Color notation

Written, tested
Written, not tested
Partially written, not tested
Not written, not tested

Actual questions:
-Game mode: real-time or turn-based?

-Engine for visualization? (proposal: Irrlicht)

-packages for data storage?
(proposals: sql database (via sqlite framework), json file(s) (via jsoncpp framework))

update() method pattern here

GameObject

#own_id: size_t
#name: std::string
#description:: std::string

+GameObject()
+~*GameObject()*
+get_own_id(): size_t
+get_name(result: std::string&): size_t
+get_description(result: std::string&): size_t
+set_own_id(value: const size_t&): size_t
+set_name(value: const std::string&): size_t
+set_description(value: const std::string&): size_t
+*what(result: std::string&): size_t*
+*update(): size_t*

Location

#stats: std::vector<size_t>
#id: size_t = 1

+Location()
+~Location()
+get_stats(result: std::vector<size_t>&): size_t
+get_stats(index: const size_t&): size_t
+set_stats(value: const std::vector<size_t>&): size_t
+set_stats(index: const size_t&, value: const size_t&): size_t
+what(result: std::string&): size_t
+update(): size_t

0..* — used as stats indices — 1

enum LocationStats

LS_COVERS
LS_VISIBILITY
LS_SIZE

Type Object pattern here

LevelableObject

#level: size_t

+LevelableObject()
+~*LevelableObject()*
+get_level(): size_t
+set_level(value: const size_t&): size_t

Faction

#influence: size_t
#relationship: size_t

+Faction()
+~*Faction()*
+get_influence(): size_t
+get_relationship(): size_t
+set_influence(value: const size_t&): size_t
+set_relationship(value: const size_t&): size_t
+what(result: std::string&): size_t
+update(): size_t
+increase_influence(amount: const size_t&): size_t
+decrease_influence(amount: const size_t&): size_t
+increase_relationship(amount: const size_t&): size_t
+decrease_relationship(amount: const size_t&): size_t
+create_contract(result: Contract*&): size_t

1..* — offers contracts to / takes and completes contracts for — 1

Contract

#reward: size_t
#employer_faction: size_t
#target_faction: size_t
#phase: size_t
#bonuses: std::vector<size_t>
#mercs: std::vector<Mercenary*> //or their own_ids shall be used here?
#enemies: std::vector<Enemy*>
#location: Location&
#id: size_t = 1

+Contract()
+*Contract()*
+get_reward(): size_t
+get_employer_faction(): size_t
+get_target_faction(): size_t
+get_phase(): size_t
+get_bonuses(result: std::vector<size_t>&): size_t
+get_bonuses(index: const size_t&): size_t
+get_mercs(result: std::vector<Mercenary*>&): size_t
+get_mercs(index: const size_t&, result: Mercenary*&): size_t
+get_enemies(result: std::vector<Enemy*>&): size_t
+get_enemies(index: const size_t&, result: Enemy*&): size_t
+get_location(): Location&
+get_risk(): size_t
+set_reward(value: const size_t&): size_t
+set_employer_faction(value: const size_t&): size_t
+set_target_faction(value: const size_t&): size_t
+set_phase(value: const size_t&): size_t
+set_bonuses(value: const std::vector<size_t>&): size_t
+set_bonuses(index: const size_t&, value: const size_t&): size_t
+set_mercs(value: const std::vector<Mercenary*>&): size_t
+set_mercs(index: const size_t&, value: const Mercenary*&): size_t
+set_enemies(value: const std::vector<Enemy*>&): size_t
+set_enemies(index: const size_t&, value: const Enemy*&): size_t
+set_location(value: const Location&): size_t
+what(result: std::string&): size_t
+update(): size_t
+add_mercenary(new_merc: const Mercenary*&): size_t
+remove_mercenary(index: const size_t&): size_t
+make_offer(summ: const size_t&): size_t
+is_done(): bool

1 — 1 — 1..*

0..*

enum ContractPhases

CF_PENDING
CF_RECRUITING
CF_IN_PROGRESS
CF_SUCCESS
CF_FAILED
CF_SIZE

used as phase values

GameCharacter

#faction_id: size_t

+GameCharacter()
+~*GameCharacter()*
+get_faction_id(): size_t
+set_faction_id(value: const size_t&): size_t

Enemy

#task_stats: std::vector<size_t>
#loot_list: std::vector<std::pair<size_t, size_t> >
#reward: std::vector<size_t>
#id: size_t = 1

+Target()
+~*Target()*
+get_task_stats(result: std::vector<size_t>&): size_t
+get_task_stats(index: const size_t&): size_t
+get_loot_list(result: std::vector<std::pair<size_t, size_t> >&): size_t
+get_loot_list(index: const size_t&, result: std::pair<size_t, size_t>&): size_t
+get_reward(result: std::vector<size_t>&): size_t
+get_reward(index: const size_t&): size_t
+set_task_stats(value: const std::vector<size_t>&): size_t
+set_task_stats(index: const size_t&, value: const size_t&): size_t
+set_loot_list(value: const std::vector<std::pair<size_t, size_t> >&): size_t
+set_loot_list(index: const size_t&, value: const std::pair<size_t, size_t>&): size_t
+set_reward(value: const std::vector<size_t>&): size_t
+set_reward(index: const size_t&, value: const size_t&): size_t
+what(result: std::string&): size_t

Mercenary

#experience: std::vector<size_t>
#task_stats: std::vector<size_t>
#recruitment_stats: std::vector<size_t>
#history: std::vector<size_t>
#equipment: Inventory*
_#id: size_t = 1_--
+Mercenary()
+~*Mercenary()*
#level_up(): size_t
#evaluate_offer(risk: const size_t&, payment: const size_t&): size_t
+get_experience(result: std::vector<size_t>&): size_t
+get_experience(index: const size_t&): size_t
+get_task_stats(result: std::vector<size_t>&): size_t
+get_task_stats(index: const size_t&): size_t
+get_recruitment_stats(result: std::vector<size_t>&): size_t
+get_recruitment_stats(index: const size_t&): size_t
+get_history(result: std::vector<size_t>&): size_t
+get_history(index: const size_t&): size_t
+get_equipment(): Inventory&
+set_experience(value: const std::vector<size_t>&): size_t
+set_experience(index: const size_t&, value: const size_t&): size_t
+set_task_stats(value: const std::vector<size_t>&): size_t
+set_task_stats(index: const size_t&, value: const size_t&): size_t
+set_recruitment_stats(value: const std::vector<size_t>&): size_t
+set_recruitment_stats(index: const size_t&, value: const size_t&): size_t
+set_history(value: const std::vector<size_t>&): size_t
+set_history(index: const size_t&, value: const size_t&): size_t
+set_equipment(value: Inventory*): size_t
+what(result: std::string&): size_t
+update(): size_t

1..* — 0..*

enum Inventory_slots

IS_HEAD
IS_ARMOUR
IS_GLOVES
IS_BOOTS
IS_PRIMARY_WEAPON
IS_SECONDARY_WEAPON
IS_SIZE

used as slots indices

used as equipped indices

Inventory

#equipped: std::vector<Item*>

+Inventory()
+~Inventory()
+get_equipped(result: std::vector<Item*>&): size_t
+get_equipped(index: const size_t&, result: Item*&): size_t
+get_bonuses(result: std::vector<size_t>&): size_t
+get_bonuses(index: const size_t&): size_t
+set_equipped(value: const: std::vector<Item*>&): size_t
+set_equipped(index: const size_t&, value: Item*): size_t
+what(result: std::string&): size_t
+update(): size_t

1 — 1 — 1..*

Item

#owner_id: size_t
#bonuses: std::vector<size_t>
#slots: std::vector<bool>
#kind: size_t
#rarity: size_t
#price: size_t
#id: size_t = 1

+Item()
+~*Item()*
+get_owner_id(): size_t
+get_bonuses(result: std::vector<size_t>&): size_t
+get_bonuses(index: const size_t&): size_t
+get_slots(result: std::vector<bool>&): size_t
+get_slots(index: const size_t&): bool
+get_kind(): size_t
+get_rarity(): size_t
+get_price(): size_t
+set_owner_id(value: const size_t&): size_t
+set_bonuses(value: const std::vector<size_t>&): size_t
+set_bonuses(index: const size_t&, value: const size_t&): size_t
+set_slots(value: const std::vector<bool>&): size_t
+set_slots(index: const size_t&, value: const bool&): size_t
+set_kind(value: const size_t&): size_t
+set_rarity(value: const size_t&): size_t
+set_price(value: const size_t&): size_t
+what(result: std::string&): size_t
+update(): size_t

1..*

Player

#cash: size_t
#history: std::vector<size_t>
#renown: std::vector<size_t>
#id: size_t = 1

+Player()
+~*Player()*
#level_up(): size_t
+get_cash(): size_t
+get_history(result: std::vector<size_t>&): size_t
+get_history(index: const size_t&): size_t
+get_renown(result: std::vector<size_t>&): size_t
+get_renown(index: const size_t&): size_t
+set_cash(value: const size_t&): size_t
+set_history(value: const std::vector<size_t>&): size_t
+set_history(index: const size_t&, value:const size_t&): size_t
+set_renown(value: const std::vector<size_t>&): size_t
+set_renown(index: const size_t&, value: const size_t&): size_t
+what(result: std::string&): size_t
+update(): size_t
+add_cash(amount: const size_t&): size_t
+spend_cash(amount: const size_t&): size_t
+add_history(index: const size_t&): size_t
+add_history(index: const size_t&, amount: const size_t&): size_t
+add_renown(amount: const size_t&): size_t

buys and sells items — 1

Vendor

#faction_id: size_t
#assortment: std::vector<Item*>
#id: size_t = 1

+Vendor()
+~*Vendor()*
#form_assortment(): size_t
+get_faction_id(): size_t
+get_assortment(result: std::vector<Item*>&):size_t
+get_assortment(index: const size_t&, result: Item*&): size_t
+set_faction_id(value: const size_t&): size_t
+set_assortment(value: const std::vector<Item*>&): size_t
+set_assortment(index: const size_t&, value: const Item*&): size_t
+what(result: std::string&): size_t
+update(): size_t
+get_price_modifier(): size_t

1..*

enum TaskStats

TS_STEALTH
TS_RANGED
TS_SAPPER
TS_MELEE
TS_HEAVY
TS_SIZE

used as task_stats indices

used as bonuses indices

enum RecruitmentStats

RS_GREED
RS_CAUTION
RS_INVENTIVE
RS_ACCURACY
RS_EDUCABILITY
RS_SIZE

used as recruitment_stats indices

enum MercHistory

MH_TASKS_COMPLETED
MH_TASKS_FAILED
MH_CASH_EARNED
MH_ENEMIES_KILLED
MH_SIZE

used as history indices

evaluate_offer() return values

enum MercAnswer

MA_OK
MA_TOO_BIG_RISK
MA_WANT_MORE_MONEY
MA_SIZE

Command pattern here

Event

#world: World&

+Event(world: World&)
+~*Event()*
+*happen(): size_t*

1

Derived classes hierarchy here to realise various events, including user input handling.

«friend»

Not the best decision, other ways to provide Event's derivatives with access to World's protected fields?

QueueNode

#id: size_t = 1
#own_id: size_t
#priority: size_t
#event: Event*

+QueueNode(priority: const size_t&, event: const Event*&)
+~QueueNode()
+get_own_id(): size_t
+get_priority(): size_t
+set_priority(value: const size_t&): size_t
+process(): size_t

1 — 0..*

EventsQueue

#events: std::vector<QueueNode>

+EventsQueue()
+~EventsQueue()
#heapify(): size_t
#pop(): size_t
+add_event(priority: const size_t&, event: const Event*&)
+remove_event(id: const size_t&): size_t
+update(): size_t

1 — 1

changes state

World

#player: Player*
#locations: std::vector<Location*>
#factions: std::vector<Faction*>
#contracts: std::vector<Contract*>
#mercs: std::vector<Mercenary*>
#vendors: std::vector<Vendor*>
#time: size_t

+World()
+~World()
+game_loop(): size_t
+get_player_info(result: std::string&):size_t
+get_faction_info(index: const size_t&, result: std::string&): size_t
+get_contract_info(index: const size_t&, result: std::string&): size_t
+get_vendor_info(index: const size_t&, result: std::string&): size_t
+get_time(result: std::string&): size_t