// ================================

// PLAYER BIBLE SYSTEM

// ================================

// Purpose: Core player data management system that stores all player

// information, progression, stats, inventory, and integrates with

// all other game systems.

// ===== PLAYER BIBLE CLASS =====

class PlayerBible {

constructor(playerId) {

this.playerId = playerId;

this.data = this.initializePlayerData();

this.autosaveInterval = null;

}

// ====================

// INITIALIZATION

// ====================

initializePlayerData() {

return {

// Core Identity

player\_id: this.playerId,

username: '',

rank: 1,

created\_at: new Date().toISOString(),

last\_login: new Date().toISOString(),

// Stats (All start at 10)

stats: {

str: 10, // Strength

agi: 10, // Agility

int: 10, // Intelligence

tac: 10, // Tactical

med: 10, // Medical

eng: 10, // Engineering

ldr: 10, // Leadership

morale: 10 // Morale

},

// Stat Training Tracking

training: {

last\_trained: {

str: null,

agi: null,

int: null,

tac: null,

med: null,

eng: null,

ldr: null,

morale: null

},

tokens\_available: 0,

tokens\_used\_today: {}

},

// Jobs & Specializations

jobs: ['recruit'], // Starting job

active\_job: 'recruit',

specialties: [],

// Economy

credits: 1000, // Starting credits

tokens: 0, // Premium currency

renown: 0,

fear: 0,

// Inventory

inventory: {

gear\_pieces: [],

completed\_weapons: [{

weapon\_id: 'starter\_rifle',

name: 'M4A1 Starter',

type: 'assault\_rifle',

rarity: 'common',

stats: { accuracy: 50, damage: 30, fire\_rate: 600, range: 300 }

}],

gadgets: [],

consumables: {

smoke\_grenade: 3,

flashbang: 3,

medkit: 2,

adrenaline\_shot: 1

},

training\_tokens: 0,

blueprint\_packs: []

},

// Real Estate

real\_estate: [],

// Platoon

platoon\_id: null,

platoon\_role: null,

// Status

status: {

location: 'base',

hospital\_until: null,

jail\_until: null,

current\_mission: null

},

// Progression

progression: {

level: 1,

experience: 0,

experience\_to\_next\_level: 1000,

missions\_completed: 0,

missions\_failed: 0,

pvp\_wins: 0,

pvp\_losses: 0,

total\_playtime: 0,

gear\_pieces\_collected: 0,

weapons\_crafted: 0,

kills: 0,

deaths: 0,

headshots: 0,

stealth\_kills: 0,

civilians\_saved: 0

},

// Session History

session\_history: [],

// Achievements

achievements: [],

// Settings

settings: {

sfx\_volume: 0.7,

music\_volume: 0.5,

difficulty\_preference: 'normal',

tutorial\_completed: false

}

};

}

// ====================

// STAT MANAGEMENT

// ====================

getStat(statName) {

return this.data.stats[statName] || 0;

}

increaseStat(statName, amount) {

if (!this.data.stats.hasOwnProperty(statName)) {

console.error(`Invalid stat: ${statName}`);

return false;

}

const oldValue = this.data.stats[statName];

this.data.stats[statName] = Math.min(100, oldValue + amount); // Cap at 100

console.log(`${statName.toUpperCase()} increased: ${oldValue} → ${this.data.stats[statName]}`);

this.save();

return true;

}

canTrainStat(statName) {

const lastTrained = this.data.training.last\_trained[statName];

if (!lastTrained) return true;

const now = new Date();

const lastTrainedDate = new Date(lastTrained);

const hoursSince = (now - lastTrainedDate) / (1000 \* 60 \* 60);

return hoursSince >= 24; // Can train once per 24 hours

}

trainStat(statName, useToken = false) {

if (!this.canTrainStat(statName) && !useToken) {

console.warn(`Cannot train ${statName} yet. Wait 24 hours or use a Training Token.`);

return false;

}

if (useToken) {

if (this.data.training.tokens\_available <= 0) {

console.error('No training tokens available');

return false;

}

this.data.training.tokens\_available--;

this.data.training.tokens\_used\_today[statName] =

(this.data.training.tokens\_used\_today[statName] || 0) + 1;

}

// Training increases stat by 1-3 points based on current level

const currentStat = this.getStat(statName);

const increase = currentStat < 50 ? 3 : currentStat < 75 ? 2 : 1;

this.increaseStat(statName, increase);

this.data.training.last\_trained[statName] = new Date().toISOString();

console.log(`Training complete: ${statName} +${increase}`);

this.save();

return true;

}

// ====================

// ECONOMY

// ====================

addCredits(amount, source = 'unknown') {

this.data.credits += amount;

console.log(`Credits added: +${amount} (${source}). Total: ${this.data.credits}`);

this.save();

}

spendCredits(amount, purpose = 'unknown') {

if (this.data.credits < amount) {

console.error(`Insufficient credits. Need ${amount}, have ${this.data.credits}`);

return false;

}

this.data.credits -= amount;

console.log(`Credits spent: -${amount} (${purpose}). Remaining: ${this.data.credits}`);

this.save();

return true;

}

addTokens(amount) {

this.data.tokens += amount;

this.save();

}

spendTokens(amount, purpose = 'unknown') {

if (this.data.tokens < amount) {

console.error(`Insufficient tokens. Need ${amount}, have ${this.data.tokens}`);

return false;

}

this.data.tokens -= amount;

console.log(`Tokens spent: -${amount} (${purpose})`);

this.save();

return true;

}

// ====================

// INVENTORY MANAGEMENT

// ====================

addGearPiece(piece) {

this.data.inventory.gear\_pieces.push(piece);

this.data.progression.gear\_pieces\_collected++;

console.log(`Gear piece added: ${piece.piece\_type} for ${piece.weapon\_family}`);

this.save();

}

removeGearPiece(pieceId) {

const index = this.data.inventory.gear\_pieces.findIndex(p => p.piece\_id === pieceId);

if (index !== -1) {

this.data.inventory.gear\_pieces.splice(index, 1);

this.save();

return true;

}

return false;

}

canCraftWeapon(weaponFamily) {

// Check if player has all required pieces

const requiredPieces = ['barrel', 'receiver', 'stock', 'magazine'];

const playerPieces = this.data.inventory.gear\_pieces.filter(

p => p.weapon\_family === weaponFamily

);

for (const pieceType of requiredPieces) {

if (!playerPieces.find(p => p.piece\_type === pieceType)) {

return false;

}

}

return true;

}

craftWeapon(weaponFamily) {

if (!this.canCraftWeapon(weaponFamily)) {

console.error(`Cannot craft ${weaponFamily} - missing pieces`);

return false;

}

// Remove used pieces

const requiredPieces = ['barrel', 'receiver', 'stock', 'magazine'];

for (const pieceType of requiredPieces) {

const piece = this.data.inventory.gear\_pieces.find(

p => p.weapon\_family === weaponFamily && p.piece\_type === pieceType

);

if (piece) {

this.removeGearPiece(piece.piece\_id);

}

}

// Create completed weapon

const weapon = {

weapon\_id: `${weaponFamily}\_${Date.now()}`,

name: weaponFamily.replace('\_', ' ').toUpperCase(),

type: weaponFamily,

rarity: 'common',

stats: this.calculateWeaponStats(weaponFamily)

};

this.data.inventory.completed\_weapons.push(weapon);

this.data.progression.weapons\_crafted++;

console.log(`Weapon crafted: ${weapon.name}`);

this.save();

return weapon;

}

calculateWeaponStats(weaponFamily) {

// Base stats by weapon type

const baseStats = {

assault\_rifle: { accuracy: 60, damage: 35, fire\_rate: 650, range: 350 },

sniper\_rifle: { accuracy: 90, damage: 80, fire\_rate: 60, range: 800 },

shotgun: { accuracy: 40, damage: 90, fire\_rate: 120, range: 50 },

smg: { accuracy: 50, damage: 25, fire\_rate: 900, range: 200 },

pistol: { accuracy: 70, damage: 30, fire\_rate: 300, range: 100 }

};

return baseStats[weaponFamily] || baseStats.assault\_rifle;

}

addConsumable(type, quantity = 1) {

if (!this.data.inventory.consumables[type]) {

this.data.inventory.consumables[type] = 0;

}

this.data.inventory.consumables[type] += quantity;

this.save();

}

useConsumable(type) {

if (!this.data.inventory.consumables[type] || this.data.inventory.consumables[type] <= 0) {

console.error(`No ${type} available`);

return false;

}

this.data.inventory.consumables[type]--;

console.log(`Used: ${type}. Remaining: ${this.data.inventory.consumables[type]}`);

this.save();

return true;

}

// ====================

// STATUS & PENALTIES

// ====================

isInHospital() {

if (!this.data.status.hospital\_until) return false;

return new Date() < new Date(this.data.status.hospital\_until);

}

isInJail() {

if (!this.data.status.jail\_until) return false;

return new Date() < new Date(this.data.status.jail\_until);

}

sendToHospital(minutes) {

const releaseTime = new Date();

releaseTime.setMinutes(releaseTime.getMinutes() + minutes);

this.data.status.hospital\_until = releaseTime.toISOString();

console.log(`Sent to hospital for ${minutes} minutes`);

this.save();

}

sendToJail(minutes) {

const releaseTime = new Date();

releaseTime.setMinutes(releaseTime.getMinutes() + minutes);

this.data.status.jail\_until = releaseTime.toISOString();

console.log(`Sent to jail for ${minutes} minutes`);

this.save();

}

reducePenaltyTime(type, percentage) {

const statusKey = type === 'hospital' ? 'hospital\_until' : 'jail\_until';

const releaseTime = new Date(this.data.status[statusKey]);

const now = new Date();

const remainingMs = releaseTime - now;

const reductionMs = remainingMs \* (percentage / 100);

releaseTime.setMilliseconds(releaseTime.getMilliseconds() - reductionMs);

this.data.status[statusKey] = releaseTime.toISOString();

console.log(`${type} time reduced by ${percentage}%`);

this.save();

}

release(type) {

const statusKey = type === 'hospital' ? 'hospital\_until' : 'jail\_until';

this.data.status[statusKey] = null;

console.log(`Released from ${type}`);

this.save();

}

// ====================

// PROGRESSION

// ====================

addExperience(amount) {

this.data.progression.experience += amount;

// Check for level up

while (this.data.progression.experience >= this.data.progression.experience\_to\_next\_level) {

this.levelUp();

}

this.save();

}

levelUp() {

this.data.progression.level++;

this.data.progression.experience -= this.data.progression.experience\_to\_next\_level;

this.data.progression.experience\_to\_next\_level = Math.floor(

this.data.progression.experience\_to\_next\_level \* 1.5

);

console.log(`LEVEL UP! Now level ${this.data.progression.level}`);

// Level up rewards

this.addCredits(500, 'level\_up');

this.data.training.tokens\_available += 1;

this.save();

}

completeMission(success, rewards) {

if (success) {

this.data.progression.missions\_completed++;

this.addCredits(rewards.credits, 'mission\_completion');

this.addExperience(rewards.experience);

// Add gear pieces

if (rewards.gear\_pieces) {

rewards.gear\_pieces.forEach(piece => this.addGearPiece(piece));

}

// Add consumables

if (rewards.consumables) {

Object.entries(rewards.consumables).forEach(([type, qty]) => {

this.addConsumable(type, qty);

});

}

} else {

this.data.progression.missions\_failed++;

}

this.save();

}

recordPvPResult(won) {

if (won) {

this.data.progression.pvp\_wins++;

this.data.fear += 5;

} else {

this.data.progression.pvp\_losses++;

}

this.save();

}

// ====================

// JOBS & SPECIALTIES

// ====================

unlockJob(jobName) {

if (!this.data.jobs.includes(jobName)) {

this.data.jobs.push(jobName);

console.log(`Job unlocked: ${jobName}`);

this.save();

}

}

setActiveJob(jobName) {

if (!this.data.jobs.includes(jobName)) {

console.error(`Job not unlocked: ${jobName}`);

return false;

}

this.data.active\_job = jobName;

console.log(`Active job changed to: ${jobName}`);

this.save();

return true;

}

unlockSpecialty(specialtyName) {

if (!this.data.specialties.includes(specialtyName)) {

this.data.specialties.push(specialtyName);

console.log(`Specialty unlocked: ${specialtyName}`);

this.save();

}

}

// ====================

// SESSIONS

// ====================

startSession() {

this.data.last\_login = new Date().toISOString();

this.sessionStartTime = Date.now();

this.save();

}

endSession() {

if (this.sessionStartTime) {

const sessionDuration = Math.floor((Date.now() - this.sessionStartTime) / 1000);

this.data.progression.total\_playtime += sessionDuration;

this.data.session\_history.push({

start: new Date(this.sessionStartTime).toISOString(),

end: new Date().toISOString(),

duration: sessionDuration

});

// Keep only last 50 sessions

if (this.data.session\_history.length > 50) {

this.data.session\_history = this.data.session\_history.slice(-50);

}

this.save();

}

}

// ====================

// SAVE/LOAD

// ====================

save() {

try {

const serialized = JSON.stringify(this.data);

localStorage.setItem(`player\_bible\_${this.playerId}`, serialized);

console.log('Player Bible saved');

return true;

} catch (error) {

console.error('Failed to save Player Bible:', error);

return false;

}

}

load() {

try {

const saved = localStorage.getItem(`player\_bible\_${this.playerId}`);

if (saved) {

this.data = JSON.parse(saved);

console.log('Player Bible loaded');

return true;

}

} catch (error) {

console.error('Failed to load Player Bible:', error);

}

return false;

}

enableAutosave(intervalSeconds = 60) {

if (this.autosaveInterval) {

clearInterval(this.autosaveInterval);

}

this.autosaveInterval = setInterval(() => {

this.save();

}, intervalSeconds \* 1000);

console.log(`Autosave enabled: every ${intervalSeconds} seconds`);

}

disableAutosave() {

if (this.autosaveInterval) {

clearInterval(this.autosaveInterval);

this.autosaveInterval = null;

console.log('Autosave disabled');

}

}

// ====================

// EXPORT/IMPORT

// ====================

exportData() {

return JSON.stringify(this.data, null, 2);

}

importData(jsonString) {

try {

const imported = JSON.parse(jsonString);

this.data = imported;

this.save();

console.log('Player data imported successfully');

return true;

} catch (error) {

console.error('Failed to import data:', error);

return false;

}

}

}

// ===== EXPORT =====

if (typeof module !== 'undefined' && module.exports) {

module.exports = { PlayerBible };

}