネック自動レポート.

7.8 期待される学術的貢献

- **MAS の適合性の検証**: 分散協調により，局所遅延が全体へ波及する過程や緩和戦略の有効性をシミュレーションで示す.
- **不確実性下の計画頑健化**: モンテカルロ＋感度分析により，計画の脆弱点と対策（配置・優先度・交渉）の効果を定量化.
- **ボトルネックの一般則**: 異なる港・配置でも再現される**反復的ボトルネック**のパターン（例：クレーンのウィンドウ制約）を抽出.

図 18　将来追加予定の可視化例（ダミー）

付録 A　C++ 入門とコード基礎

A.1　C++ 超入門

C++ は部品を組み合わせて機械を作るような言語である．最初に書く `#include` は「工具箱から必要な道具を持ってくる」合図に相当し，他の人が作った便利な部品を利用できるようにする．例えば `#include <iostream>` と書くと，標準入出力という道具一式が手元に届く．次に現れる `int main()` はプログラムの玄関であり，ここから処理が始まる．玄関で靴を履きかえるように，`main` の中で準備を整え，最後に `return 0;` と書くことで「無事に作業を終えました」と伝える仕組みだ．`main` の外に書いた関数は，工場内に置かれた専用の作業機械に似ており，必要な時だけ呼び出して仕事を任せられる．

関数は「入力を受け取り，結果を返す小さな箱」と考えると分かりやすい．ボタンを押すと決められた動作を行う自販機のようなもので，何度でも再利用できる．これをいくつも組み合わせることで，長い計算や複雑な処理を手際よく進められる．

クラスは設計図に該当する．船を建造する際に同じ図面から複数の船を作るように，クラスからは同じ構造を持つオブジェクトを何隻でも生み出せる．クラス内部にまとめておくことで，船体の状態と操縦方法を一つに管理でき，保守が楽になる．

名前空間 `namespace` は倉庫の区画番号のようなものだ．荷物置き場で同じ名前の箱が並ぶと混乱するため，`std` という札を付けた棚，`project` という札を付けた棚と分けて管理する．`std::cout` の `std` は「標準棚の中の `cout` を使う」という指示である．

上記の概念をまとめた最小の例を以下に示す．ここでは関数，クラス，名前空間を使って挨拶と加算を行っている．

Listing 1　最小の C++ プログラム

```cpp
#include <iostream>

int add(int a, int b) {
    return a + b; // 2つの数を足して返す
}

class Greeter {
public:
    void hello() const {
        std::cout << "こんにちは" << std::endl;
    }
};

int main() {
    Greeter g;       // 設計図から実体を１つ作成
    g.hello();       // メンバ関数を呼び出し
    std::cout << add(2, 3) << std::endl;
    return 0;        // 正常終了
}
```

このように，`#include` で道具を集め，`main` を入り口として処理を始め，関数やクラスを組み合わせることで複雑な作業を段階的に表現できる．名前空間を使えば工具箱が散らかるのを防ぎ，どの部品を利用しているかが一目瞭然になる．ここまで理解できれば C++ のコードを読む際の抵抗感がぐっと減り，以後のシミュレーション本編にもスムーズに入っていけるはずである．

さらに一歩踏み込むと C++ のソースコードは「翻訳（コンパイル）」と「結合（リンク）」の二段階で実行可能なプログラムへと変換される．複数のファイルをそれぞれ翻訳してから最後に合体させるイメージは，造船所で船体・機関・内

装を別々に製作し，最終工程でドックに並べて一隻の船に仕上げる作業に似ている．ヘッダファイル（.h）には部品の仕様書，ソースファイル（.cpp）には実際の作り方が記され，#include によって仕様書を参照しながら組み立てが進む．

標準ライブラリの存在も重要だ．std 名前空間の中には入出力や文字列操作，数値計算，コンテナといった再利用可能な道具が豊富に用意されており，これらを活用することでゼロから車輪を作り直す手間を避けられる．必要な道具だけを選び，using std::cout; のように個別に呼び込めば，棚のどこに何があるかを明確にしたまま作業を進められる．

反対に，using namespace std; と倉庫全体を丸ごと作業場に持ち込むと，同じ名前の部品が衝突して混乱する危険がある．本プロジェクトではこのような無差別な持ち込みを避け，適切な区画管理を徹底している．小さな心掛けだが，長期的には保守性と可読性を大きく左右するポイントである．

A.2　ポインタと参照の基礎

変数に別名を付けるのが「参照」で，住所をメモした紙切れが「ポインタ」である．参照はあだ名と同じで元の人間がいなければ存在できず，必ず誰か本体を指している．一方ポインタは空白のメモを持てるため，「まだ誰の家にも行っていない状態」を表現できる．

以下のコードでは，整数 x に対して参照 ref とポインタ ptr を用意し，どちらから値を変更しても同じ実体が書き換わることを示す．*ptr と書く操作は「メモの住所をたどって玄関を開ける」動作に相当する．

Listing 2　参照とポインタの基本

```
int x = 10;
int& ref = x;   // ref は x のあだ名
int* ptr = &x;  // ptr には x の住所が入る

ref = 20;       // x が 20 に更新される
*ptr = 30;      // 住所をたどって x を 30 に

std::cout << x << std::endl; // 30 が表示される
```

ポインタ型 T* は「T 型のための住所型」である．例えば int* は整数の住所，std::string* は文字列オブジェクトの住所，char* は C 風文字列の先頭住所を表す．ポインタには必ず住所を代入しなければならず，int* ptr = x; のように値そのものを入れることはできない．また ptr = 30; と数値を直接書き込むと，存在しない番地を指し示してしまう．正しくは int* ptr = &x; のように & で住所を取得し，*ptr = 30; と書いて初めて中身を更新できる．参照は一度結び付けた相手を変更できないが，ポインタは別の住所を代入して付け替えられる点も覚えておきたい．

参照とポインタのもう一つの違いは「付け替えの可否」である．参照は一度あだ名を付けると別の対象に変更できないが，ポインタは住所メモを書き換えるだけで新しい家を指し示せる．この性質は，あるエージェントが次々と異なる基礎を扱うようなケースで便利に働く．一方で，メモを書き換えた直後に古い住所の住人が家を取り壊してしまうと，ポインタは宙ぶらりんの危険な紙切れ（ダングリングポインタ）となる．これを防ぐには，nullptr を使って「今はどこも指していない」と明示したり，スマートポインタで寿命管理を自動化したりするのが有効だ．

ポインタ演算という概念もある．これは地図を片手に「一つ隣の区画へ進む」といった相対移動を行う操作で，配列の先頭アドレスから必要な要素へジャンプする際に使われる．ただし，誤って範囲外へ進むと他人の家に勝手に侵入することになり，プログラムが異常終了する．したがってポインタ演算を使うときは，あらかじめ配列の大きさを把握しておき，越境しないように細心の注意を払う必要がある．

以上のように，参照は「確実に存在する対象への別名」，ポインタは「存在するか分からない対象への住所メモ」と覚えておけば，コードを読む際に役割の違いを直感的に把握できるだろう．

現場の C++ コードでは，生ポインタは「誰が片付けるのか」責任が曖昧になりがちなため，必要最小限にとどめるの

が一般的である．こうしたポインタ設計の習慣は，複雑なシミュレーションでもバグの温床を減らす最初の一歩となる．最初は難しく感じても，住所メモとあだ名という比喩を頭に置いてコードを追えば，指し示す対象と寿命の関係が自然と理解できるはずである．

A.3　コピー／参照／ポインタ比較表

前節で触れた概念を整理するため，引数の渡し方を比較する．

方式	記法	特徴
値渡し	`Environment env`	毎回コピーが発生し大きな構造体では低速・多メモリ．関数内での変更は呼び出し元に影響しない．
参照渡し	`Environment& env`	コピーなしで高速．呼び出し元と同一個体を扱うため関数内の変更が呼び出し元に反映される．
const 参照	`const Environment& env`	コピーなし＋変更禁止の契約を追加．読み取り専用 API に適合．
ポインタ渡し	`Environment* env`	`nullptr` を表現できるが，ヌルチェックが必要．所有権の意図が曖昧になりがち．

```
struct Vec2 { double x, y; };
void byValue(Vec2 v) { v.x = 1; }
void byRef(Vec2& v) { v.x = 1; }
void byConstRef(const Vec2& v) { /* v.x = 1; // \texttt{const} なのでエラー */ }
void byPtr(Vec2* v) { if (v) v->x = 1; }
int main() {
    Vec2 p{0,0};
    byValue(p);      // p.x は 0 のまま
    byRef(p);        // p.x は 1 になる
    byConstRef(p);   // 読み取り専用
    byPtr(&p);       // p.x は 1 になる
}
```

続いて，所有権管理を自動化するスマートポインタを紹介する．

A.4　スマートポインタ入門

生ポインタはメモの紛失に注意が必要で，誰も参照しなくなってもメモリ上に物が残り続ける危険がある．そこで鍵束の比喩で説明される `std::shared_ptr` が登場する．共有の鍵束を持っている人が一人でもいれば部屋（オブジェクト）は維持され，誰も鍵を持たなくなった瞬間に自動的に片付けられる．

Listing 3　shared_ptr による共有と自動掃除

```
auto foundation = std::make_shared<FloatingFoundation>();
std::shared_ptr<FloatingFoundation> a = foundation; // 鍵束を共有
{
    std::shared_ptr<FloatingFoundation> b = a; // もう１人が鍵を持つ
} // b が範囲を抜けると鍵を返却
// a が破棄された時点で部屋は自動的に掃除される
```

各行の意味をもう少し詳しく追ってみよう．

1. `std::make_shared<FloatingFoundation>()` はヒープ領域に `FloatingFoundation` を確保し，参照カウント付きの共有ポインタを生成する．ヒープとは `new` などで動的に確保され，スコープを抜けても自動では解放されない領域である．`auto` により `foundation` の型は `std::shared_ptr<FloatingFoundation>` と推論される．
2. 既存の鍵束 `foundation` をコピーして `a` も同じ部屋を指す．この時点で参照カウントは 2 になる．
3. 3–5 行目ではブロック内で `b` が追加の鍵を受け取る．ブロックを抜けると `b` のデストラクタが自動的に呼ばれ，参照カウントが 1 減る．
4. 外側のスコープを抜けると `a` も破棄される．最後の鍵束が手放された瞬間に参照カウントが 0 となり，`FloatingFoundation` のデストラクタが実行されて部屋が片付けられる．明示的な破棄コードは不要で，RAII（Resource Acquisition Is Initialization）の原則に従って自動的に行われる．

この仕組みにより，複数のエージェントが同じ基礎を安全に共有できる．参照では鍵を手放すタイミングを制御できないが，共有ポインタなら所有者数がゼロになった瞬間に自動破棄されるため，後片付けのし忘れを心配する必要がない．研究者が複数のデータセットを同時に扱う状況にも似ており，誰かが利用を終えたら鍵を返却していくイメージで理解すると覚えやすい．

```
auto sp = std::make_shared<Vec2>(0.0, 0.0); // 自動で new/delete を管理
use(sp); // 引数は const std::shared_ptr<Vec2>& とするのが一般的
if (sp.use_count() == 1) { /* 自分だけが所有 */ }
```

生のポインタの代わりに `std::shared_ptr` を使うとメモリ解放のし忘れを防ぎ，複数のオブジェクトで安全に共有できる．所有権が一意な場合は `std::unique_ptr` も選択肢である．参照で済む場面では積極的に参照を用い，所有権の共有が必要なときだけスマートポインタを選ぶといった住み分けを意識したい．

A.5　変更手順の黄金律

本システムを改造する際は「`constants.h` を直す→マップ JSON を直す→再ビルド→ Python 可視化」という順序を守ると安全である．これは料理のレシピを変えるときに，まず材料表を書き換え，次に買い物リストを更新し，調理器具を用意してから試食する手順に相当する．

最初の `constants.h` は，すべてのエージェントが参照する共通設定の集約場所であり，時間刻みや船の台数といった基礎的なパラメータが定義されている．ここを更新し忘れると，プログラム全体で古い値が使われ続け，どこを修正しても挙動が変わらないといった混乱を招く．言い換えれば，機械の制御盤のダイヤルを回さずに現場の作業員へ口頭で指示を出しているような状態になり，事故のもとだ．

続いて編集する JSON は，現場の地図や設備配置を記した「施工図面」である．港内の座標や容量，接続関係などがここに記載され，`constants.h` の値と矛盾があると実体化したエージェントが迷子になる．図面を更新した後は，必ずシンタックスチェックを行い，構造が壊れていないかを確かめることが大切である．

例えばシミュレーション時間の刻みを 3 分に変更したい場合，`constants.h` では次のように値を更新する．

Listing 4　constants.h の変更例

```
// 時間刻み（時間単位）
constexpr double TIME_STEP_HOURS = 0.05;
// 分単位で参照したい場合は変換関数を使用
const SimTime stepMin = hours_to_min(TIME_STEP_HOURS);
```

このように刻み幅の基準は TIME_STEP_HOURS のみで管理し，分刻みが必要な処理では hours_to_min 関数で逐次換算する．値を二重に保持しないため，刻み幅改定時は TIME_STEP_HOURS のみを調整すれば全体が自動的に連動する．

続いて港湾マップの設定ファイルである JSON を書き換える．ヤードの座標やバース数など，constants.h に合わせて整合が取れているかを確認する．

Listing 5　港湾マップ JSON の一部

```
{
  "yard": {
    "capacity": 4,
    "position": [100, 50]
  }
}
```

設定を保存したらルートディレクトリで make を実行し，バイナリを再生成する．最後に Python スクリプトで結果を可視化すれば，変更が想定通り反映されたかを目で確かめられる．

Listing 6　ビルドと可視化のコマンド

```
$ make
$ cd cpp && ./main.exe
$ cd ../python && python visualize.py
```

実行ログにエラーがないことを確認したら，logs/ ディレクトリに生成される CSV やテキストをチェックし，値が想定通り変化しているかを目視する．ここで違和感に気付いた場合は，まず constants.h と JSON の差分を見比べ，設定の打ち間違いや桁のズレがないかを疑うと良い．小さな誤りでも後工程で大きな遅延を招くため，早期発見が重要である．

この黄金律を守ることで，設定値の不整合やビルド漏れによる「動かない」問題を避けられる．研究プロジェクトでは多数のパラメータを試行錯誤することになるが，作業のたびに上記の順番で手続きを踏むことで，どの段階で問題が発生したかを切り分けやすくなる．まるで造船所で工程ごとに検査票へチェックを入れていくように，改造の履歴と結果を丁寧に追跡する姿勢が品質を守る鍵となる．

さらに細かなコツとして，make clean で古い成果物を消してから再ビルドすると，過去の設定が混入する事故を防げる．JSON を編集した際には，鍵の綴り間違いがないか python -m json.tool map.json で整形確認すると安心だ．可視化結果は日付付きのファイル名で保存し，どの設定で得られたグラフなのかを注釈しておくと，後日比較する際に迷わない．

最終的に変更が意図通り動作することを確認したら，忘れずに Git へコミットする．コードと同時に constants.md や本ドキュメントの該当箇所を更新しておけば，「どのパラメータがいつ変わったか」をチーム全員で共有できる．こうした手順の徹底は，研究の再現性を高めるだけでなく，将来の自分が過去の変更理由を振り返る際の助けにもなる．地道な手順に思えるが，プロジェクトが大規模になるほどこのチェックリストが威力を発揮する．どこか一か所でも抜け落ちると，原因追跡に膨大な時間を費やすことになりかねない．黄金律を日常的に実践し，変更内容と結果を確実に記録することが，信頼できるシミュレーション基盤を育てる最短ルートである．

A.6　末尾アンダースコアの意味

クラスは自分固有の状態を保管する箱で，その中に入る値がメンバ変数である．メンバ変数はオブジェクトが破棄されるまで生き続ける．新しいオブジェクトを作る際に外部から渡す初期値がコンストラクタ引数であり，読み込まれると役目を終える．関数内で計算を助ける短命の一時値がローカル変数で，処理が終われば消える．本プロジェクトではメンバ

変数にのみ末尾のアンダースコアを付け，radius_ のように書いてローカル値 radius と一目で区別する．この習慣が擬似コードとの対応を明確にし，複数人で解析する際の混乱を抑える．

本付録では，本モデルの中核である Agent と Environment，そしてそれらを結ぶ EventBus 設計を，C++ の言語機能（参照，const，仮想関数，静的メンバ）と結び付けて詳解する．サンプル断片はすべて本体コードに準拠し，そのままビルド可能な最小例もあわせて提示する．

A.7　Agent クラスの詳細構造

■ 宣言の再掲と要点

```
class Agent {
protected:
    int id_;
    Vec2 pos_;
    std::deque<Vec2> route_;
    std::string status_;
    EventBus* bus_;
    static std::shared_ptr<Environment> env_;
public:
    virtual void step(double dt) = 0;
    virtual void receive(const Message& msg) = 0;
    void planRoute(const Environment& env);
    void moveAlongRoute(double dt, const Environment& env);
};
```

■ アクセス修飾子の選択理由

- **protected メンバ**（id_, pos_, …）：派生クラス（Dock, TowFleet など）から直接アクセスしたいが，外部関数からの直接変更は避けたいため．
- **public インターフェース**：step と receive は派生クラス実装の入口であり，外部から多態的に呼び出される．
- **static メンバ**（env_）：全エージェントで共有する環境を 1 箇所に集約し，ライフサイクル管理と依存関係を簡素化する意図．

■ 公開継承と可視性の境界

派生クラスは class Dock : public Agent のように基底を明示し，: public Agent は公開継承を示す．これにより Agent の public 成員が Dock からもそのまま利用・外部公開される．一方 public: 等のラベルはクラス内部の可視範囲を区切る仕切りであり，外部・派生・自クラスのアクセス可否は表 19 のとおりである．

	外部	派生	自クラス
public	✓	✓	✓
protected	×	✓	✓
private	×	×	✓

図 19　アクセス修飾子と到達範囲

■ 直感的な理解

class Dock : public Agent と宣言すると Dock は Agent の一種となり，Agent の公開 API をそのまま外部に示す．クラス内部の public:，protected:，private: はそれぞれ玄関ドア，社員専用通路，自分の机の引き出しに対応し，誰が触れられるかの境界を切り替える．表 19 は public が全員，protected が自分と派生クラス，private が自分のみアクセス可能であることを整理したものであり，公開継承はインタフェースの再公開を意味する．

■ 純粋仮想関数（=0）の意味

step と receive は「必ず派生クラス側で実装すべき」契約を表す．これにより Agent は抽象クラスとなり，インターフェースの役割を果たす．

■ 小さなアンチパターン

- すべてを public にする：外部からの恣意的な書き換えで不変条件が破られやすい．
- すべてを private にする：派生で必要な拡張が困難（ゲッター／セッターだらけになり可読性を損ねる）．

■ 派生クラス実装のための整理

Agent は内部状態を protected メンバで保持し，環境への参照 env_ を static std::shared_ptr として共有する設計になっている．派生クラスはこれらのメンバに直接アクセスできるため，状態更新や環境参照を自然な形で記述できる．さらに step と receive を純粋仮想関数として宣言することで，派生側に振る舞いの実装を強制し，多態的な呼び出し点を明確化する．アクセス修飾子と仮想関数の役割を理解しておくと，新しいエージェントを追加する際の指針が得られ，実装漏れを防げる．

A.8　参照（&）と const の実践

■ なぜ参照を使うのか

Environment はゾーン・障害物・バースなど多数のデータを保持する大きなオブジェクトである．関数引数を Environment env（値渡し）にすると毎回コピーが発生し非効率になる．そこで **参照渡し**（Environment& env）または **const 参照渡し**（const Environment& env）を使う．後者は「**呼び出し先で書き換えない**」ことを型で保証する．

■ 末尾 const（メンバ関数修飾子）

```
std::deque<Vec2> Environment::findPath(
    const Vec2& start, const Vec2& goal, double step) const; // ←末尾  const
```

この const は「findPath の中で Environment の状態を変更しない」という保証である．したがって，下記のように読み取り専用の環境（const Environment）からも呼び出せる：

```
const Environment env = Environment::loadFromJson("map.json");
auto path = env.findPath(a, b); //  (末尾OK const があるため)
```

もし末尾の const を外すと，次のようなエラーになる（概念図）：

```
error: passing 'const Environment' as 'this' argument discards qualifiers
```

A.9　静的メンバ env_ とライフサイクル

■ 設計意図

各エージェントに環境のコピーを持たせるのではなく，**単一の共有環境**を指すポインタ（std::shared_ptr<Environment>）をクラス静的メンバとして保持する．この方式は次の効果を狙う：

- どのエージェントも同一の港湾地図・速度プロファイルを参照するため整合性が保たれる．
- 環境の寿命管理を shared_ptr に委ねられる（最後の参照が消えた時に解放）．
- テスト時に Agent::setEnvironment(mock) のように差し替えが容易．

■ 設定と利用の最小例

```
int main() {
    auto env = std::make_shared<Environment>(
        Environment::loadFromJson("../data/maps/port_map_detailed.json"));
    Agent::setEnvironment(env); // ←全エージェントに共有環境をセット

    EventBus bus;
    auto dock = std::make_shared<Dock>(DOCK_ID, FOUNDATION_MAKE_LINES, FOUNDATION_MAKE_DAYS);
    dock->setEventBus(&bus);
    // ... 以降，agents を生成してループへ
}
```

■ 注意点

- 共有資源に対する**破壊的変更**は全エージェントへ波及するため，読み取り主体の API はできるだけ const を付ける．
- 並列化する場合はスレッド安全性（ロックやデータ分割）を別途設計する必要がある．

A.10　EventBus と疎結合設計

■ なぜ直接呼び出さないのか

エージェント同士がメソッドを直接呼び合うと依存関係が絡み合い，順序問題や循環参照が発生しやすい．**EventBus** は Observer/Publish-Subscribe パターンに基づき，送信者と受信者を疎結合に保つ：

- 送信側：bus.post(Message{sender, receiver, type, payload});
- 配送側：キューから取り出し，receiver に応じて Agent::receive または Environment::handleMessage へ渡す．
- 受信側：receive 内で型安全に処理（std::variant を利用）．

■ メリット

- モジュール境界が明確になりテストが容易（モック EventBus で検証可能）．
- ブロードキャストや将来のログ・監査フックを挿入しやすい．

A.11　設計パターンとの対応

- **Observer / Pub-Sub**（EventBus）：イベント発行と購読の分離.
- **State**（`FloatingFoundation::stage_`）：工程段階の遷移を明示.
- **Strategy**（各派生 Agent の `step` 実装）：共通インターフェースに対し戦略を差し替え.
- **Context**（Environment）：全体コンテキストを集約し読み取り API を提供.

A.12　失敗例と修正版

■ (1) 値渡しで重い引数をコピー

```
// 悪い例：毎回 Environment をコピー
void Agent::planRoute(Environment env) { /* ... */ }
```

修正：必ず `const Environment&` にする.

```
void Agent::planRoute(const Environment& env) { /* ... */ }
```

■ (2) 末尾 const を付け忘れ

```
// 悪い例：読み取り専用のはずが const で呼べない
std::deque<Vec2> Environment::findPath(const Vec2& s, const Vec2& g, double step) {
// ... 状態は変更しない実装 ...
}
```

修正：

```
std::deque<Vec2> Environment::findPath(const Vec2& s, const Vec2& g, double step) const {
// ... 状態を変更しないことを保証 ...
}
```

■ (3) 直接呼び出しで依存が密結合

```
// 悪い例：Foundation が Dock の内部 API を直接呼ぶ
foundation->dock()->startFabrication(foundation);
```

修正：EventBus を介して要求を発行し，受け手がキュー処理する.

```
send(dockId, MessageType::JoinQueue);
```

A.13　最小動作例：Agent を 1 体だけ動かす

■ ミニマル Agent

```
class DummyAgent : public Agent {
public:
    explicit DummyAgent(int id) : Agent(id) { setStatus("idle"); }
    void step(double dt) override {
```

```
        (void)dt;
        if (route_.empty()) {
            auto targets = environment()->getTargets();
            if (!targets.empty()) {
                auto goal = targets.front();
                auto path = environment()->findPath(position(), goal, 1.0);
                route_ = std::deque<Vec2>(path.begin(), path.end());
                setStatus("moving");
            }
        } else {
            moveAlongRoute(dt, *environment());
        }
    }
    void receive(const Message& msg) override { (void)msg; }
};
```

■ 最小 main

```
int main() {
    auto env = std::make_shared<Environment>(Environment::loadFromJson("map.json"));
    Agent::setEnvironment(env);
    EventBus bus;

    auto a = std::make_shared<DummyAgent>(1);
    a->setEventBus(&bus);
    a->setPosition({10, 10});

    std::vector<std::shared_ptr<Agent>> agents = {a};
    for (int t = 0; t < 100; ++t) {
        bus.deliverAll(agents, env.get());
        for (auto& ag : agents) ag->step(1.0);
    }
}
```

A.14　C++ における引数の渡し方

A.15　値渡し Environment env

コピーを作って渡すイメージ．呼び出し元のオブジェクトを丸ごと複製し，そのコピーを関数に渡す．関数内でどれだけ変更しても呼び出し元には影響しない．ただし大きな構造体ではコピーに時間とメモリがかかる．

```
void foo(Environment env) {
    env.width = 100; // コピーに書き換え→呼び出し元には影響なし
}
```

A.16　参照渡し Environment& env

実体をそのまま渡すため，高速でメモリ効率も良いが，関数内の変更が呼び出し元に反映される．

```
void foo(Environment& env) {
    env.width = 100; // 呼び出し元の width が 100 に変わる
}
```

A.17　const 参照渡し const Environment& env

本人を借りるが変更しないことを約束して使う．コピーが発生せず高速で，読み取り専用 API に適する．コンパイラが関数内での変更を禁止し安全性を保証する．

```
void foo(const Environment& env) {
    double h = env.height;  // 読み取りはOK
    // env.height = 200;    // コンパイルエラー！変更禁止
}
```

A.18　ポインタ渡し Environment* env

本人の住所メモを渡す形で *env として実体にアクセスする．nullptr が来る可能性があるため必ずチェックが必要で，所有権の扱いも曖昧になりがち．

```
void foo(Environment* env) {
    if (env) {
        env->width = 100; // 呼び出し元を直接変更
    }
}
```

A.19　まとめ

- コピーして独立して使う → 値渡し
- 呼び出し元を直接操作したい → 参照渡し
- 読み取り専用で速く → const 参照
- nullptr の有無を扱いたい／C 互換が必要 → ポインタ

■ 研究コードでの推奨

Environment のような重い構造体では基本 const Environment& で渡し，変更が必要なときだけ Environment& を使う．ポインタは「存在しないかもしれない」場合に限定するのが安全である．

A.20　FAQ（よくある疑問）

■ Q1: const Environment env と const Environment& env の違いは？

前者は**読み取り専用のコピー**（値）を受け取る．後者は**読み取り専用の参照**で，コピーが発生しない．大きな構造体では後者を使う．

■ Q2: 引数をポインタ（Environment*）にしないのは？

nullptr 取扱いと所有権の曖昧さが混ざるため．本設計では存在が前提の共有環境なので参照が自然．

■ Q3: 末尾 const を付けられるか迷ったら？

関数内でメンバを書き換えていないなら基本的に付ける．IDE の *read-only* チェック（this が const）で設計バグが早期に露見する．

A.21　まとめとガイドライン

- 大きなデータは **const 参照**で受ける（`const T&`）.
- 読み取り API は**末尾** `const` を徹底する.
- エージェント間は **EventBus** で疎結合に保つ.
- 環境は**静的共有**し，差し替え可能な形にする.

以上により，拡張容易性・性能・可読性のバランスを取りつつ，研究用途の反復シミュレーションに耐える設計となる.

A.22　Message 型と `std::variant`

メッセージの箱である `Message` 構造体は「誰が」「誰へ」「どんな種類の手紙か」とともに，中身を `std::variant` で持ちます．`variant` は「どれか 1 つ」を入れられるお弁当箱のようなものです.

```
enum class MessageType { TowRequest, Status, Ping };
using Payload = std::variant<std::monostate, Vec2, std::string>;
struct Message {
    AgentID sender;
    AgentID receiver;
    MessageType type;
    Payload payload;
};
```

1–2 行目　送る手紙の種類を `enum class` で列挙．`Payload` は座標や文字列など「どれか 1 つ」を入れられる箱です.
3–7 行目　宛先や差出人と一緒に `payload` を詰めた `Message` 構造体.

ペイロードは `std::variant` により単一の型しか保持しない箱である．本実装では `std::monostate` により空を表現し，座標を渡す `Vec2`，文字列メッセージ用の `std::string`，簡単な数値 `int`，資材一覧を示す `Bom` 等から成る．送信側は `Message{..., payload}` として詰め，受信側では `std::get<T>(payload)` で直接参照するか，`std::visit` を用いて型ごとに処理を分岐させる.

中身を安全に取り出すには `std::visit` を使います．`visit` は `payload` の実際の型に合わせて関数を呼び分けてくれます.

```
void Agent::receive(const Message& m) {
    std::visit([](auto&& v){
        using T = std::decay_t<decltype(v)>;
        if constexpr (std::is_same_v<T, Vec2>) {
            /* 座標を受け取ったときの処理 */
        } else if constexpr (std::is_same_v<T, std::string>) {
            /* 文字メッセージを受け取ったとき */
        } else {
            /* 何も入っていないとき */
        }
    }, m.payload);
}
```

この仕組みにより，型の取り違えによるバグをコンパイル時に防げます．`if` で型名をチェックするよりも短く，読みやすく書けるのが利点です.

■ ポイント

- enum class でメッセージ種別を明示すると，他の列挙体との混同を防げます．
- std::variant は複数型の値を 1 つにまとめる便利な入れ物です．
- std::visit は「今どの型が入っているか」を自動で見分け，対応する処理を呼び出します．

A.23　Environment の構造とパス探索

Environment は港の地図や障害物を丸ごと抱える大きな箱です．格子状の升目を想像し，船がマス目を 1 つずつ進む様子を考えると理解しやすくなります．

```
struct Zone { std::string name, type; double x, y, w, h; };
struct Obstacle { double cx, cy, w, h, angle; };
class Environment {
public:
    double width, height;
    std::vector<Zone> zones;
    std::vector<Obstacle> obstacles;
    std::deque<Vec2> findPath(const Vec2& s, const Vec2& g, double step = 1.0) const;
};
```

1–2 行目　ゾーンや障害物の最小単位．w,h は幅と高さ，angle は角度です．

3–8 行目　Environment が持つ地図情報と，findPath という道案内関数．

■ A*法による道案内

findPath は step $= 1.0$ の正方格子上で四近傍を探索する A* アルゴリズムである．評価関数 $f(n) = g(n) + h(n)$ は，実コスト g とヒューリスティック h から構成される．

探索格子と近傍　候補点 $\boldsymbol{p} = (x, y)$ の近傍は

$$\mathcal{N}(\boldsymbol{p}) = \{(x \pm \texttt{step}, y),\ (x, y \pm \texttt{step})\} \tag{4}$$

の四近傍である．各遷移の実コストは

$$g_{k+1} = g_k + 1 \tag{5}$$

で等しく，速度差を考慮しない．ヒューリスティックはゴール $\boldsymbol{g}$ とのユークリッド距離

$$h(\boldsymbol{p}) = \|\boldsymbol{p} - \boldsymbol{g}\|_2 \tag{6}$$

を用いる．

通行判定　isTraversable は候補点が障害物に含まれず，かつ水域タイプのゾーンに属するとき真となる．陸地や非水域ゾーンは探索対象から除外される．

ステップ幅と障害物　障害物厚みは step 以上を推奨する．より薄い障害物は四近傍の跳躍によってすり抜けが生じる可能性がある．

計算量とステップ幅　マップ幅を W，高さを H とすると，A* が調べる格子点の概数は

$$N = \frac{W}{\texttt{step}} \times \frac{H}{\texttt{step}} \tag{7}$$

であり，計算量は $O(N)$（優先度付きキューを用いた場合は $O(N \log N)$）に比例する．したがって step を半分にすると探索空間はおよそ 4 倍に膨らみ，実行時間とメモリ使用量が急増する．実運用では step $= 1.0$ を標準とし，障害物厚が小さく経路が見つからない場合に限り 0.5 などへ縮小する程度が現実的である．

将来拡張　現状の実コストは一定だが，水域速度 $v(\boldsymbol{p})$ に基づき

$$g_{k+1} = g_k + \frac{1}{v(\boldsymbol{p})} \tag{8}$$

と重み付けすれば，速い水路を優先する経路が得られる．実装は未着手であり設計方針のみ示す．

アルゴリズム概要

1. 開始点を開リストに格納する．
2. f 値が最小のノードを取り出し近傍を展開する．
3. 進入可能な近傍に対し g と f を更新し，再び開リストに追加する．
4. ゴールに到達したら親ポインタを遡って経路を復元する．

■ 最小コードイメージ

```
std::deque<Vec2> Environment::findPath(const Vec2& s, const Vec2& g, double step) const {
    struct Node { int x, y; double f; };
    std::priority_queue<Node> open; // 次に調べるマス
    // ... A* の本文（詳細は実装コードを参照） ...
    return {}; // ルート（見つからなければ空）
}
```

格子の一歩を step で決められるので，細かいルートでも荒いルートでも自由に計算できる．既定値は 1.0 ユニットである．障害物の厚みより大きい値を与えると経路が壁を飛び越え，結果として「すり抜け」が生じ得る．壁や陸地は isTraversable で事前に弾かれるため，適切な step を選べば船は安全な水路だけを進む．

A.24　EventBus と優先度付きメッセージング

メッセージのやり取りは EventBus が司令塔である．ここではすべてのメッセージが std::priority_queue にたまり，時計の順番で配達される流れを，追っていく．

A.25　メッセージが届くまで

1. Agent や Environment が post でメッセージと「いつ届けるか」を EventBus に渡す．
2. EventBus はそれを priority_queue に入れる．早く届ける手紙ほど先頭に並ぶ郵便箱を想像してください．
3. シミュレーションが進んで deliverAll が呼ばれると，配達の時間になったメッセージから順に取り出す．
4. 宛先が BROADCAST_ID なら全員に配達，特定の AgentID ならその子だけ，負の ID なら Environment に渡す．
5. それぞれ Agent::receive や Environment::handleMessage が呼ばれ，中身に応じて動く．

A.26　最小コード例

```
struct QueuedMessage {
    SimTime deliverAt;
    Message msg;
    bool operator<(const QueuedMessage& other) const {
        return deliverAt > other.deliverAt; // 早い順
    }
};
```

```
8
9  void EventBus::post(SimTime t, const Message& m) {
10     queue_.push({t, m});
11 }
12
13 void EventBus::deliverAll(const std::vector<std::shared_ptr<Agent>>& agents,
14                            Environment* env) {
15     while (!queue_.empty() && queue_.top().deliverAt <= current_) {
16         auto q = queue_.top();
17         queue_.pop();
18         if (q.msg.receiver == BROADCAST_ID) {
19             for (auto& a : agents) a->receive(q.msg);
20         } else if (q.msg.receiver >= 0) {
21             agents[q.msg.receiver]->receive(q.msg);
22         } else {
23             env->handleMessage(q.msg);
24         }
25     }
26 }
```

A.27　頭の中で描くイメージ

```
1  post --> [priority_queue] --deliverAll--> Agent::receive()
2                            \--deliverAll--> Environment::handleMessage()
```

■ ポイント

- 優先度付きキューで「いつ配達するか」を自動で並べ替えられる.
- deliverAll は仮想関数を通じて多様な Agent の振る舞いを引き出す.
- 新しい Agent が増えても EventBus 側の変更はほとんど不要.

付録 B　シナリオ再現用資料と JSON 入力例

本付録では入力／出力仕様に関する具体的なサンプルをまとめ，シミュレーションシナリオを再現する手順を示す．

B.1　マップ JSON の項目一覧

マップ定義は表 9 のキーを持つ JSON で記述する．

キー	内容
width	マップ横幅 [m]
height	マップ縦幅 [m]
zones	ゾーン配列．各要素は name, type, x, y, w, h を持つ
obstacles	障害物配列．軸平行矩形 (x,y,w,h) と回転矩形 (cx,cy,w,h,angle) を許容し，欠損キーは自動補完される
targets	目標地点配列（空配列も可）
speed_profiles	ゾーン種別ごとの移動速度．フラット形式と階層形式の双方を受理
berths	id, quay, x, y, depth_m の配列．`Environment.berths[quay]` に格納される

表 9　マップ JSON の主要項目

B.1.1　obstacles

`obstacles` は矩形障害物を列挙する．中心座標（`cx`, `cy`）と寸法（`w`, `h`）に回転角 `angle` を加えた回転矩形と，左下隅（`x`, `y`）と寸法（`w`, `h`）の軸平行矩形を併用できる．いずれかの形で不足したキーはゼロや既定値で補完される．

```
"obstacles": [
  {"x": 0, "y": 0, "w": 10, "h": 5},
  {"cx": 20, "cy": 20, "w": 6, "h": 3, "angle": 30}
]
```

左は軸平行矩形であり角度が欠けたため 0° と解釈される．右は中心を基準とする回転矩形であり左下隅は自動計算される．

B.1.2　speed_profiles

速度プロファイルは二つの記法を受け付ける．単純なフラット形式では `"sea":1.0` のようにゾーン種別から速度を直接指定する．階層形式では `fallback` にゾーン種別→速度，`override_by_zone` にゾーン名→速度を与え，同名ゾーンに対する上書きを表現する．いずれの場合も最終的には同一のテーブルに展開される．

```
"speed_profiles": {
  "sea": 1.0,
  "channel": 0.8
}
```

フラット形式の例であり，ゾーン種別から速度を直接決定する．

```
"speed_profiles": {
  "fallback": {"sea": 1.0, "channel": 0.8},
  "override_by_zone": {"Canal1": 0.5}
```

```
}
```

ゾーン定義に Sea, ChannelA, Canal1 があるとき，Canal1 の速度は 0.5 に上書きされ，読み込み後の表は Sea → 1.0，ChannelA → 0.8，Canal1 → 0.5 となる．

■ 速度プロファイル解決 API

速度の決定は関数

$$v(z) = \texttt{speedForZone}(z) \tag{9}$$

により行う．入力 z はゾーン名 n あるいは種別 t を含む識別子であり，出力は単位時間あたりの速度 $v(z)$ である．実装上のシグネチャは `double Environment::speedForZone(const std::string& nameOrType)const` であり，返却値は常に非負の実数である．処理手順は以下の通りである．

1. n が override_by_zone に存在すれば速度 v_n を返す．
2. 上書きが無い場合，ゾーン種別 t に基づき fallback から速度 v_t を取得する．
3. いずれも該当しなければ既定値 v_{const}（定数 TOW_SPEED_PORT）を返す．

式で書けば

$$v(z) = \begin{cases} v_n & (n \in \texttt{override\_by\_zone}) \\ v_t & (t \in \texttt{fallback}) \\ v_{\text{const}} & \text{otherwise} \end{cases} \tag{10}$$

となる．図 20 に処理の決定木を示す．

例としてゾーン ApproachLane（type=channel）を照会すると，override_by_zone.ApproachLane が存在すればその値を返す．存在しない場合は fallback.channel に定義された値を用い，それも無ければ定数 TOW_SPEED_PORT を返す．

B.1.3　目標点スナップ（snapToWaterTarget）

陸上の座標を牽引対象に指定すると，その地点から最寄りの水域へ自動的にスナップされる．snapToWaterTarget 関数は入力ベクトル $\boldsymbol{r} = (x, y)$ の属するゾーンを調べ，その種別が水域であれば $\boldsymbol{r}$ をそのまま返す．陸上であればマップ JSON の water_types に列挙された水域種別（既定値は {dock, basin, channel, sea}）を優先順位順に走査し，ゾーン中心 $\boldsymbol{c}_t$ を探索して最初に見つかったものへ遷移する:

$$\boldsymbol{r}' = \begin{cases} \boldsymbol{r} & (t(\boldsymbol{r}) \in \text{water}) \\ \arg\min_{\boldsymbol{c}_t} \|\boldsymbol{c}_t - \boldsymbol{r}\| & \text{otherwise} \end{cases} \tag{11}$$

ここで $t(\boldsymbol{r})$ は $\boldsymbol{r}$ を含むゾーンの種別であり，水域判定に失敗した場合でも最寄りの水域中心が返される．water_types の比較は大文字小文字を区別しないため，マップ側で Sea や CHANNEL のように記述しても同じ水域として扱われる．この処理により，地上目的地でもシミュレーションが停止せず，牽引隊は最寄りの航行可能水域へ向かう．

■ ログ出力

牽引艦隊 TowFleet はパス探索に失敗した際，元のゾーン名とスナップ後のゾーン名をログに残す．形式は no path: rawZone=*R* navZone=*N* であり，R が指定された座標のゾーン，N がスナップ先のゾーンを示す．差異の確認や設定ミスの追跡に利用できる．

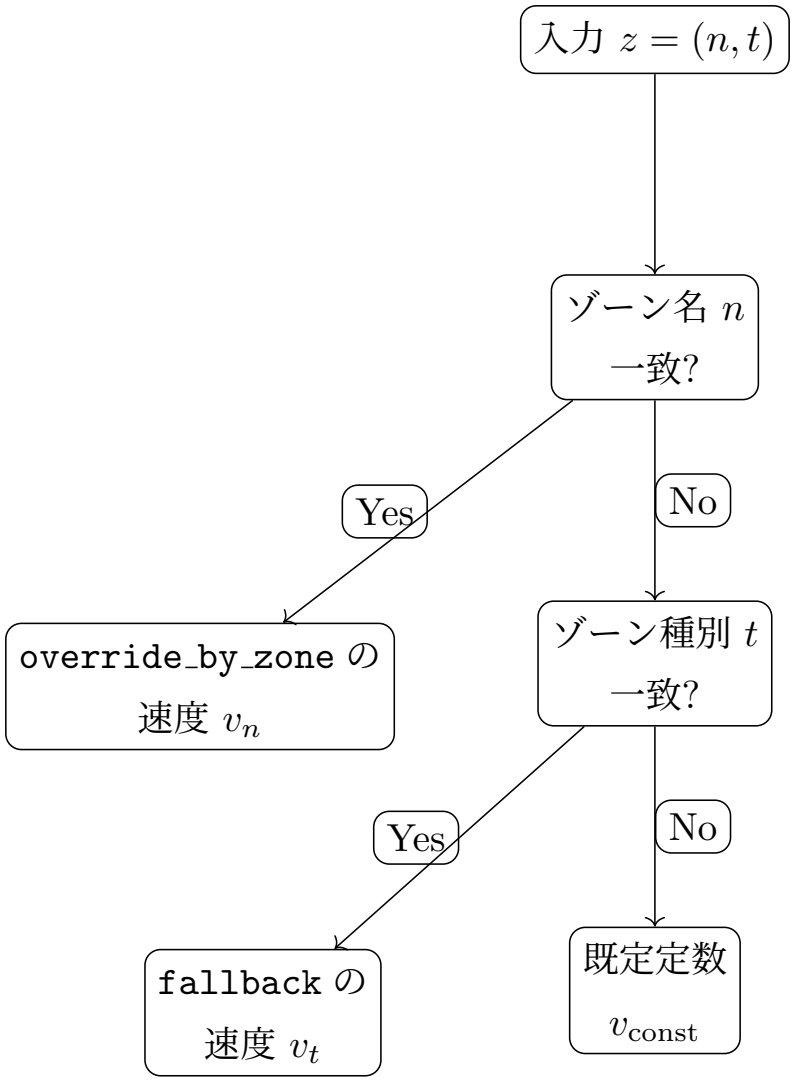

図 20　speedForZone における速度選択の決定木

B.1.4　berths

berths は接岸地点の配列である．各要素は識別子 id，岸壁名 quay，座標 (x, y)，水深 depth_m を持つ．読み込み時に Environment.berths[quay] に分類され，使用状況の管理に供する．

```
"berths": [
  {"id": 1, "quay": "East", "x": 30, "y": 40, "depth_m": 10},
  {"id": 2, "quay": "East", "x": 35, "y": 42, "depth_m": 10},
  {"id": 3, "quay": "West", "x": 10, "y": 40, "depth_m": 12}
]
```

この例では Environment.berths["East"] に 1 と 2 が追加され，Environment.berths["West"] には 3 が格納される．

■ バース資源管理 API

港湾内のバースは有限資源であり，以下の関数により占有状態を制御する．

`freeBerths(quay)`　入力: 岸壁名 q．出力: 未占有バース番号集合 $\{i\}$．未知の岸壁では空集合．

`acquireBerth(quay, i)`　q と番号 i を指定して占有を試みる．成功時は `true` を返し，存在しない岸壁・範囲外番号・既占有のいずれかであれば `false`．

`releaseBerth(quay, i)`　q と i を指定して占有フラグを解放する．無効な入力は無視され副作用を生じない．

`pickAnyFreeBerth(quay)`　岸壁 q における最初の空き番号を返す．空きが無い場合や岸壁が未定義の場合は -1．

これらの API はマップ JSON に `berths` が存在する場合に有効である．`berths` が欠落しているときは後方互換のため，定数 `DOCK_BERTHS` および `YARD_BERTHS` に基づくグローバルカウンタ処理へフォールバックする．現状，牽引艦隊 `TowFleet` はなおグローバルカウンタを使用しており，将来的にバース API へ移行する予定である．

■ ログ仕様

バースの取得および解放イベントは将来, `berth acquire quay=<Q> idx=<i>`, `berth release quay=<Q> idx=<i>` の形式でログ出力する計画である．受信側での解析を容易にするため，ログには岸壁名とバース番号が含まれる．

B.2　座標系と単位系

座標系は数学座標系である．原点を左下に置き，x は右向きを正，y は上向きを正とする．距離と速度の基本単位は**ユニット**であり，`speed_profiles` の値はユニット毎時として解釈する．マップ JSON に `scale_m_per_unit` を含めることで，1 ユニットあたりの実長 [m] を定義できる．この係数を用いれば，ユニット毎時を m/h へ換算できるが，現行の C++ 実装ではまだ参照していない．

B.3　サンプル CSV

`logs/samples.csv` はエージェントの状態を定期的に出力する．最小例はリポジトリ内 `data/sample_samples.csv` にも同梱している．

```
time,id,type,x,y,zone,status,destinationName
6,10,Dock,110,480,Dock,idle,Dock
6,15,TowFleet,110,480,Dock,idle,Dock
6,20,MooringFleet,0,0,SeaOuter,idle,Sea
6,31,YardStorage,400,580,Yard,idle,Yard
```

■ 列仕様と単位

表 10 に各列の意味と単位を示す．座標 (x, y) はマップ固有の**ユニット**で表現され，マップ JSON におけるスケール係数 $s =$ `scale_m_per_unit` を用いると実長へ換算できる:

$$x_{[\mathrm{m}]} = s\,x, \qquad v_{[\mathrm{m/h}]} = s\,v.$$

時間は分単位で記録され，時間 [h] は $t_{\mathrm{h}} = t/60$ により得られる．

■ サンプリング間隔

`samples.csv` の出力頻度は `cpp/main.cpp` において制御される．各ステップでシミュレーション時間 Δt を `TIME_STEP_MIN` (デフォルト 6 分) とその換算値 `TIME_STEP_HOURS` だけ進め, その累積値 $n\Delta t$ が 10 分以上になるたびにログ行が追記される．したがって標準設定では $n = \lceil 10/6 \rceil = 2$ となり, 実際の出力間隔は約 12 分である．より細かな解析が必要であれば, `cpp/include/constants.h` 内の `TIME_STEP_MIN` と `TIME_STEP_HOURS` を整合するように小さく設定し, さらに `main.cpp` の閾値 10 を希望する分数に変更し再ビルドすることで, 任意のサンプリング間隔 $n\Delta t$

列名	内容	単位（例）
time	シミュレーション開始からの時刻 t	分 (例: 6)
id	エージェント識別子	– (例: 10)
type	エージェント種別	– (例: Dock)
x, y	座標	ユニット (例: 110, 480)
zone	位置するゾーン名	– (例: Dock)
status	状態文字列	– (例: idle)
destinationName	進行中の目的地ゾーン	– (例: Dock)

表 10　`logs/samples.csv` の列と単位

を得られる．

将来的には JSON 形式の設定ファイルからこれらの時間刻みを読み込み, 再ビルドなしに実行時に調整できるよう拡張する計画である．

B.4　再現手順

本シミュレーションは C++17 対応のコンパイラで動作するよう設計されており，Win 11 上 g++ 13.2.0 を用いて検証した．同等の環境であれば再現可能である。在庫調達に関わる乱数は `IMPORT_RNG_SEED` (=42) を基点に，ブレード・ナセル・タワーそれぞれへ固有のオフセットを加えたメルセンヌツイスタで生成されるため，部材ごとのリードタイムが相互に独立する。[*7]

1. `data/maps/` にマップ JSON を配置し，必要に応じて `constants.md` のパラメータを調整する．
2. ルートディレクトリで `make` を実行し，C++17 でビルドされた `cpp/main.exe` を生成する．
3. `cd cpp` で実行ディレクトリへ移動し，`./main.exe` を起動する．
4. 実行後，`logs/` 以下に `samples.csv` と `events.csv` が出力されるので，`python/visualize_logs.py` などで解析する．

■ ステータスログモード選択

実行時の標準出力にはシミュレーションの進捗を確認するためのステータスログが流れる。既定では艦隊・資機材グループ単位のサマリーを定期的に表示し，端末をクリアして最新状況のみを描画する。`--status-log-format unit` を指定すると，牽引ユニットや係留艦など個別エージェントごとの位置・目的地・状態を一覧できる詳細モードへ切り替わり，`--status-log-format fleet` または省略時は従来どおりグループ集約表示となる。画面クリアを抑止して追記モードでログを積み重ねたい場合は `--status-clear off` を併用する。[*8]

表示間隔は `--status-interval <分>` で制御でき，指定が無い場合は 30 分ごと（`STATUS_DISPLAY_INTERVAL_MIN`）に描画される。刻み幅はシミュレーションの時間ステップ（既定 6 分）に合わせて自動的に丸め込まれ，必要に応じて調整内容や無効な入力に関する警告が標準エラー出力へ表示される。[*9]

```
./main.exe --status-log-format unit            % 詳細ログ（エージェント単位）
```

[*7] `cpp/src/material.cpp`, `cpp/src/material_stock_agent.cpp`
[*8] `cpp/main.cpp`
[*9] `cpp/include/constants.h`, `cpp/main.cpp`

```
./main.exe --status-log-format fleet            % 集約ログ（既定値）
./main.exe --status-interval 60                 % 分ごとにステータス更新60
./main.exe --status-clear off --status-interval 15   % 分間隔・追記モード15
```

不正な値を指定した場合はエラーメッセージと共に利用可能な選択肢が表示される。`--help` あるいは `-h` を付けて起動すると，上記オプションを含む利用方法が表示されるため，投入時の設定確認に活用できる。

■ 可視化ラインスタイル

表 11 に `python/visualize_logs.py` で使用される線種と色を示す．ケーブル経路のスタイルは `CABLE_STYLE`，帰還経路は `RETURN_LINE_STYLE` に定義されており，拡張時はこれらを更新して整合性を保つこと．

ライン種	色	Matplotlib の `linestyle`
エージェント軌跡	`plt.cm.tab20` より割当	-
帰還経路	black	–
輸出ケーブル	#d62728	-
アレイケーブル	#1f77b4	–
既定ケーブル	#8b4513	-.
敷設中ケーブル（アニメーション）	orange	-

表 11　`python/visualize_logs.py` における線種とスタイル

B.5　JSON 入力例とバリデーション

本付録ではマップなどの入力ファイルを JSON 形式で記述する際の最小例と，構文および必須項目の検証手順を示す．C++ に不慣れな読者でも再現できるよう，操作を段階的に整理する．

B.5.1　最小マップ JSON 例

```
{
  "width": 100,
  "height": 80,
  "zones": [
    {"name": "Sea", "type": "sea", "x": 0, "y": 0, "w": 100, "h": 80}
  ],
  "obstacles": [],
  "targets": []
}
```

width, height　マップ全体の寸法 [m]．
zones　ゾーン配列．各要素は名前・種類と位置・大きさを持つ．
obstacles　障害物配列．存在しない場合は空配列とする．
targets　目標地点配列．必要に応じて複数定義できる．

B.5.2　構文の確認

JSON の構文は Python 標準モジュールで即座に確認できる．

```
$ python -m json.tool data/maps/port_map.json
```

エラーが出なければ構文は正しい．

B.5.3　必須項目の検証

MapValidator は以下の観点でマップをチェックし，問題の深刻度ごとに Info/Warning/Error を出力する．代表的な判定項目を列挙する．

- ルートが JSON オブジェクトであること（そうでなければ致命的エラー）．
- zones[] が存在し，各要素がオブジェクトで数値プロパティ x,y,w,h を持つこと．名前や種類が文字列でない場合もエラー扱いとなる．
- obstacles[] が存在する場合は配列であり，polygon 指定は配列，矩形指定では位置・寸法が数値になっていること．
- berths[] が未定義の場合は警告として通知し，旧来の定数ベースロジックへフォールバックする．
- water_types[] が無い場合は情報メッセージを表示して既定値を使用する．
- speed_profiles を定義する際は JSON オブジェクトである必要がある．

致命的エラーが 1 件でもあれば「Simulation stopped because of the error.」と表示されシミュレーションは開始されない．警告のみの場合は「We'll keep going, but please double-check those warnings.」と表示されて継続する。チェック結果は標準出力に色付きで表示されるほか，logs/run.log にも同じ内容が保存されるため，事後にレビューすることも可能である．補助的なスクリプトで静的に確認したい場合は，従来通り Python からキー存在チェックを行っても良いが，実行バイナリだけでも十分に整合性を担保できるようになった．

■ 従来の確認手法

以下の短い Python スクリプトは必要キーの有無を検査する最小例である．

```python
import json, sys

REQUIRED = ["width", "height", "zones", "obstacles", "targets"]
with open(sys.argv[1]) as f:
    data = json.load(f)

missing = [k for k in REQUIRED if k not in data]
if missing:
    raise SystemExit(f"missing: {', '.join(missing)}")
print("OK")
```

使用例を示す．

```
$ python validate_map.py data/maps/port_map.json
OK
```

この手順により，入力ファイルの整合性を事前に確認できる．