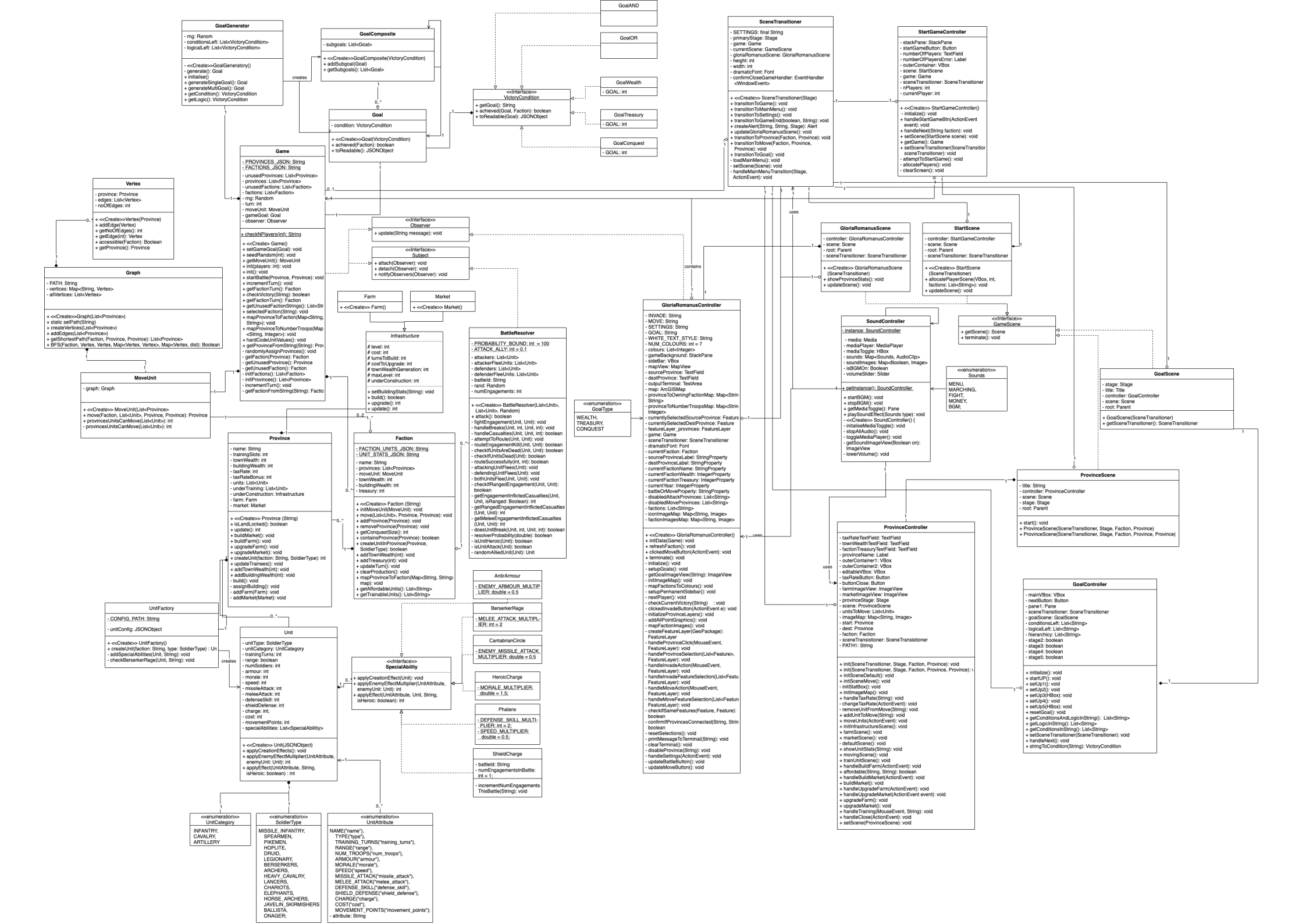
GoalAND
GoalOR
GoalGenerator
- rng: Ranom
- conditionsLeft: List<VictoryCondition>
- logicalLeft: List<VictoryCondition>
- <<Create>>GoalGenerator()
- generate(): Goal
+ initialise()
+ generateSingleGoal(): Goal
+ generateMultiGoal(): Goal
+ getCondition(): VictoryCondition
+ getLogic(): VictoryCondition
GoalComposite
- subgoals: List<Goal>
+ <<Create>>GoalComposite(VictoryCondition)
+ addSubgoal(Goal)
+ getSubgoals(): List<Goal>
creates
Goal
- condition: VictoryCondition
+ <<Create>>Goal(VictoryCondition)
+ achieved(Faction): boolean
+ toReadable(): JSONObject
<<Interface>>
VictoryCondition
+ getGoal(): String
+ achieved(Goal, Faction): boolean
+ toReadable(Goal): JSONObject
GoalWealth
- GOAL: int
GoalTreasury
- GOAL: int
GoalConquest
- GOAL: int
SceneTransitioner
- SETTINGS: final String
- primaryStage: Stage
- game: Game
- currentScene: GameScene
- gloriaRomanusScene: GloriaRomanusScene
- height: int
- width: int
- dramaticFont: Font
- confirmCloseGameHandler: EventHandler
<WindowEvent>
+ <<Create>> SceneTransitioner(Stage)
+ transitionToGame(): void
+ transitionToMainMenu(): void
+ transitionToSettings(): void
+ transitionToGameEnd(boolean, String): void
+ createAlert(String, String, Stage): Alert
+ updateGloriaRomanusScene(): void
+ transitionToProvince(Faction, Province): void
+ transitionToMove(Faction, Province, Province): void
+ transitionToGoal(): void
- loadMainMenu(): void
- setScene(Scene): void
- handleMainMenuTransition(Stage, ActionEvent): void
StartGameController
- stackPane: StackPane
- startGameButton: Button
- numberOfPlayers: TextField
- numberOfPlayersError: Label
- outerContainer: VBox
- scene: StartScene
- game: Game
- sceneTransitioner: SceneTransitioner
- nPlayers: int
- currentPlayer: int
+ <<Create>> StartGameController()
- initialize(): void
+ handleStartGameBtn(ActionEvent event): void
+ handleNext(String faction): void
+ setScene(StartScene scene): void
+ getGame(): Game
+ setSceneTransitioner(SceneTransitior sceneTransitioner): void
- attemptToStartGame(): void
- allocatePlayers(): void
- clearScreen(): void
Vertex
- province: Province
- edges: List<Vertex>
- noOfEdges: int
+ <<Create>>Vertex(Province)
+ addEdge(Vertex)
+ getNoOfEdges(): int
+ getEdge(int): Vertex
+ accessible(Faction): Boolean
+ getProvince(): Province
Game
- PROVINCES_JSON: String
- FACTIONS_JSON: String
- unusedProvinces: List<Province>
- provinces: List<Province>
- unusedFactions: List<Faction>
- factions: List<Faction>
- rng: Random
- turn: int
- moveUnit: MoveUnit
- gameGoal: Goal
- observer: Observer
+ checkNPlayers(int): String
+ <<Create> Game()
+ setGameGoal(Goal): void
+ seedRandom(int): void
+ getMoveUnit(): MoveUnit
+ init(players: int): void
+ init(): void
+ startBattle(Province, Province): void
+ increment(Turn): void
+ getFactionTurn(): Faction
+ checkVictory(String): boolean
+ getFactionTurn(): Faction
+ getUnusedFactionStrings(): List<Str
+ selectedFaction(String): void
+ mapProvinceToFaction(Map<String, String>): void
+ mapProvinceToNumberTroops(Map <String, Integer>): void
+ hardCodeUnitValues(): void
+ getProvinceFromString(String): Prov
- randomlyAssignProvinces(): void
- getFaction(Province): Faction
- getUnusedProvince(): Province
- getUnusedFaction(): Faction
- initFactions(): List<Faction>
- initProvinces(): List<Province>
- incrementTurn(): void
- getFactionFromString(String): Factic
<<Interface>>
Observer
+ update(String message): void
<<Interface>>
Subject
+ attach(Observer): void
+ detach(Observer): void
+ notifyObservers(Observer): void
contains
uses
GloriaRomanusScene
- controller: GloriaRomanusController
- scene: Scene
- root: Parent
- sceneTransitioner: SceneTransitioner
+ <<Create>> GloriaRomanusScene (SceneTransitioner)
+ showProvinceStats(): void
+ updateScene(): void
StartScene
- controller: StartGameController
- scene: Scene
- root: Parent
- sceneTransitioner: SceneTransitioner
+ <<Create>> StartScene (SceneTransitioner)
+ allocatePlayerScene(VBox, int, factions: List<String>): void
+ updateScene(): void
Graph
- PATH: String
- vertices: Map<String, Vertex>
- allVertices: List<Vertex>
+ <<Create>>Graph(List<Province>)
+ static setPath(String)
+ createVertices(List<Province>)
+ addEdges(List<Province>)
+ getShortestPath(Faction, Province, Province): List<Province>
+ BFS(Faction, Vertex, Vertex, Map<Vertex, Vertex>, Map<Vertex, dist): Boolean
Farm
+ <<Create>> Farm()
Market
+ <<Create>> Market()
Infrastructure
level: int
cost: int
turnsToBuild: int
costToUpgrade: int
townWealthGeneration: int
maxLevel: int
underConstruction: int
+ setBuildingStats(String): void
+ build(): boolean
+ upgrade(): int
+ update(): int
BattleResolver
- PROBABILITY_BOUND: int = 100
- ATTACK_ALLY: int = 0.1
- attackers: List<Unit>
- attackerFleeUnits: List<Unit>
- defenders: List<Unit>
- defenderFleeUnits: List<Unit>
- battleId: String
- rand: Random
- numEngagements: int
+ <<Create>> BattleResolver(List<Unit>, List<Unit>, Random)
+ attack(): boolean
- fightEngagement(Unit, Unit): void
- handleBreaks(Unit, int, Unit, int): void
- handleCasualties(Unit, Unit, int): boolean
- attemptToRoute(Unit, Unit): void
- routeEngagementKill(Unit, Unit): boolean
- checkIfUnitsAreDead(Unit, Unit): boolean
- checkIfUnitsDead(Unit): boolean
- routeSuccessfully(int, int): boolean
- attackingUnitFlees(Unit): void
- defendingUnitFlees(Unit): void
- bothUnitsFlee(Unit, Unit): void
- checkIfRangedEngagement(Unit, Unit): boolean
- getEngagementInflictedCasualties(Unit, Unit, isRanged: Boolean): int
- getRangedEngagementInflictedCasualties (Unit, Unit): int
- getMeleeEngagementInflictedCasualties (Unit, Unit): int
- doesUnitBreak(Unit, int, Unit, int): boolean
- resolverProbability(double): boolean
- isUnitHeroic(Unit): boolean
- isUnitAttack(Unit): boolean
- randomAlliedUnit(Unit): Unit
<<enumeration>>
GoalType
WEALTH,
TREASURY,
CONQUEST
GloriaRomanusController
- INVADE: String
- MOVE: String
- SETTINGS: String
- GOAL: String
- WHITE_TEXT_STYLE: String
- NUM_COLOURS: int = 7
- colours: List<Integer>
- gameBackground: StackPane
- sideBar: VBox
- mapView: MapView
- sourceProvince: TextField
- destProvince: TextField
- outputTerminal: TextArea
- map: ArcGISMap
- provinceToOwningFactionMap: Map<Strin String>
- provinceToNumberTroopsMap: Map<Strin Integer>
- currentlySelectedSourceProvince: Feature
- currentlySelectedDestProvince: Feature
- featureLayer_provinces: FeatureLayer
- game: Game
- sceneTransitioner: SceneTransitioner
- dramaticFont: Font
- currentFaction: Faction
- sourceProvinceLabel: StringProperty
- destProvinceLabel: StringProperty
- currentFactionName: StringProperty
- currentFactionWealth: IntegerProperty
- currentFactionTreasury: IntegerProperty
- currentYear: IntegerProperty
- battleOrMoveProperty: StringProperty
- disabledAttackProvinces: List<String>
- disabledMoveProvinces: List<String>
- factions: List<String>
- iconImageMap: Map<String, Image>
- factionImagesMap: Map<String, Image>
+ <<Create>> GloriaRomanusController()
+ initData(Game): void
+ refreshFaction(): void
+ clickedMoveButton(ActionEvent): void
+ terminate(): void
- initialize(): void
- setupGoals(): void
- getGoalImageView(String): ImageView
- initImageMap(): void
- mapFactionsToColours(): void
- setupPermanentSidebar(): void
- nextPlayer(): void
- checkCurrentVictory(String) : void
- clickedInvadeButton(ActionEvent e): void
- initializeProvinceLayers(): void
- addAllPointGraphics(): void
- mapFactionImages(): void
- createFeatureLayer(GeoPackage): FeatureLayer
- handleProvinceClick(MouseEvent, FeatureLayer): void
- handleProvinceSelection(List<Feature>, FeatureLayer): void
- handleInvadeAction(MouseEvent, FeatureLayer): void
- handleInvadeFeatureSelection(List<Featu FeatureLayer): void
- handleMoveAction(MouseEvent, FeatureLayer): void
- handleMoveFeatureSelection(List<Featur FeatureLayer): void
- checkIfSameFeatures(Feature, Feature): boolean
- confirmIfProvincesConnected(String, Strin boolean
- resetSelections(): void
- printMessageToTerminal(String): void
- clearTerminal(): void
- disableProvince(String): void
- handleSettings(ActionEvent): void
- updateBattleButton(): void
- updateMoveButton(): void
SoundController
- instance: SoundController
- media: Media
- mediaPlayer: MediaPlayer
- mediaToggle: HBox
- sounds: Map<Sounds, AudioClip>
- soundImages: Map<Boolean, Image>
- isBGMOn: Boolean
- volumeSlider: Slider
+ getInstance(): SoundController
+ startBGM(): void
+ stopBGM(): void
+ getMediaToggle(): Pane
+ playSoundEffect(Sounds type): void
- <<Create>> SoundController() {
- initaliseMediaToggle(): void
- stopAllAudio(): void
- toggleMediaPlayer(): void
- getSoundImageView(Boolean on): ImageView
- lowerVolume(): void
<<Interface>>
GameScene
+ getScene(): Scene
+ terminate(): void
<<enumeration>>
Sounds
MENU,
MARCHING,
FIGHT,
MONEY,
BGM;
GoalScene
- stage: Stage
- title: Title
- controller: GoalController
- scene: Scene
- root: Parent
+ GoalScene(SceneTransitioner)
+ getSceneTransitioner(): SceneTransitioner
MoveUnit
- graph: Graph
+ <<Create>> MoveUnit(List<Province>
+ move(Faction, List<Unit>, Province, Province): Province
+ provincesUnitsCanMove(List<Unit>): int
- provincesUnitsCanMove(List<Unit>): int
Province
- name: String
- trainingSlots: int
- townWealth: int
- buildingWealth: int
- taxRate: int
- taxRateBonus: int
- units: List<Unit>
- underTraining: List<Unit>
- underConstruction: Infrastructure
- farm: Farm
- market: Market
+ <<Create>> Province (String)
+ isLandLocked(): boolean
+ update(): int
+ buildMarket(): void
+ buildFarm(): void
+ upgradeFarm(): void
+ upgradeMarket(): void
+ createUnit(faction: String, SoldierType): int
+ updateTrainees(): void
+ addTownWealth(int): void
+ addBuildingWealth(int): void
- build(): void
- assignBuilding(): void
- addFarm(Farm): void
- addMarket(Market): void
Faction
- FACTION_UNITS_JSON: String
- UNIT_STATS_JSON: String
- name: String
- provinces: List<Province>
- moveUnit: MoveUnit
- townWealth: int
- buildingWealth: int
- treasury: int
+ <<Create>> Faction (String)
+ initMoveUnit(MoveUnit): void
+ move(List<Unit>, Province, Province): void
+ addProvince(Province): void
+ removeProvince(Province): void
+ getConquestSize(): int
+ containsProvince(Province): boolean
+ createUnitInProvince(Province, SoldierType): boolean
+ addTownWealth(int): void
+ addTreasury(int): void
+ updateTurn(): void
+ clearProduction(): void
+ mapProvinceToFaction(Map<String, String> map): void
+ getAffordableUnits(): List<String>
+ getTrainableUnits(): List<String>
ProvinceScene
- title: String
- controller: ProvinceController
- scene: Scene
- stage: Stage
- root: Parent
+ start(): void
+ ProvinceScene(SceneTransitioner, Stage, Faction, Province)
+ ProvinceScene(SceneTransitioner, Stage, Faction, Province, Province)
ProvinceController
- taxRateTextField: TextField
- townWealthTextField: TextField
- factionTreasuryTextField: TextField
- provinceName: Label
- outerContainer1: VBox
- outerContainer2: VBox
- editableVBox: VBox
- taxRateButton: Button
- buttonClose: Button
- farmImageView: ImageView
- marketImageView: ImageView
- provinceStage: Stage
- scene: ProvinceScene
- unitsToMove: List<Unit>
- imageMap: Map<String, Image>
- start: Province
- dest: Province
- faction: Faction
- sceneTransistioner: SceneTransistioner
- PATH1: String
+ init(SceneTransitioner, Stage, Faction, Province): void
+ init(SceneTransitioner, Stage, Faction, Province, Province): \
+ initSceneDefault(): void
+ initSceneMove(): void
- initStatBox(): void
+ initImageMap(): void
+ handleTaxRate(String): void
- changeTaxRate(ActionEvent): void
- removeUnitFromMove(String): void
+ addUnitToMove(String): void
+ moveUnits(ActionEvent): void
+ initInfrastructureScene(): void
+ farmScene(): void
+ marketScene(): void
- defaultScene(): void
+ showUnitStats(String): void
+ movingScene(): void
+ trainUnitScene(): void
+ handleBuildFarm(ActionEvent): void
+ affordable(String, String): boolean
+ handleBuildMarket(ActionEvent): void
+ buildMarket(): void
+ handleUpgradeFarm(ActionEvent): void
+ handleUpgradeMarket(ActionEvent event): void
+ upgradeFarm(): void
+ upgradeMarket(): void
+ handleTraining(MouseEvent, String): void
+ handleClose(ActionEvent): void
+ setScene(ProvinceScene): void
GoalController
- mainVBox: VBox
- nextButton: Button
- pane1: Pane
- sceneTransitioner: SceneTransitioner
- goalScene: GoalScene
- conditionsLeft: List<String>
- logicalLeft: List<String>
- hierarchicy: List<String>
- stage2: boolean
- stage3: boolean
- stage4: boolean
- stage5: boolean
+ initialize(): void
+ startUP(): void
+ setUp1(): void
+ setUp2(): void
+ setUp3(HBox): void
+ setUp4(): void
+ setUp5(HBox): void
+ resetGoal(): void
+ getConditionsAndLogicInString(): List<String>
+ getLogicInString(): List<String>
+ getConditionsInString(): List<String>
+ setSceneTransitioner(SceneTransitioner): void
+ handleNext(): void
+ stringToCondition(String): VictoryCondition
UnitFactory
- CONFIG_PATH: String
- unitConfig: JSONObject
+ <<Create>> UnitFactory()
+ createUnit(faction: String, type: SoldierType) : Un
- addSpecialAbilities(Unit, String): void
- checkBerserkerRage(Unit, String): void
creates
Unit
- unitType: SoldierType
- unitCategory: UnitCategory
- trainingTurns: int
- range: boolean
- numSoldiers: int
- armour: int
- morale: int
- speed: int
- missileAttack: int
- meleeAttack: int
- defenseSkill: int
- shieldDefense: int
- charge: int;
- cost: int
- movementPoints: int
- specialAbilities: List<SpecialAbility>
+ <<Create>> Unit(JSONObject)
+ applyCreationEffects(): void
+ applyEnemyEffectMultiplier(UnitAttribute, enemyUnit: Unit): int
+ applyEffect(UnitAttribute, String, isHeroic: boolean) : int
<<Interface>>
SpecialAbility
+ applyCreationEffect(Unit): void
+ applyEnemyEffectMultiplier(UnitAttribute, enemyUnit: Unit): int
+ applyEffect(UnitAttribute, Unit, String, isHeroic: boolean): int
AntiArmour
- ENEMY_ARMOUR_MULTIPLIER: double = 0.5
BerserkerRage
- MELEE_ATTACK_MULTIPLIER: int = 2
CantabrianCircle
- ENEMY_MISSILE_ATTACK_MULTIPLIER: double = 0.5
HeroicCharge
- MORALE_MULTIPLIER: double = 1.5;
Phalanx
- DEFENSE_SKILL_MULTIPLIER: int = 2;
- SPEED_MULTIPLIER: double = 0.5;
ShieldCharge
- battleId: String
- numEngagementsInBattle: int = 1;
- incrementNumEngagementsThisBattle(String): void
<<enumeration>>
UnitCategory
INFANTRY,
CAVALRY,
ARTILLERY
<<enumeration>>
SoldierType
MISSILE_INFANTRY,
SPEARMEN,
PIKEMEN,
HOPLITE,
DRUID,
LEGIONARY,
BERSERKERS,
ARCHERS,
HEAVY_CAVALRY,
LANCERS,
CHARIOTS,
ELEPHANTS,
HORSE_ARCHERS,
JAVELIN_SKIRMISHERS
BALLISTA,
ONAGER;
<<enumeration>>
UnitAttribute
NAME("name"),
TYPE("type"),
TRAINING_TURNS("training_turns"),
RANGE("range"),
NUM_TROOPS("num_troops"),
ARMOUR("armour"),
MORALE("morale"),
SPEED("speed"),
MISSILE_ATTACK("missile_attack"),
MELEE_ATTACK("melee_attack"),
DEFENSE_SKILL("defense_skill"),
SHIELD_DEFENSE("shield_defense"),
CHARGE("charge"),
COST("cost"),
MOVEMENT_POINTS("movement_points");
- attribute: String