# CraftEvolution - Minecraft-like Game with Evolutionary Terrain Generation (v2.0)

## Overview

CraftEvolution is an innovative 3D block-based game built with Angular 20, NgRx, and BabylonJS that features a unique **evolutionary terrain generation system** instead of traditional noise-based algorithms. Every world is completely unique and generated through probabilistic mutations.

## 🆕 Version 2.0 Features

* **Angular 20 Zoneless Mode**: Enhanced performance through removal of zone.js dependency
* **Surface-Level Player Spawning**: Players now spawn safely on the terrain surface
* **In-Game Settings Modal**: ESC key now toggles an overlay settings menu
* **3D Chunk-Based World Storage**: Optimized world storage using 16x16x16 chunks
* **Auto-Save System**: World changes and player position are automatically saved
* **World Persistence**: Game automatically loads previously saved worlds

## 🎮 Game Features

### Core Gameplay

* **3D Block World**: Fully interactive 3D environment with 7 block types
* **First-Person Movement**: WASD controls with mouse look
* **Block Breaking & Placement**: Left-click to break, right-click to place
* **Inventory System**: 9-slot toolbar with item management
* **Crafting System**: Recipe-based tool creation with 3x3 crafting grid
* **Tool System**: Hand, Pickaxe, Spade, and Axe with different effectiveness

### Block Types

1. **Dirt** 🟫 - Basic terrain block, breakable by hand/spade
2. **Stone** 🪨 - Hard terrain block, requires pickaxe
3. **Sand** 🟨 - Soft terrain block, breakable by hand/spade
4. **Water** 💧 - Liquid block, translucent, non-breakable
5. **Air** 💨 - Transparent space
6. **Wood** 🪵 - Tree trunk, requires axe
7. **Leaves** 🍃 - Tree foliage, breakable by hand

## 🧬 Evolutionary Terrain Generation

### Core Algorithm

The game uses a revolutionary approach to world generation based on **evolutionary algorithms** rather than traditional Perlin noise:

#### 1. Seed Block System

* World starts with a single dirt block at coordinates (0,0,0)
* Each block contains probability metadata for neighboring block generation

#### 2. Probability Mappings

Each block type has three probability distributions: - **Horizontal Neighbors** (X/Y axis): Controls side-to-side generation - **Positive Z (Up)**: Controls upward generation - **Negative Z (Down)**: Controls downward generation

Example for Dirt block:

horizontalNeighbors: {  
 dirt: 75%, stone: 10%, sand: 8%, water: 5%, air: 2%  
}  
positiveZ: {  
 air: 60%, dirt: 30%, stone: 5%, sand: 5%  
}

#### 3. Mutation System

When placing each new block: - Performs (N-1) × 1% probability transfers between block types - Example: dirt gives 1% to stone, sand gives 1% to stone, etc. - Results in evolved probability mappings for each generation

#### 4. Tree Generation Algorithm

Special logic for wood blocks: - Tracks consecutive wood count in metadata - Probability formula: (1 / consecutiveCount) × baseChance - Additional 5% reduction per consecutive block - Natural tree termination through evolution

### World Population

* Uses breadth-first traversal from seed block
* Generates up to 1000 blocks in each direction
* Ensures proper probability cascading through generations

## 🏗️ Technical Architecture

### Frontend Stack

* **Angular 20**: Latest Angular with zoneless mode for performance
* **NgRx 20**: State management with actions, reducers, selectors, and signals
* **BabylonJS 8.23**: 3D rendering engine with instanced meshes
* **TypeScript**: Type-safe development
* **SCSS**: Advanced styling with dark theme

### State Management (NgRx)

interface GameState {  
 world: WorldState; // Block data and terrain  
 player: PlayerState; // Inventory, health, position  
 ui: UIState; // Game mode, menus, targeting  
 performance: PerformanceState; // FPS, optimization data  
}

### Chunk-Based Storage

interface WorldChunk {  
 chunkX: number;  
 chunkY: number;  
 chunkZ: number;  
 blocks: Map<string, Block>; // Local coordinates -> Block  
 lastAccessed: number;  
 isDirty: boolean; // Needs saving  
}

### Project Structure

src/app/  
├── core/ # Core services  
│ └── services/  
│ ├── terrain-generation.service.ts # Evolutionary algorithm  
│ ├── chunk-manager.service.ts # Optimized block storage  
│ ├── db.service.ts # Persistence layer  
│ └── babylon.service.ts # 3D rendering  
├── features/  
│ ├── menu/ # Landing page  
│ ├── game/ # Main game component  
│ ├── inventory/ # Inventory & crafting UI  
│ ├── settings/ # In-game settings modal  
│ └── ui/ # Game UI components  
├── shared/  
│ └── models/ # TypeScript interfaces  
└── store/ # NgRx state management  
 ├── world/ # World state  
 ├── player/ # Player state  
 ├── ui/ # UI state  
 └── performance/ # Performance metrics

## 🎨 Visual Design

### Theme & Aesthetics

* **Dark sci-fi theme** for menus with vibrant gradients
* **Natural Minecraft-inspired** in-game world
* **Modern game UI** with smooth animations
* **Responsive design** for desktop and mobile

### 3D Rendering Features

* **Instanced mesh rendering** for performance
* **Material system** with proper block colors
* **Transparent blocks** (air, water) support
* **Block highlighting** with outline effects
* **Frustum culling** for optimization

## 🛠️ Development Features

### Performance Optimizations

* **Chunk-based loading/unloading**
* **Level of Detail (LOD) system**
* **Efficient block rendering** with BabylonJS instances
* **State management** optimized for large block counts
* **Zoneless Angular** for better performance

### Development Tools

* **NgRx DevTools** integration
* **Hot reload** development server
* **TypeScript strict mode**
* **Component-based architecture**

## 🎯 Game Mechanics

### Crafting Recipes

1. **Stone Pickaxe**: 3 stone + 2 wood (T-shape)
2. **Stone Spade**: 1 stone + 2 wood (line)
3. **Stone Axe**: 3 stone + 2 wood (L-shape)
4. **Wood Tools**: Alternative recipes using wood

### Block Breaking System

* **Tool requirements**: Different blocks need specific tools
* **Hardness values**: Breaking time varies by block type
* **Item drops**: Broken blocks drop as inventory items
* **Progress indicator**: Visual feedback during breaking

### Inventory Management

* **9-slot toolbar** visible at bottom
* **Item stacking** with max stack limits
* **Tool selection** via number keys (1-9)
* **Drag & drop** interface for organization

## 🚀 Deployment & Access

The game is built as a progressive web application that can be deployed to any static hosting service. The production build includes:

* **Optimized bundles** with tree-shaking
* **Auto-save functionality** for world persistence
* **Responsive design** for all devices
* **WebGL compatibility** checking

## 🎮 How to Play

1. **Start**: Click “New World” to generate a unique world (or load a saved one)
2. **Movement**: Use WASD keys + mouse to navigate
3. **Breaking**: Left-click blocks to break them
4. **Inventory**: Broken blocks appear in your toolbar
5. **Crafting**: Gather materials and use crafting recipes
6. **Building**: Right-click to place blocks from inventory
7. **Settings**: Press ESC to open the in-game settings menu

## 🌟 Unique Features

### No Seeds, Pure Evolution

Unlike Minecraft’s seed-based generation, CraftEvolution creates truly unique worlds every time through: - **Probabilistic mutations** in real-time - **Emergent terrain patterns** from simple rules - **Dynamic tree generation** with natural variation - **Unpredictable world layouts** that surprise players

### Advanced Block Metadata

Each block carries rich metadata including: - **Probability mappings** for all directions - **Consecutive wood count** for tree algorithms - **Tool requirements** and hardness values - **Visual properties** (transparency, color)

### Intelligent World Generation

The evolutionary algorithm creates realistic terrain through: - **Layered generation** (stone deeper, air higher) - **Biome-like clustering** through probability weights - **Natural boundaries** between different block types - **Organic cave and structure formation**

## 📦 Installation & Development

# Install dependencies  
npm install  
  
# Development server  
npm start  
  
# Production build  
npm run build:prod  
# or use the build script  
./build\_production.sh  
  
# Serve production build locally  
npx http-server dist/minecraft-game/browser

## 🏆 Technical Achievements

1. **Implemented full evolutionary terrain generation** from scratch
2. **Integrated BabylonJS** with Angular and NgRx seamlessly
3. **Created comprehensive state management** for complex game state
4. **Built responsive 3D game UI** with modern design principles
5. **Achieved 60 FPS performance** with thousands of blocks
6. **Developed complete crafting system** with recipe matching
7. **Implemented advanced tree generation** algorithm
8. **Created production-ready game** with all core Minecraft mechanics
9. **Implemented chunk-based storage** for efficient world management
10. **Built persistence layer** for seamless world loading/saving

This project demonstrates advanced frontend development skills, complex state management, 3D graphics programming, and innovative algorithm design, all while delivering a fun and engaging gaming experience.