FARMULATOR quality: int - CalculateQuality(): int Javier Sepulveda Fertilizante Pesticida <<abstract>> Block Alimento para animal Herbicida # workable: bool Riego Fungicida Vacuna Agua para animal Terrain - blocks: Block[,] 0..1 - build: Build <<static>> MenuManager Market - pricesProducts: List<PriceProduct> + Construction(Build build): void PriceTerrain <<abstract>> - static game: Game Consumable - pricesConsumables: List<PriceConsumable> + Destroy(): void - terrain: Terrain - pricesTerrains: List<PriceTerrain> + static StartMenu(): void - name: string + GenerateBlocks(): void - price: int - static NewGameMenu(): void - description: string + PriceMarketProduct(List<Product> products): void - static LoadGameMenu(): void + PriceMarketConsumable(List<Consumable> consumables): void - static OptionsMenu(): void + PriceMarketTerrain(Map map): void PriceConsumable - static GameMenu(): void 1 + CalculatePricesProducts(int turn): void - consumable: Consumable terrains: List<Terrain> - static MarketMenu(): void - price: int - consumables: List<Consumables> + AddConsumables(Consumable consumable, int quantity): void Game + AddTerrain(Terrain terrain): void - turn: int + GenerateFarm(Terrain[,] terrainsMap): void - money: int PriceProduct - map: Map - product: Product - market: Market <<static>> Print 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> - GenerateTerrains(): void + SaveGame(): bool <<abstract>> Product + LoadGame(): bool - ResetMap(): void # name: string + DeleteGame(): bool - InsertAssets(List<int[,]> positions, bool direction = true): void # waterConsumption: int 0..1 + NextTurn(): void # minWater: int + ConstructionBuilding(Terrain terrain, Build build, int cost): void # waterPenalty: int + ConstructionSell(Terrain terrain, int cost): void # timeProduction: int + BuyConsumables(List<PriceConsumable> consumables, int quantity, int totalValue): void # diseaseProbability: int River + BuyTerrain(Terrian terrain, int value): void # diseasePenalty: int - positions: List<int[]> positions: List<int[]> - direction: bool + GenerateLake(): void + GenerateRiver(): void <<abstract>> Storage FinalProduct Animal Build maxCapacity: int - product: Product - foodConsumption: int - nutrientsConsumption: int # name: string - minFood: int - finalProducts: List<FinalProduct> - quality: int - minNutrients: int # buyPrice: int foodPenalty: int - nutrientsPenalty: int + qualityDecline(): void # sellPrice: int escapeProbability: int wormsProbability: int Ranch escapeRange: int[] wormsPenalty: int - animal: Animal deadProbability: int - undergrowthProbability: int <<abstract>> - food: int Production - deadRange: int[] - undergrowthPenalty: int # health: int - units: int - priceVariation: int # water: int # maturity: int Land # finalProduction: double - seed: Seed # disease: bool - nutrients: int - worms: bool - undergrowth: bool