## **FARMULATOR** quality: int - CalculateQuality(): int Javier Sepulveda Pesticida Fertilizante <<abstract>> Block Alimento para animal Herbicida # workable: bool Riego Fungicida Vacuna Agua para animal - blocks: Block[,] - build: Build Market + Construction(Build build): void MenuManager pricesProducts: List<PriceProduct> PriceTerrain <<abstract>> + Destroy(): void - static game: Game Consumable - pricesConsumables: List<PriceConsumable> → - terrain: Terrain + GenerateBlocks(): void + static StartMenu(): void name: string - pricesTerrains: List<PriceTerrain> - price: int + CreateFinalProdcut(Storage storage, int quality): void - static NewGameMenu(): void - description: string + PriceMarketProduct(List<Product> products): void - static LoadGameMenu(): void $+ \ Price Market Consumable (List < Consumable > consumables): \ void$ - static OptionsMenu(): void + PriceMarketTerrain(Map map): void PriceConsumable - static GameMenu(): bool 1 + CalculatePricesProducts(int turn): void - consumable: Consumable terrains: List<Terrain> - static MarketMenu(): bool - price: int - consumables: List<Consumables> - static FarmMenu(): bool + AddConsumables(Consumable consumable, int quantity): void Game + ApplyConsumable(Consumable consumable, Build build): void - turn: int + AddTerrain(Terrain terrain): void - money: int + GenerateFarm(Terrain[,] terrainsMap): void PriceProduct - map: Map - product: Product - market: Market initialPrice: int - creationDate: DateTime sellPrice: int + static RenderMenu(List<string> options, string title, bool subMenu): int - terrains: Terrain[,] - saveDate: DateTime maxPriceVariation: int + static RenderMarket(Game game, int typeBuy, int buy): bool - builds: List<Build> - river: River - priceHistory: List<int> + static RenderMap(Map map): bool - lake: Lake products: List<Product> + AddPrice(int price): void - static TextCenter(string text): string - farm: Farm - consumables: List<Consumable> $+\ static\ Render Farm Management (Game\ game,\ int\ type Management Build,\ int\ type Management):\ bool$ + static RenderFarmDetails(Map map): void - GenerateTerrains(): void + SaveGame(): bool + static RenderSaveMenu(Game game): bool ResetMap(): void + DeleteGame(): bool + static RenderLoadMenu(Game game): string # name: string + NextTurn(): void - InsertAssets(List<int[,]> positions, bool direction = true): void # waterConsumption: int + ConstructionBuilding(Terrain terrain, Build build, int cost): void 0..1 # minWater: int + ConstructionSell(Terrain terrain, int cost): void # waterPenalty: int + BuyConsumables(List<PriceConsumable> consumables, int quantity, int totalValue): void # timeProduction: int + BuyTerrain(Terrian terrain, int value): void # diseaseProbability: int River + SellFinalProduct(Terrian terrain, Storage storage, int price, int quality): void - positions: List<int[]> # diseasePenalty: int - positions: List<int[]> + SellFinalProductStorage(Storage storage, FinalProduct finalProduct, int price): void - direction: bool + GenerateLake(): void FinalProduct + GenerateRiver(): void <<abstract>> Animal Build - maxCapacity: int - product: Product foodConsumption: int - nutrientsConsumption: int # name: string - finalProducts: List<FinalProduct> quality: int - minFood: int - minNutrients: int # buyPrice: int + qualityDecline(): void + QualityDecline(): void - foodPenalty: int - nutrientsPenalty: int # sellPrice: int + AddFinalProduct(Build build, int quality): void escapeProbability: int wormsProbability: int + SellFinalProduct(FinalProduct finalProduct): void Ranch escapeRange: int[] wormsPenalty: int - animal: Animal deadProbability: int - undergrowthProbability: int <<abstract>> Production - food: int deadRange: int[] - undergrowthPenalty: int - units: int - priceVariation: int + FoodWaterConsumption(type): type # health: int + ToMature(): void # water: int + HealthPenalty(): void # maturity: int + ApplyConsumable(Consumable consumable): void # finalProduction: double + DiseaseProbability(): void # disease: bool - seed: Seed - undergrowth: bool worms: bool nutrients: int + FoodWaterConsumption(type): type + ToMature(): void + HealthPenalty(): void + DiseaseProbability(): void + ApplyConsumable(Consumable consumable): void