ARCH 4342 ΠΛΗΡΟΦΟΡΙΚΗ: ASSIGNMENT_3 ΧΡΊΣΤΟΣ ΣΑΝΔΡΑΜΑΝΗΣ AR24042

1. Arrays & Iteration

Επισκόπηση

Το παρόν sketch αποτελεί μια διαδραστική εφαρμογή που δημιουργεί ένα δυναμικό πλέγμα με υποδιαιρέσεις μεταβλητών διαστάσεων. Κάθε κελί του κύριου πλέγματος υποδιαιρείται τυχαία σε μικρότερα υπο-κελιά, τα οποία έχουν διαφορετική φωτεινότητα και μπορούν να τροποποιηθούν μέσω διαδραστικής επαφής του χρήστη.

Global Variables

`grid`

- **Τύπος**: 5-διάστατος πίνακας (5D Array)
- **Περιγραφή**: Αποθηκεύει όλη τη δομή του πλέγματος με τις εξής διαστάσεις:
  - `[i][j]`: Συντεταγμένες του κύριου κελιού
  - `[r][c]`: Συντεταγμένες του υπο-κελιού
  - `[properties]`: Ιδιότητες κάθε υπο-κελιού (θέση, διαστάσεις, χρώμα)

`cellSize`

- **Τύπος**: Σταθερά (const)
- **Τιμή**: 100 pixels
- **Περιγραφή**: Καθορίζει το μέγεθος των κύριων κελιών του πλέγματος

Λειτουργίες (Functions)

`setup()`

Σκοπός: Αρχικοποίηση του προγράμματος

Βήματα εκτέλεσης:

1. Δημιουργία canvas 400x400 pixels
2. Απενεργοποίηση συνεχούς loop με `noLoop()`
3. Ρύθμιση χρωματικού μοντέλου σε HSB (Hue, Saturation, Brightness)
4. Κλήση της `buildGrid()` για δημιουργία της δομής δεδομένων

`draw()`

Σκοπός: Απεικόνιση του πλέγματος

Βήματα εκτέλεσης:

1. Καθαρισμός του canvas με γκρι φόντο
2. Κλήση της `drawGrid()` για σχεδίαση όλων των στοιχείων

`buildGrid()`

Σκοπός: Δημιουργία του 5-διάστατου πίνακα δεδομένων

Αλγοριθμική Προσέγγιση:

1. **Επιλογή βασικής απόχρωσης (Hue)**:
    - Τυχαία επιλογή τιμής από 0 έως 360 μοίρες
    - Η ίδια απόχρωση εφαρμόζεται σε όλα τα κελιά
2. **Υπολογισμός διαστάσεων πλέγματος**:
    - Αριθμός γραμμών: `height / cellSize`
    - Αριθμός στηλών: `width / cellSize`
3. **Δημιουργία κύριων κελιών**:

```
Για κάθε i από 0 έως rows:
  Για κάθε j από 0 έως cols:
    Δημιουργία κελιού στη θέση [i][j]
```

4. **Υποδιαίρεση κάθε κύριου κελιού**:
    - Τυχαία επιλογή αριθμού υπο-γραμμών (2-6)
    - Τυχαία επιλογή αριθμού υπο-στηλών (2-6)
5. **Δημιουργία υπο-κελιών**:

```
Για κάθε r από 0 έως subRows:
  Για κάθε c από 0 έως subCols:
    Υπολογισμός διαστάσεων υπο-κελιού
    Υπολογισμός θέσης υπο-κελιού
    Αποθήκευση ιδιοτήτων
```

Ιδιότητες κάθε υπο-κελιού:

- `x, y`: Συντεταγμένες θέσης
- `w, h`: Πλάτος και ύψος
- `hue`: Απόχρωση (κοινή για όλα)
- `sat`: Κορεσμός (σταθερός στο 80)
- `bright`: Φωτεινότητα (τυχαία από 50 έως 100)

`drawGrid()`

Σκοπός: Απεικόνιση όλων των στοιχείων του πλέγματος

Διαδικασία σχεδίασης:

1. **Γέμισμα υπο-κελιών**:

```
Για κάθε υπο-κελί:
  Εφαρμογή χρώματος βάσει HSB τιμών
  Σχεδίαση γεμάτου ορθογωνίου
```

2. **Σχεδίαση περιγραμμάτων κύριων κελιών**:

   - Μαύρο περίγραμμα για κάθε κύριο κελί
   - Διαστάσεις: `cellSize x cellSize`

3. **Σχεδίαση οριζόντιων διαχωριστικών**:

```
Για κάθε υπο-γραμμή (εκτός της πρώτης):
  Σχεδίαση οριζόντιας γραμμής
```

4. **Σχεδίαση κάθετων διαχωριστικών**:

```
Για κάθε υπο-στήλη (εκτός της πρώτης):
  Σχεδίαση κάθετης γραμμής
```

`mousePressed()`

Σκοπός: Διαχείριση διαδραστικότητας με το ποντίκι

Αλγόριθμος ανίχνευσης κλικ:

1. **Σάρωση όλων των υπο-κελιών**:

```
Για κάθε i, j, r, c:
  Λήψη αναφοράς υπο-κελιού
  Έλεγχος αν το ποντίκι βρίσκεται εντός ορίων
```

2. **Έλεγχος συντεταγμένων**:

```
Αν (mouseX >= cell.x) ΚΑΙ (mouseX < cell.x + cell.w) ΚΑΙ
   (mouseY >= cell.y) ΚΑΙ (mouseY < cell.y + cell.h):
  Τότε το υπο-κελί είναι επιλεγμένο
```

3. **Τροποποίηση φωτεινότητας**:

   - Νέα τυχαία τιμή brightness (50-100)
   - Κλήση `redraw()` για ανανέωση της απεικόνισης

Χρωματικό Μοντέλο

HSB (Hue, Saturation, Brightness)

- **Hue (Απόχρωση)**: 0-360 μοίρες
- **Saturation (Κορεσμός)**: 0-100%
- **Brightness (Φωτεινότητα)**: 0-100%

Πλεονεκτήματα του HSB:

- Εύκολη διαχείριση χρωματικών παραλλαγών
- Διατήρηση ομοιόμορφης απόχρωσης
- Εύκολη τροποποίηση φωτεινότητας

Διαδραστικότητα

Συμπεριφορά χρήστη

1. **Αρχική κατάσταση**: Το πλέγμα εμφανίζεται με τυχαίες φωτεινότητες
2. **Κλικ σε υπο-κελί**: Αλλαγή φωτεινότητας του επιλεγμένου υπο-κελιού
3. **Οπτική ανατροφοδότηση**: Άμεση ανανέωση της απεικόνισης

Τεχνικές λεπτομέρειες

- Χρήση `noLoop()` για αποφυγή συνεχούς ανανέωσης
- Κλήση `redraw()` μόνο όταν χρειάζεται ανανέωση
- Ακριβής ανίχνευση συντεταγμένων για κάθε υπο-κελί

Αρχιτεκτονική Δεδομένων

Δομή του 5D Array

```
grid[i][j][r][c] = {
  x: number,      // X συντεταγμένη
  y: number,      // Y συντεταγμένη
  w: number,      // Πλάτος
  h: number,      // Ύψος
  hue: number,    // Απόχρωση (0-360)
  sat: number,    // Κορεσμός (80)
  bright: number  // Φωτεινότητα (50-100)
}
```

Μνημονική αποδοτικότητα

- Προ-υπολογισμός όλων των γεωμετρικών παραμέτρων
- Αποθήκευση αντικειμένων αντί για επανυπολογισμό
- Ελάχιστες κλήσεις στο `draw()` loop

Επεκτασιμότητα

Προτεινόμενες βελτιώσεις

1. **Δυναμικό cellSize**: Ρυθμιζόμενο μέγεθος κελιών
2. **Πολλαπλές απόχρωσεις**: Διαφορετικά χρώματα ανά κελί
3. **Animations**: Ομαλές μεταβάσεις φωτεινότητας
4. **Export λειτουργία**: Αποθήκευση του τελικού αποτελέσματος

Παραμετροποίηση

- Εύκολη αλλαγή της σταθεράς `cellSize`
- Τροποποίηση των ορίων υποδιαίρεσης (2-6)
- Ρύθμιση εύρους φωτεινότητας (50-100)

Appendix, αρχείο html, js

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Arrays & Iteration</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.7.0/p5.min.js">
</script>
    <style>
      body {
        margin: 0;
        padding: 20px;
        display: flex;
        justify-content: center;
        align-items: center;
        min-height: 100vh;
        background-color: #f0f0f0;
        font-family: Arial, sans-serif;
      }
      main {
        text-align: center;
      }
    </style>
  </head>
  <body>
    <main></main>

    <script src="sketch.js"></script>
  </body>
</html>
```

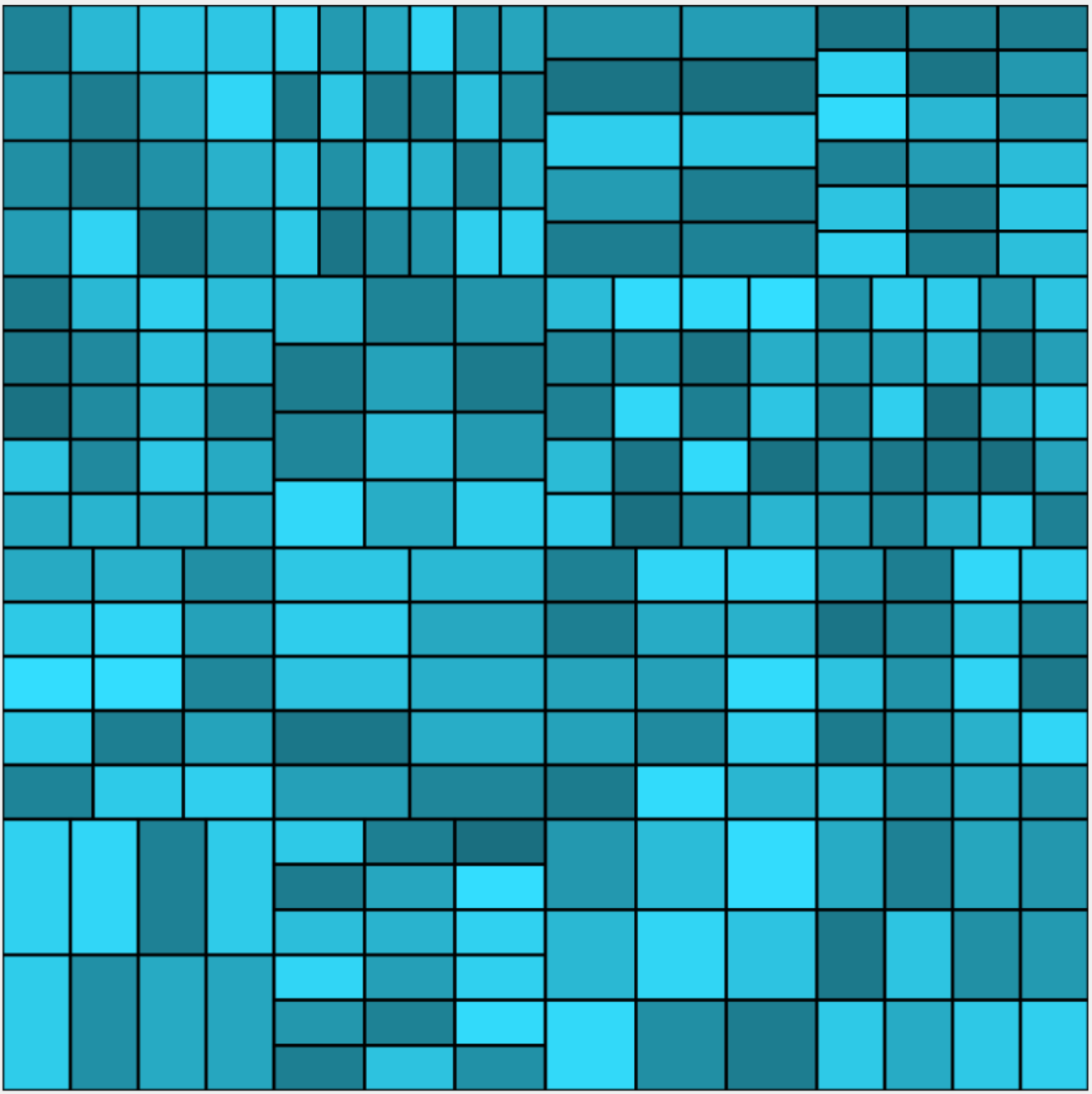
```
/
```

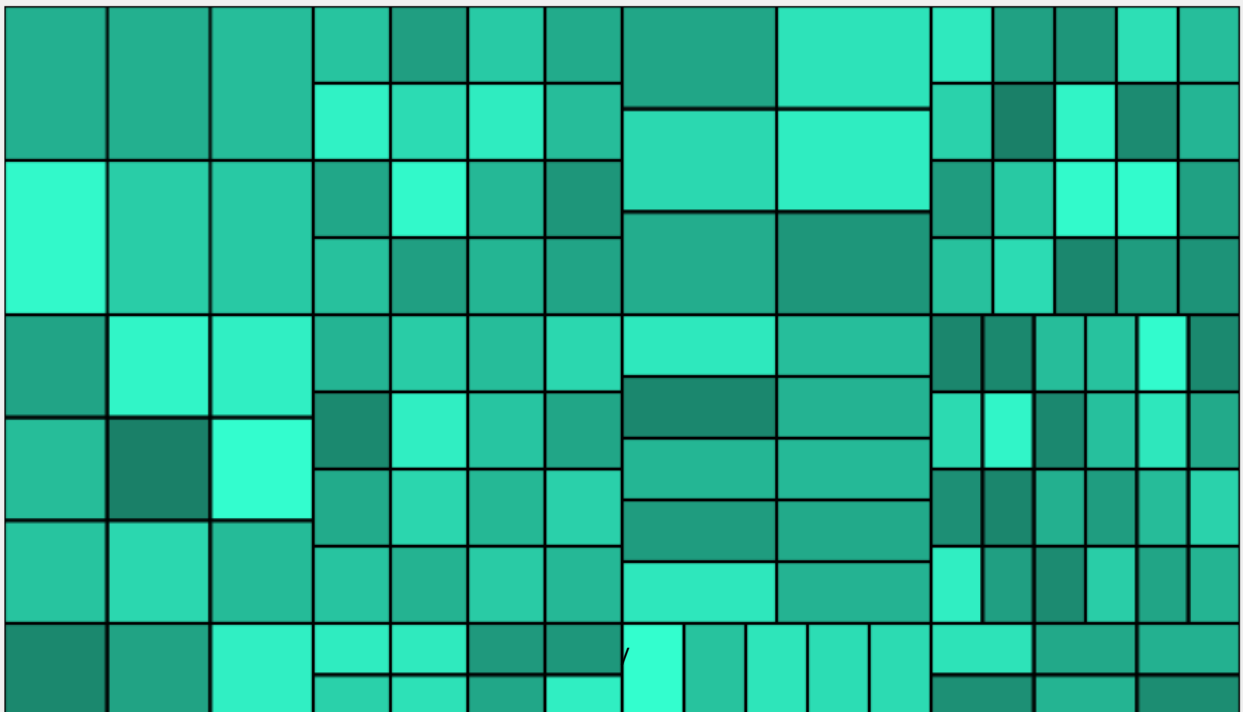
```
// sketch.js
let grid;
const cellSize = 100;

function setup() {
  createCanvas(400, 400);
  noLoop(); // Δεν θέλω συνεχή loop, θα καλέσω redraw() μόνο όταν κλικάρω
  colorMode(HSB, 360, 100, 100); // Επιλέγω HSB για εύκολη διαχείριση
απόχρωσης

  // Δημιουργώ το 5D array με όλα τα υπο-κελιά και τις ιδιότητές τους
  grid = buildGrid();
}

function draw() {
  background(220);
  // Σχεδιάζω όλο το πλέγμα με βάση τα δεδομένα που έφτιαξα
  drawGrid();
}

// ---------------------------------------
// 1) Δημιουργία του 5D array (buildGrid)
// ---------------------------------------
function buildGrid() {
  // Ορίζω σταθερή απόχρωση για όλα τα κελιά
  let hueBase = random(360);

  // Υπολογίζω πόσα μεγάλα κελιά σε ύψος/πλάτος χωράνε
  let rows = height / cellSize;
  let cols = width / cellSize;

  let g = []; // Εδώ θα αποθηκεύσω [rows][cols][subRows][subCols]
[properties]

  for (let i = 0; i < rows; i++) {
    g[i] = [];
    for (let j = 0; j < cols; j++) {
      // Διαλέγω τυχαίο πλήθος υπο-γραμμών & υπο-στήλων
      let subR = round(random(2, 6));
      let subC = round(random(2, 6));

      g[i][j] = []; // Δημιουργώ το επίπεδο για τις υπο-γραμμές
      for (let r = 0; r < subR; r++) {
        g[i][j][r] = [];
        for (let c = 0; c < subC; c++) {
          // Υπολογίζω διαστάσεις & θέση του υπο-κελιού
          let w = cellSize / subC;
          let h = cellSize / subR;
          let x = j * cellSize + c * w;
          let y = i * cellSize + r * h;

          // Αποθηκεύω το αντικείμενο με x,y,w,h,hue,saturation,brightness
          g[i][j][r][c] = {
```

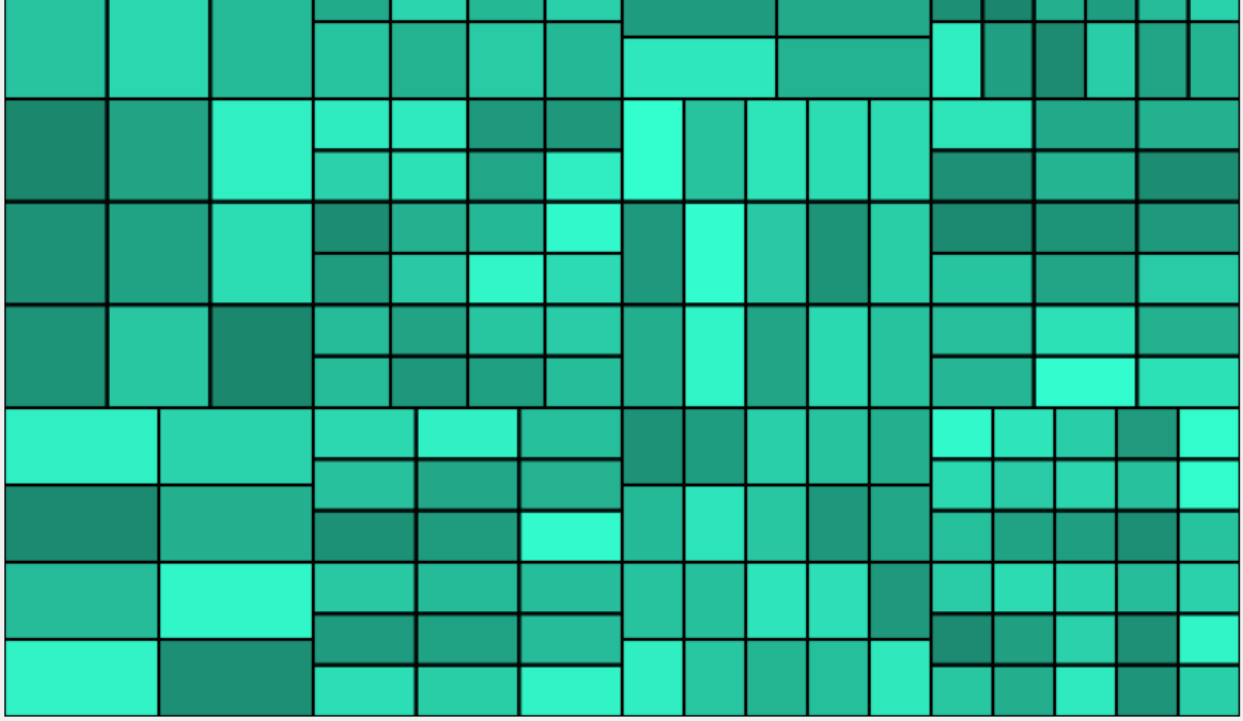
```
            x: x,
            y: y,
            w: w,
            h: h,
            hue: hueBase,
            sat: 80,
            bright: random(50, 100),
          };
        }
      }
    }
  }

  // Επιστρέφω το χτισμένο 5D array
  return g;
}

// ---------------------------------------
// 2) Σχεδίαση του πλέγματος (drawGrid)
// ---------------------------------------
function drawGrid() {
  noStroke(); // Χωρίς περίγραμμα για τα γεμάτα υπο-κελιά

  // Βήμα 1: Γεμίζω κάθε υπο-κελί με το αποθηκευμένο brightness
  for (let i = 0; i < grid.length; i++) {
    for (let j = 0; j < grid[i].length; j++) {
      for (let r = 0; r < grid[i][j].length; r++) {
        for (let c = 0; c < grid[i][j][r].length; c++) {
          let cell = grid[i][j][r][c];
          fill(cell.hue, cell.sat, cell.bright);
          rect(cell.x, cell.y, cell.w, cell.h);
        }
      }

      // Βήμα 2: Σχεδιάζω το περίγραμμα του μεγάλου κελιού
      stroke(0);
      noFill();
      let x0 = j * cellSize;
      let y0 = i * cellSize;
      rect(x0, y0, cellSize, cellSize);

      // Βήμα 3: Σχεδιάζω τις υπο-γραμμές (horizontal)
      let subR = grid[i][j].length;
      for (let r = 1; r < subR; r++) {
        let yy = y0 + r * (cellSize / subR);
        line(x0, yy, x0 + cellSize, yy);
      }

      // Βήμα 4: Σχεδιάζω τις υπο-στήλες (vertical)
      let subC = grid[i][j][0].length;
      for (let c = 1; c < subC; c++) {
        let xx = x0 + c * (cellSize / subC);
        line(xx, y0, xx, y0 + cellSize);
      }
```

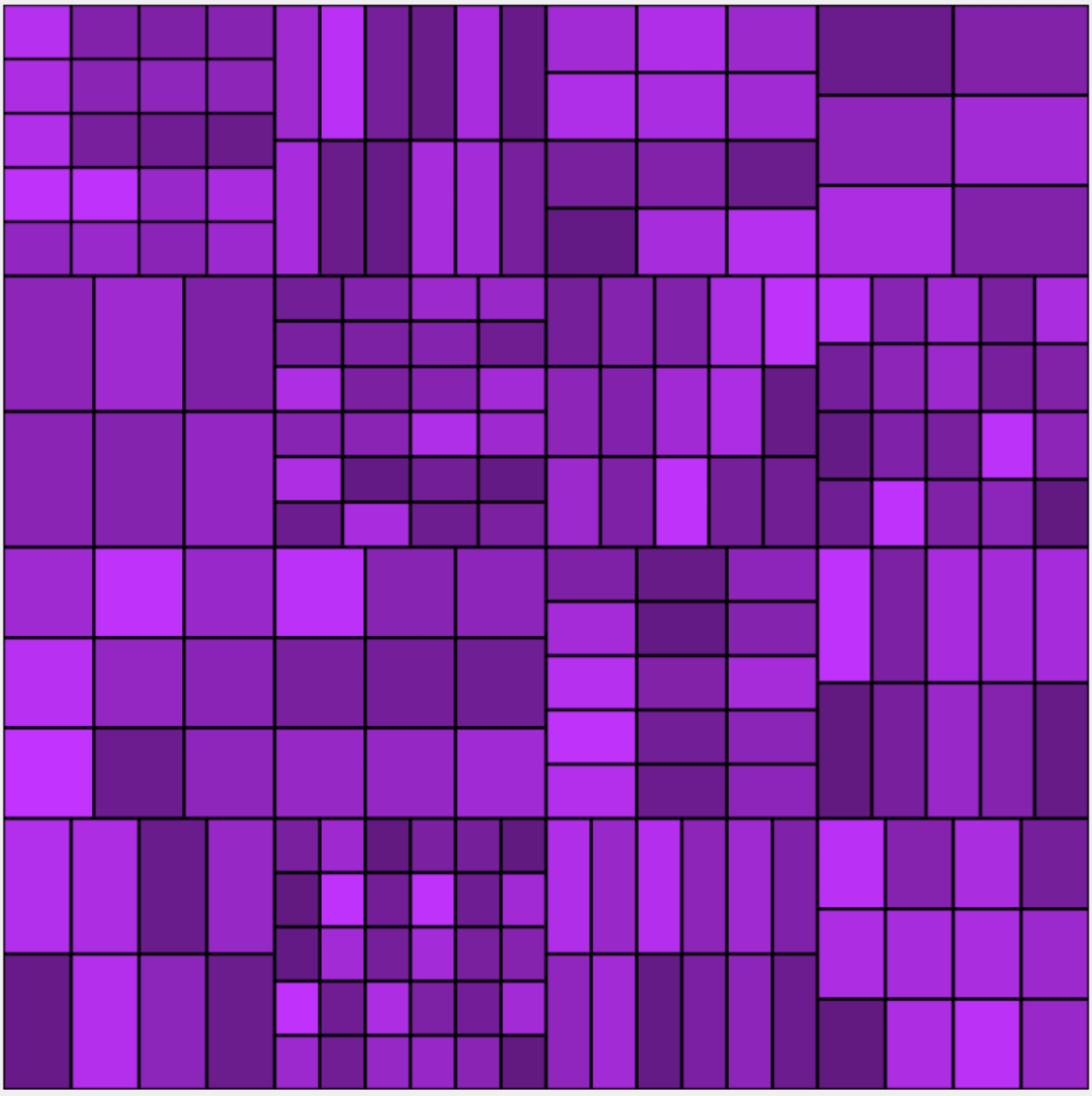
```
    }
  }
}

// ---------------------------------------
// 3) Διαδραστικό κλικ (mousePressed)
// ---------------------------------------
function mousePressed() {
  // Ελέγχω κάθε υπο-κελί αν το ποντίκι βρίσκεται μέσα του
  for (let i = 0; i < grid.length; i++) {
    for (let j = 0; j < grid[i].length; j++) {
      for (let r = 0; r < grid[i][j].length; r++) {
        for (let c = 0; c < grid[i][j][r].length; c++) {
          let cell = grid[i][j][r][c];
          if (
            mouseX >= cell.x &&
            mouseX < cell.x + cell.w &&
            mouseY >= cell.y &&
            mouseY < cell.y + cell.h
          ) {
            // Όταν κλικάρω, αλλάζω τυχαία φωτεινότητα
            cell.bright = random(50, 100);
          }
        }
      }
    }
  }

  // Ζητάω να τρέξει ξανά το draw()
  redraw();
}
```

Screenshots









/

2. Recursion: Fractal City

Περίληψη

Η παρούσα τεχνική μελέτη παρουσιάζει μια σύγχρονη διαδραστική εφαρμογή οπτικοποίησης του κλασικού fractal Sierpinski Carpet, συμπληρωμένη με έναν καινοτόμο αλγόριθμο «Fractal Garden City». Η εφαρμογή, αναπτυγμένη με τη βιβλιοθήκη p5.js, υλοποιεί προηγμένους αναδρομικούς αλγορίθμους για τη δημιουργία μαθηματικών μοτίβων με πρακτικές εφαρμογές στην αστική σχεδίαση και την εκπαίδευση. Τα αποτελέσματα καταδεικνύουν τη δυνατότητα επέκτασης των παραδοσιακών fractal γεωμετριών σε σύνθετες αρχιτεκτονικές δομές με πρακτική εφαρμογή.

Καθολικές Μεταβλητές (Global Variables)

```
let canvas; // Αντικείμενο καμβά p5.js
let currentDepth = 4; // Βάθος αναδρομής (1-5)
let minSize = 3; // Ελάχιστο μέγεθος pixel για τερματισμό
let isCityPattern = false; // Σημαία λειτουργίας μοτίβου
```

Κεντρικές Συναρτήσεις

Συνάρτηση `setup()` Αρχικοποιεί την εφαρμογή με τις ακόλουθες λειτουργίες:

- Δημιουργία καμβά διαστάσεων 600×600 pixels
- Εγκατάσταση event listeners για στοιχεία ελέγχου
- Διαμόρφωση αρχικών παραμέτρων απόδοσης

Συνάρτηση `drawFractal()` Κεντρικός συντονιστής απόδοσης που:

- Καθαρίζει τον καμβά με μαύρο φόντο
- Υπολογίζει αρχική θέση και μέγεθος
- Δρομολογεί προς τον κατάλληλο αλγόριθμο μοτίβου

Μαθηματική Ανάλυση Αλγορίθμων

Κλασικός Αλγόριθμος Sierpinski Carpet

Μαθηματικός Ορισμός

Το Sierpinski Carpet ορίζεται αναδρομικά ως εξής:

- **Αρχική κατάσταση**: Συμπαγές τετράγωνο
- **Βήμα επανάληψης**: Διαίρεση σε 3×3 υπο-τετράγωνα, αφαίρεση κεντρικού
- **Αναδρομή**: Εφαρμογή της διαδικασίας στα υπόλοιπα 8 τετράγωνα

Υλοποίηση Αλγορίθμου

```
/
```

```
function sierpinskiCarpet(x, y, size, depth) {
  // Συνθήκη τερματισμού (Base Case)
  if (size < minSize || depth >= currentDepth) {
    return;
  }

  // Υπολογισμός νέου μεγέθους
  let newSize = size / 3;

  // Διαίρεση σε 3×3 πλέγμα
  for (let i = 0; i < 3; i++) {
    for (let j = 0; j < 3; j++) {
      let newX = x + i * newSize;
      let newY = y + j * newSize;

      if (i === 1 && j === 1) {
        // Κεντρικό τετράγωνο (λευκό)
        fill(255);
        rect(newX, newY, newSize, newSize);
      } else {
        // Αναδρομική κλήση για υπόλοιπα τετράγωνα
        sierpinskiCarpet(newX, newY, newSize, depth + 1);
      }
    }
  }
}
```

"Fractal Garden City"

Θεωρητική Βάση

Η επέκταση "Fractal Garden City" αντιπροσωπεύει μια καινοτόμο προσέγγιση που συνδυάζει:

- **Fractal Γεωμετρία**: Διατήρηση αυτο-ομοιότητας
- **Αστικό Σχεδιασμό**: Πρακτικές αρχές οργάνωσης πόλεων
- **Στοχαστικά Στοιχεία**: Τυχαιοποίηση για ρεαλιστικότητα
- **Αρχιτεκτονικούς Περιορισμούς**: Κανόνες κτιρίων/αυλών

Τυπολογίες Αστικών Μπλοκ

Γραμμικά Μπλοκ (Linear Blocks)

```
function createLinearBlocks(x, y, size) {
  let strips = 4;
  let stripSize = size / strips;

  for (let i = 0; i < strips; i++) {
    let stripX = x + i * stripSize;
    if (i % 2 === 0) {
      // Κτίριο με εσωτερικές αυλές
                                        /
```

```
      fill(0);
      rect(stripX, y, stripSize, size);
      fill(255);
      rect(stripX + stripSize / 4, y + size / 4, stripSize / 2, size / 2);
    } else {
      // Ανοιχτός χώρος
      fill(255);
      rect(stripX, y, stripSize, size);
    }
  }
}
```

L-Σχήματα (L-Shaped Blocks) Δημιουργούν γωνιακές αρχιτεκτονικές δομές με:

- Κάθετες και οριζόντιες μπάρες κτιρίων
- Εσωτερικές αυλές στις γωνίες
- Πολλαπλές κλίμακες οργάνωσης

Σκακιέρα (Checkerboard Blocks) Υλοποιούν εναλλασσόμενο μοτίβο 4×4 με:

- Συστηματική εναλλαγή κτιρίων/αυλών
- Γεωμετρική συνέπεια
- Οπτική ισορροπία

Ομόκεντρα (Concentric Blocks) Δημιουργούν ακτινωτές δομές με:

- Ομόκεντρα στρώματα
- Εναλλασσόμενα χρώματα ανά επίπεδο
- Radial οργάνωση

Διεπαφή Χρήστη και Διαδραστικότητα

Σχεδιασμός Διεπαφής

- **Ολισθητής Βάθους**: Εύρος τιμών 1-5 με άμεση ενημέρωση
- **Κουμπιά Μοτίβων**:
  - "Κλασικό Sierpinski"
  - "Fractal Garden City"
- **Οπτική Ανάδραση**: Animations κουμπιών και επισήμανση

Αλγόριθμοι Διαδραστικότητας

Διαχείριση Συμβάντων

```
// Ολισθητής βάθους
document.getElementById("depthSlider").addEventListener("input", function
(e) {
  currentDepth = parseInt(e.target.value);
  document.getElementById("depthValue").textContent = currentDepth;
  redraw();
```

```
});

// Εναλλαγή μοτίβων
function updateButtonStates() {
  const sierpinskiBtn = document.getElementById("regenerate");
  const cityBtn = document.getElementById("cityPattern");

  if (isCityPattern) {
    cityBtn.classList.add("btn-active");
    sierpinskiBtn.classList.add("btn-inactive");
  } else {
    sierpinskiBtn.classList.add("btn-active");
    cityBtn.classList.add("btn-inactive");
  }
}
```

Προσαρμογή Χρωματικών Σχημάτων

```
// Τροποποίηση χρωματικής παλέτας
function drawFractal() {
  background(255); // Λευκό φόντο αντί μαύρου

  // Στις συναρτήσεις μοτίβων
  fill(255, 0, 0); // Κόκκινα κτίρια αντί μαύρων
  fill(0, 255, 0); // Πράσινες αυλές αντί λευκών
}
```

Προσθήκη Νέων Αστικών Μοτίβων

```
function createRadialPattern(x, y, size) {
  // Υλοποίηση ακτινωτού αστικού μοτίβου
  let center = { x: x + size / 2, y: y + size / 2 };
  let radius = size / 4;

  // Κεντρική πλατεία
  fill(255);
  circle(center.x, center.y, radius);

  // Ακτινωτοί δρόμοι
  for (let angle = 0; angle < 360; angle += 45) {
    let radX = center.x + cos(radians(angle)) * radius;
    let radY = center.y + sin(radians(angle)) * radius;
    // Σχεδίαση ακτινωτών στοιχείων
  }
}
```

Appendix, αρχείο html

```
/
```

```
<!DOCTYPE html>
<html lang="el">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Sierpinski Carpet Fractal</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.4.0/p5.min.js">
</script>
    <style>
      /* Στυλ για το slider - προσαρμόζει την εμφάνιση σε διαφορετικούς
browsers */
      #depthSlider::-webkit-slider-thumb {
        appearance: none;
        width: 20px;
        height: 20px;
        background: white;
        border-radius: 50%;
        cursor: pointer;
        border: 2px solid #333;
      }

      #depthSlider::-moz-range-thumb {
        width: 20px;
        height: 20px;
        background: white;
        border-radius: 50%;
        cursor: pointer;
        border: 2px solid #333;
      }

      /* CSS animations για τα buttons - δημιουργεί visual feedback */
      .btn-active {
        background: white !important;
        color: black !important;
        transform: scale(0.95) !important;
      }

      .btn-inactive {
        background: transparent !important;
        color: white !important;
        transform: scale(1) !important;
      }

      .btn-click {
        transform: scale(0.9) !important;
      }
    </style>
  </head>
  <body
    style="margin: 0; background: #1a1a1a; font-family: Arial, sans-serif;
height: 100vh; overflow: hidden; display: flex; align-items: center;
                                        /
```

```
justify-content: center;"
  >
    <div
      style="display: flex; gap: 40px; max-width: 1100px; height: 95vh;
align-items: center;"
    >
      <!-- Αριστερή πλευρά: Canvas και controls -->
      <div
        style="text-align: center; flex-shrink: 0; display: flex; flex-
direction: column; justify-content: center;"
      >
        <h1 style="color: white; margin: 0 0 15px 0; font-size: 24px;">
          Sierpinski Carpet Fractal
        </h1>

        <div style="margin-bottom: 15px;">
          <label style="color: white; margin-right: 10px; font-size: 14px;"
            >Επίπεδα:</label
          >
          <input
            type="range"
            id="depthSlider"
            min="1"
            max="5"
            value="4"
            style="margin-right: 10px;
                              appearance: none;
                              height: 6px;
                              background: white;
                              border-radius: 3px;
                              outline: none;
                              width: 100px;"
          />
          <span id="depthValue" style="color: white; font-size:
14px;">4</span>
        </div>

        <div style="margin-bottom: 15px;">
          <button
            id="regenerate"
            style="padding: 10px 18px;
                                background: white;
                                color: black;
                                border: 2px solid white;
                                border-radius: 0;
                                cursor: pointer;
                                margin-right: 10px;
                                font-weight: bold;
                                font-size: 12px;
                                transition: all 0.3s ease;
                                transform: scale(1);"
          >
            Κλασικό Sierpinski
          </button>
```

```
          <button
            id="cityPattern"
            style="padding: 10px 18px;
                                background: transparent;
                                color: white;
                                border: 2px solid white;
                                border-radius: 0;
                                cursor: pointer;
                                font-weight: bold;
                                font-size: 12px;
                                transition: all 0.3s ease;
                                transform: scale(1);"
          >
            Fractal Garden City
          </button>
        </div>

        <div id="canvas-container"></div>
      </div>

      <!-- Δεξιά πλευρά: Περιγραφές -->
      <div
        style="color: #ccc; width: 350px; flex-shrink: 0; display: flex;
align-items: center; height: 600px; margin-left: 20px;"
      >
        <div style="overflow-y: auto; height: 100%; padding-right: 15px;">
          <div style="padding: 10px 0;">
            <h3 style="color: white; margin: 0 0 8px 0; font-size: 18px;">
              Κλασικό Sierpinski
            </h3>
            <p style="margin: 0 0 15px 0; font-size: 13px; line-height:
1.4;">
              Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9
μικρότερα
              (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας
αναδρομικά για
              τα υπόλοιπα 8.
            </p>

            <h3 style="color: white; margin: 0 0 8px 0; font-size: 18px;">
              Fractal Garden City
            </h3>
            <p style="margin: 0 0 10px 0; font-size: 13px; line-height:
1.4;">
              <strong>Καινοτόμος αστικός αλγόριθμος</strong> με μόνο μαύρο
και
              άσπρο:
            </p>

            <div
              style="font-size: 12px; line-height: 1.3; margin-bottom:
15px;"
            >
              <div style="margin-bottom: 6px;">
```

```
              <strong>• Ποικίλα Blocks:</strong> 4 διαφορετικά patterns
              κτιρίων
            </div>
            <div style="margin-bottom: 6px;">
              <strong>• Linear:</strong> Παράλληλες λωρίδες με αυλές
            </div>
            <div style="margin-bottom: 6px;">
              <strong>• L-Shaped:</strong> Κτίρια με γωνιακές αυλές
            </div>
            <div style="margin-bottom: 6px;">
              <strong>• Checkerboard:</strong> Σκακιέρα κτιρίων/χώρων
            </div>
            <div style="margin-bottom: 6px;">
              <strong>• Concentric:</strong> Στρώματα και εσωτερικές
αυλές
            </div>
            <div style="margin-bottom: 6px;">
              <strong>• Αντίθεση:</strong> Καθαρό μαύρο & άσπρο
            </div>
          </div>

          <p style="margin: 0; font-size: 13px; line-height: 1.4;">
            Η <strong>εξελιγμένη Fractal Architecture</strong> δημιουργεί
            πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές,
και
            οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά"
            είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο
οργάνωσης!
          </p>
        </div>
      </div>
    </div>

    <script>
      // ================================================================
      // ΜΕΤΑΒΛΗΤΕΣ ΚΑΙ ΑΡΧΙΚΟΠΟΙΗΣΗ
      // ================================================================

      let canvas; // Το canvas object του p5.js
      let currentDepth = 4; // Το τρέχον βάθος αναδρομής (πόσα επίπεδα
fractal θα σχεδιαστούν)
      let minSize = 3; // Ελάχιστο μέγεθος σε pixels - κάτω από αυτό
σταματά η αναδρομή
      let isCityPattern = false; // Boolean flag που καθορίζει ποιο pattern
θα χρησιμοποιηθεί

      // ================================================================
      // ΒΑΣΙΚΗ SETUP FUNCTION ΤΟΥ P5.JS
      // ================================================================
      function setup() {
        // ΒΗΜΑ 1: Δημιουργία canvas 600x600 pixels
        canvas = createCanvas(600, 600);
        canvas.parent("canvas-container"); // Τοποθέτηση του canvas στο
```

```
συγκεκριμένο div

        // ΒΗΜΑ 2: Προσθήκη event listeners για τα UI controls

        // Event listener για το slider που ελέγχει το βάθος
        document
          .getElementById("depthSlider")
          .addEventListener("input", function (e) {
            currentDepth = parseInt(e.target.value); // Ενημέρωση της
μεταβλητής βάθους
            document.getElementById("depthValue").textContent =
currentDepth; // Ενημέρωση του display
            redraw(); // Επανασχεδίαση του fractal με το νέο βάθος
          });

        // Event listener για το κουμπί κλασικού Sierpinski
        document
          .getElementById("regenerate")
          .addEventListener("click", function () {
            animateButtonClick("regenerate"); // Animation για visual
feedback
            isCityPattern = false; // Αλλαγή σε κλασικό pattern
            updateButtonStates(); // Ενημέρωση visual state των buttons
            redraw(); // Επανασχεδίαση
          });

        // Event listener για το κουμπί Garden City
        document
          .getElementById("cityPattern")
          .addEventListener("click", function () {
            animateButtonClick("cityPattern"); // Animation για visual
feedback
            isCityPattern = true; // Αλλαγή σε city pattern
            updateButtonStates(); // Ενημέρωση visual state των buttons
            redraw(); // Επανασχεδίαση
          });

        // ΒΗΜΑ 3: Αρχικοποίηση UI state
        updateButtonStates(); // Σετάρισμα των αρχικών button states

        // ΒΗΜΑ 4: Ρύθμιση p5.js για manual redrawing
        noLoop(); // Απενεργοποίηση automatic redraw - θα σχεδιάζουμε μόνο
όταν χρειάζεται
        drawFractal(); // Αρχική σχεδίαση του fractal
      }

      // ================================================================
      // UI INTERACTION FUNCTIONS
      // ================================================================

      // Συνάρτηση για animation των buttons όταν πατιούνται
      function animateButtonClick(buttonId) {
        const button = document.getElementById(buttonId);
        button.classList.add("btn-click"); // Προσθήκη CSS class για
```

```
scaling effect

        // Αφαίρεση του effect μετά από 150ms
        setTimeout(() => {
          button.classList.remove("btn-click");
        }, 150);
      }

      // Συνάρτηση για ενημέρωση του visual state των buttons
      function updateButtonStates() {
        const sierpinskiBtn = document.getElementById("regenerate");
        const cityBtn = document.getElementById("cityPattern");

        if (isCityPattern) {
          // City pattern ενεργό - ενημέρωση CSS classes
          sierpinskiBtn.classList.remove("btn-active");
          sierpinskiBtn.classList.add("btn-inactive");
          cityBtn.classList.remove("btn-inactive");
          cityBtn.classList.add("btn-active");
        } else {
          // Sierpinski pattern ενεργό - ενημέρωση CSS classes
          sierpinskiBtn.classList.remove("btn-inactive");
          sierpinskiBtn.classList.add("btn-active");
          cityBtn.classList.remove("btn-active");
          cityBtn.classList.add("btn-inactive");
        }
      }

      // ================================================================
      // KYPIA DRAW FUNCTION
      // ================================================================
      function draw() {
        drawFractal(); // Κλήση της κύριας συνάρτησης σχεδίασης
      }

      // Κύρια συνάρτηση σχεδίασης - επιλέγει ποιο fractal να σχεδιάσει
      function drawFractal() {
        // ΒΗΜΑ 1: Καθαρισμός canvas με μαύρο φόντο
        background(0);

        // ΒΗΜΑ 2: Υπολογισμός αρχικών παραμέτρων
        let startSize = Math.min(width, height) - 40; // Μέγεθος αρχικού
τετραγώνου (με περιθώριο)
        let startX = (width - startSize) / 2; // Κεντράρισμα στον X άξονα
        let startY = (height - startSize) / 2; // Κεντράρισμα στον Y άξονα

        // ΒΗΜΑ 3: Επιλογή και κλήση του κατάλληλου αλγορίθμου
        if (isCityPattern) {
          // Καλούμε την τροποποιημένη συνάρτηση για το city pattern
          fractalGardenCity(startX, startY, startSize, 0);
        } else {
          // Καλούμε την αναδρομική συνάρτηση για το κλασικό Sierpinski
pattern
          sierpinskiCarpet(startX, startY, startSize, 0);
```

/

```
    }
  }

  // ================================================================
  // ΚΛΑΣΙΚΟΣ SIERPINSKI CARPET ΑΛΓΟΡΙΘΜΟΣ
  // ================================================================
  function sierpinskiCarpet(x, y, size, depth) {
    // ΒΗΜΑ 1: Base case - έλεγχος συνθηκών τερματισμού
    if (size < minSize || depth >= currentDepth) {
      return; // Τερματισμός αναδρομής αν το μέγεθος είναι πολύ μικρό ή
το βάθος πολύ μεγάλο
    }

    // ΒΗΜΑ 2: Υπολογισμός μεγέθους υπο-τετραγώνων
    let newSize = size / 3; // Κάθε υπο-τετράγωνο είναι 1/3 του
τρέχοντος μεγέθους

    // ΒΗΜΑ 3: Διαίρεση του τετραγώνου σε 3x3 grid και επεξεργασία κάθε
κελιού
    for (let i = 0; i < 3; i++) {
      // Βρόχος για τις 3 στήλες
      for (let j = 0; j < 3; j++) {
        // Βρόχος για τις 3 γραμμές
        // Υπολογισμός θέσης του τρέχοντος υπο-τετραγώνου
        let newX = x + i * newSize;
        let newY = y + j * newSize;

        // ΒΗΜΑ 4: Έλεγχος αν βρισκόμαστε στο κεντρικό τετράγωνο
        if (i === 1 && j === 1) {
          // Κεντρικό τετράγωνο (θέση [1,1]) - σχεδιάζουμε λευκό
τετράγωνο
          fill(255); // Χρώμα λευκό
          noStroke(); // Χωρίς περίγραμμα
          rect(newX, newY, newSize, newSize); // Σχεδίαση τετραγώνου
        } else {
          // ΒΗΜΑ 5: Για τα υπόλοιπα 8 τετράγωνα - αναδρομική κλήση
          sierpinskiCarpet(newX, newY, newSize, depth + 1);
          // Η αναδρομή συνεχίζει με μικρότερο μέγεθος και αυξημένο
βάθος
        }
      }
    }
  }

  // ================================================================
  // FRACTAL GARDEN CITY ΑΛΓΟΡΙΘΜΟΣ
  // ================================================================
  function fractalGardenCity(x, y, size, depth) {
    // ΒΗΜΑ 1: Base case - όπως στον κλασικό αλγόριθμο
    if (size < minSize || depth >= currentDepth) {
      // Στο τέλος της αναδρομής δημιουργούμε σύνθετα αστικά τετράγωνα
      createUrbanBlock(x, y, size);
      return;
    }
```

```
        // ΒΗΜΑ 2: Δημιουργία πιο σύνθετης διαίρεσης του χώρου
        createComplexUrbanGrid(x, y, size, depth);
      }

      // Δημιουργία σύνθετου αστικού grid
      function createComplexUrbanGrid(x, y, size, depth) {
        let gridSize = size / 3; // Διαίρεση σε 3x3 όπως στον κλασικό

        // ΒΗΜΑ 1: Διαίρεση σε 3x3 grid με διαφορετική επεξεργασία για κάθε
τύπο κελιού
        for (let i = 0; i < 3; i++) {
          for (let j = 0; j < 3; j++) {
            let newX = x + i * gridSize;
            let newY = y + j * gridSize;

            if (i === 1 && j === 1) {
              // ΒΗΜΑ 2: Κεντρική πλατεία - πάντα λευκή (όπως στον κλασικό)
              fill(255);
              noStroke();
              rect(newX, newY, gridSize, gridSize);
            } else if (i === 1 || j === 1) {
              // ΒΗΜΑ 3: Δρόμοι (κεντρικές γραμμές/στήλες) - ειδική
επεξεργασία
              createRoadWithBuildings(newX, newY, gridSize, i === 1);
            } else {
              // ΒΗΜΑ 4: Γωνιακές γειτονιές - αναδρομική κλήση για
περαιτέρω subdivision
              fractalGardenCity(newX, newY, gridSize, depth + 1);
            }
          }
        }
      }

      // Δημιουργία δρόμων με κτίρια
      function createRoadWithBuildings(x, y, size, isHorizontal) {
        let segments = 5; // Διαίρεση του δρόμου σε 5 τμήματα
        let segmentSize = size / segments;

        // ΒΗΜΑ 1: Δημιουργία τμημάτων κατά μήκος του δρόμου
        for (let i = 0; i < segments; i++) {
          // Υπολογισμός θέσης και διαστάσεων ανάλογα με την κατεύθυνση
          let segX = isHorizontal ? x + i * segmentSize : x;
          let segY = isHorizontal ? y : y + i * segmentSize;
          let segW = isHorizontal ? segmentSize : size;
          let segH = isHorizontal ? size : segmentSize;

          // ΒΗΜΑ 2: Εναλλαγή μεταξύ κτιρίων και ανοιχτών χώρων
          if (i % 2 === 0) {
            // Κτίρια με εσωτερικές αυλές
            createBuildingWithCourtyard(segX, segY, segW, segH);
          } else {
            // Ανοιχτοί χώροι (αυλές/πάρκα)
            fill(255);
```

```
      noStroke();
      rect(segX, segY, segW, segH);
    }
  }
}

// Δημιουργία κτιρίου με εσωτερική αυλή
function createBuildingWithCourtyard(x, y, w, h) {
  // ΒΗΜΑ 1: Υπολογισμός διαστάσεων και θέσης εσωτερικής αυλής
  let courtyardSize = Math.min(w, h) / 3;
  let courtyardX = x + (w - courtyardSize) / 2; // Κεντράρισμα στον X
  let courtyardY = y + (h - courtyardSize) / 2; // Κεντράρισμα στον Y

  // ΒΗΜΑ 2: Σχεδίαση κτιρίου (μαύρο φόντο)
  fill(0);
  noStroke();
  rect(x, y, w, h);

  // ΒΗΜΑ 3: Σχεδίαση εσωτερικής αυλής (λευκή)
  fill(255);
  rect(courtyardX, courtyardY, courtyardSize, courtyardSize);
}

// ================================================================
// ΔΗΜΙΟΥΡΓΙΑ ΔΙΑΦΟΡΕΤΙΚΩΝ ΑΣΤΙΚΩΝ PATTERNS
// ================================================================

// Δημιουργία αστικού τετραγώνου με τυχαίο pattern
function createUrbanBlock(x, y, size) {
  // ΒΗΜΑ 1: Τυχαία επιλογή ενός από τα 4 διαφορετικά patterns
  let blockPattern = Math.floor(Math.random() * 4);

  // ΒΗΜΑ 2: Κλήση της κατάλληλης συνάρτησης ανάλογα με το pattern
  switch (blockPattern) {
    case 0:
      createLinearBlocks(x, y, size);
      break;
    case 1:
      createLShapeBlocks(x, y, size);
      break;
    case 2:
      createCheckerboardBlocks(x, y, size);
      break;
    case 3:
      createConcentricBlocks(x, y, size);
      break;
  }
}

// Pattern 1: Γραμμικές διατάξεις κτιρίων
function createLinearBlocks(x, y, size) {
  let strips = 4; // Διαίρεση σε 4 κάθετες λωρίδες
  let stripSize = size / strips;
```

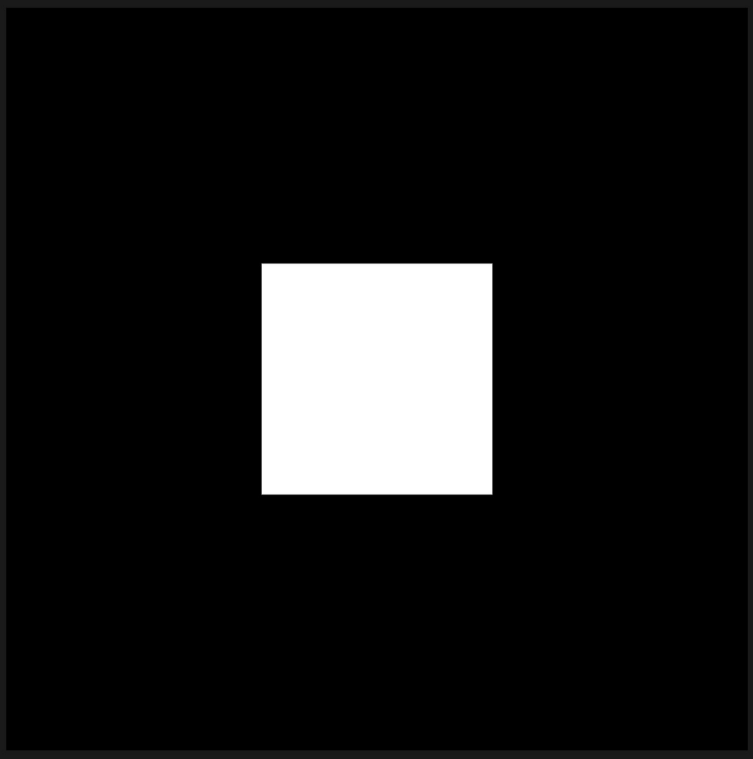
```
        for (let i = 0; i < strips; i++) {
          let stripX = x + i * stripSize;
          if (i % 2 === 0) {
            // Κτίριο με εσωτερική αυλή
            fill(0);
            rect(stripX, y, stripSize, size);
            // Μικρή εσωτερική αυλή
            fill(255);
            rect(stripX + stripSize / 4, y + size / 4, stripSize / 2, size
/ 2);
          } else {
            // Ανοιχτή αυλή
            fill(255);
            rect(stripX, y, stripSize, size);
          }
        }
      }

      // Pattern 2: L-shaped κτίρια
      function createLShapeBlocks(x, y, size) {
        let halfSize = size / 2;

        // Δημιουργία L-shaped κτιρίου
        fill(0);
        rect(x, y, halfSize, size); // Κάθετη μπάρα του L
        rect(x, y, size, halfSize); // Οριζόντια μπάρα του L

        // Εσωτερική αυλή στη γωνία του L
        fill(255);
        rect(x + halfSize, y + halfSize, halfSize, halfSize);

        // Μικρές διακοσμητικές αυλές
        fill(255);
        rect(x + halfSize / 4, y + halfSize / 4, halfSize / 4, halfSize /
4);
        rect(
          x + halfSize / 4,
          y + (3 * halfSize) / 2,
          halfSize / 4,
          halfSize / 4
        );
      }

      // Pattern 3: Σκακιέρα από κτίρια και αυλές
      function createCheckerboardBlocks(x, y, size) {
        let gridSize = 4; // 4x4 grid
        let cellSize = size / gridSize;

        for (let i = 0; i < gridSize; i++) {
          for (let j = 0; j < gridSize; j++) {
            let cellX = x + i * cellSize;
            let cellY = y + j * cellSize;

            // Δημιουργία σκακιέρας pattern
```

```
            if ((i + j) % 2 === 0) {
              fill(0); // Κτίριο
            } else {
              fill(255); // Αυλή
            }
            rect(cellX, cellY, cellSize, cellSize);
          }
        }
      }

      // Pattern 4: Ομόκεντρα τετράγωνα
      function createConcentricBlocks(x, y, size) {
        let layers = 3; // 3 στρώματα

        // Σχεδίαση από έξω προς τα μέσα
        for (let layer = 0; layer < layers; layer++) {
          let layerSize = size - (layer * size) / layers;
          let offset = (size - layerSize) / 2; // Κεντράρισμα

          // Εναλλαγή χρωμάτων ανά στρώμα
          if (layer % 2 === 0) {
            fill(0); // Κτίριο
          } else {
            fill(255); // Αυλή
          }
          rect(x + offset, y + offset, layerSize, layerSize);
        }
      }

      // ================================================================
      // UTILITY FUNCTIONS
      // ================================================================

      // Επανασχεδίαση όταν αλλάζει το μέγεθος του παραθύρου
      function windowResized() {
        resizeCanvas(600, 600);
        redraw();
      }
    </script>
  </body>
</html>
```

Sierpinski Carpet Fractal

Επίπεδα: 1

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

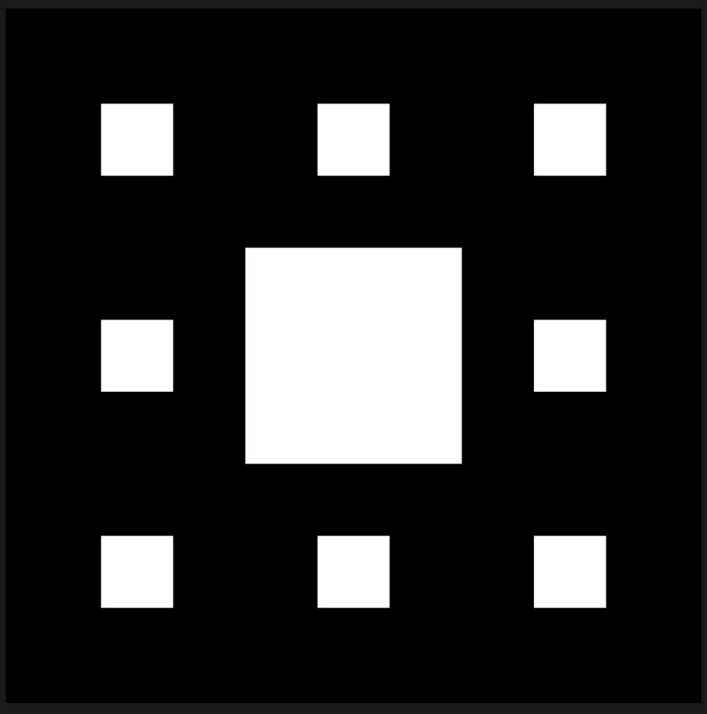
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 2

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

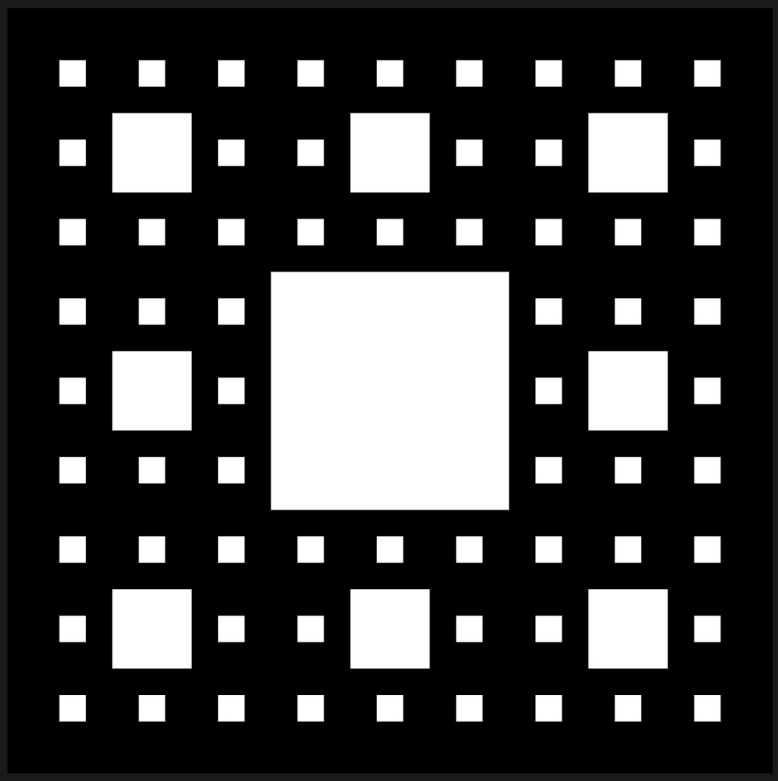
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 3

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

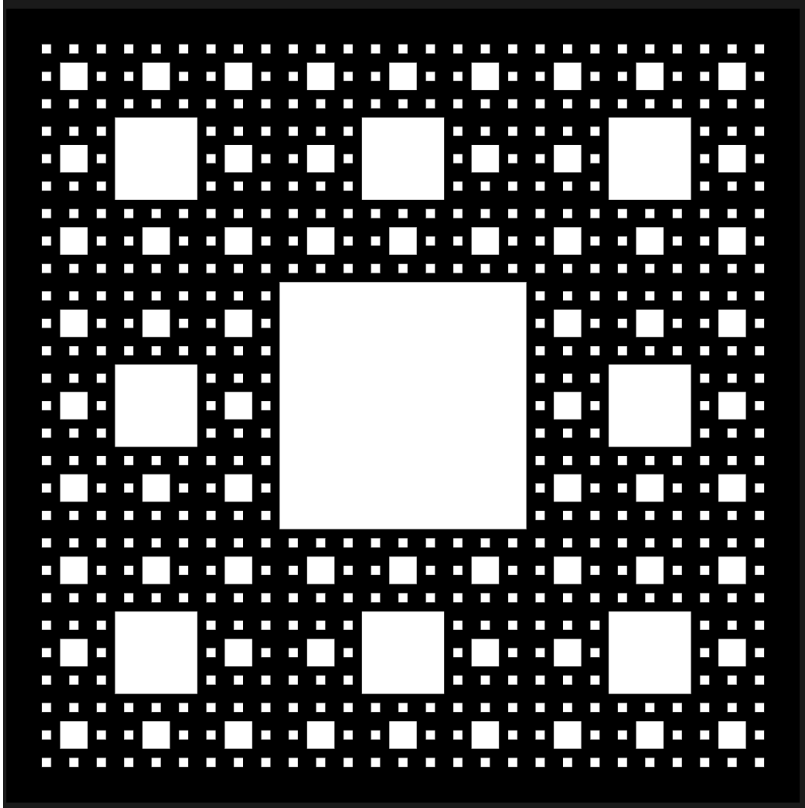
Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

/

Sierpinski Carpet Fractal

Επίπεδα: 4

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

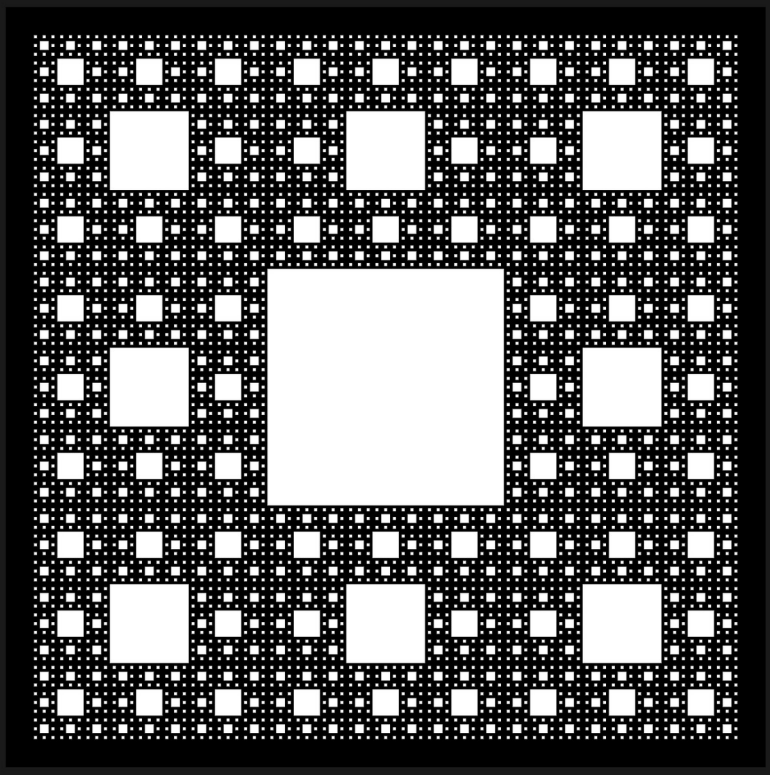
Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

/

Sierpinski Carpet Fractal

Επίπεδα: 5

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

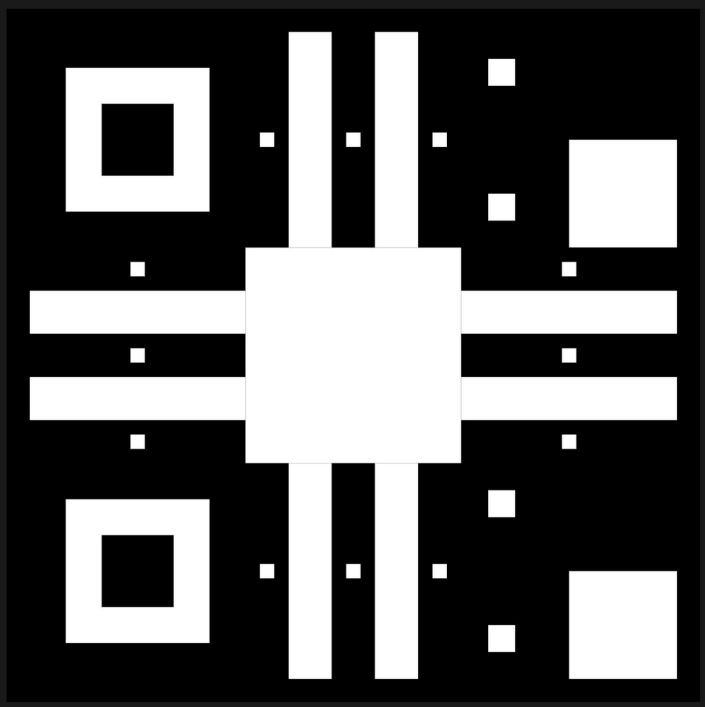
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 1

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

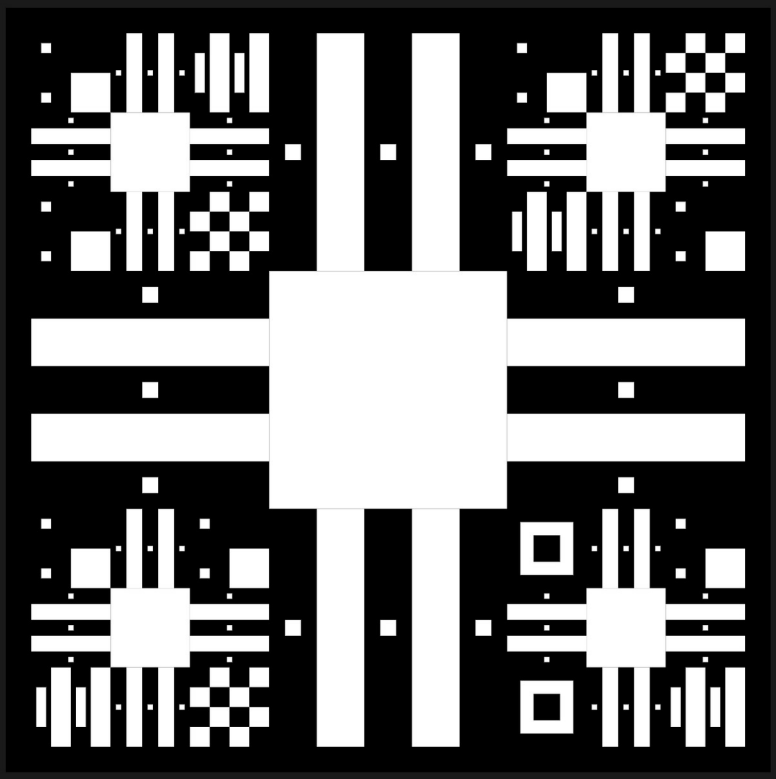
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 2

Κλασικό Sierpinski | **Fractal Garden City**



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

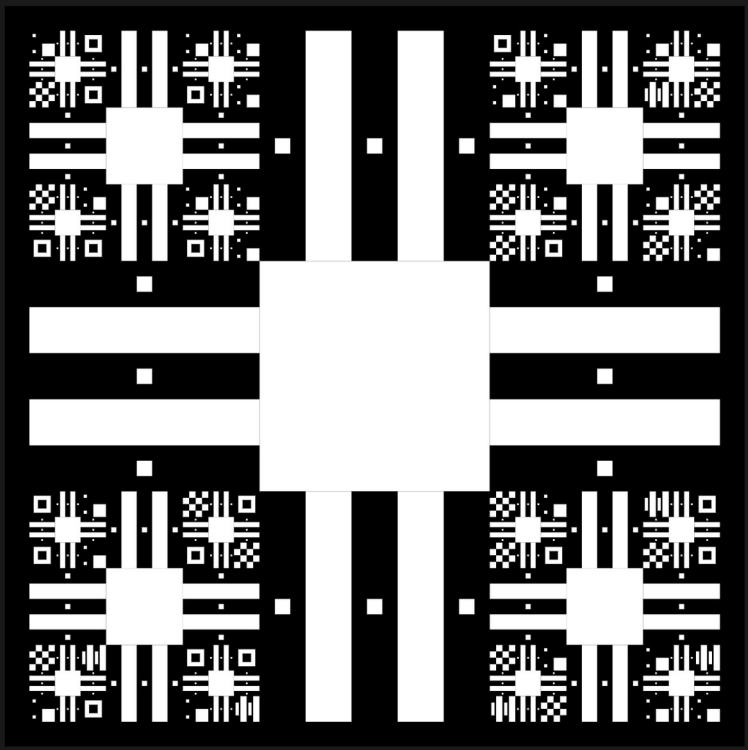
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 3

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

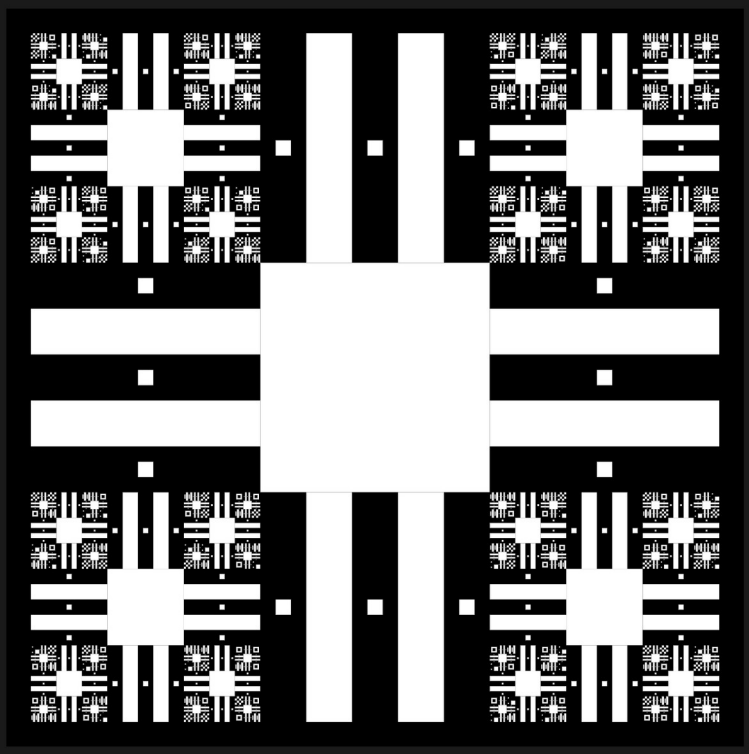
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 4

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

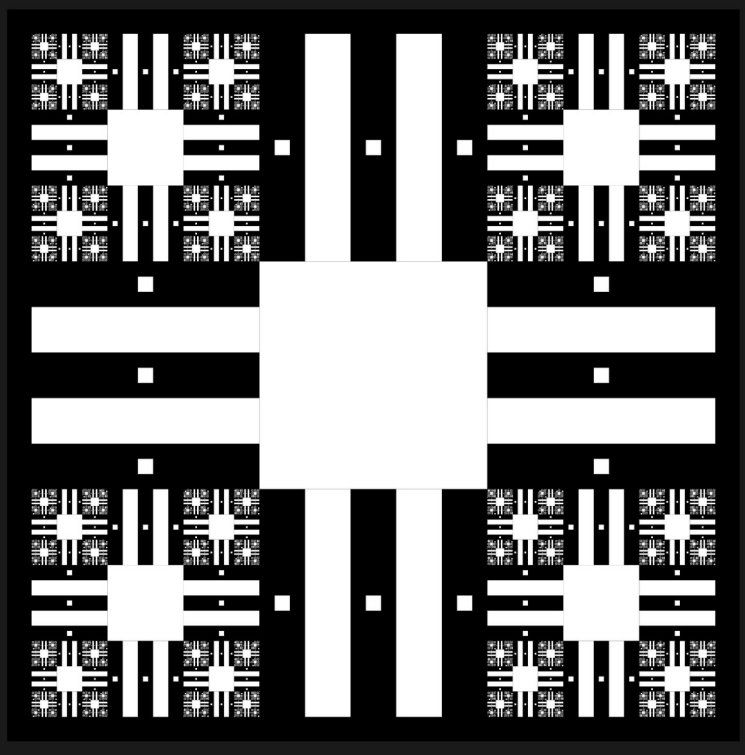
- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

Sierpinski Carpet Fractal

Επίπεδα: 5

Κλασικό Sierpinski | Fractal Garden City



Κλασικό Sierpinski

Fractal που δημιουργείται διαιρώντας ένα τετράγωνο σε 9 μικρότερα (3x3), αφαιρώντας το κεντρικό, και επαναλαμβάνοντας αναδρομικά για τα υπόλοιπα 8.

Fractal Garden City

Καινοτόμος αστικός αλγόριθμος με μόνο μαύρο και άσπρο:

- **Ποικίλα Blocks:** 4 διαφορετικά patterns κτιρίων
- **Linear:** Παράλληλες λωρίδες με αυλές
- **L-Shaped:** Κτίρια με γωνιακές αυλές
- **Checkerboard:** Σκακιέρα κτιρίων/χώρων
- **Concentric:** Στρώματα και εσωτερικές αυλές
- **Αντίθεση:** Καθαρό μαύρο & άσπρο

Η **εξελιγμένη Fractal Architecture** δημιουργεί πολύπλοκες αστικές δομές: ποικίλα κτίρια, πολυσχιδείς αυλές, και οργανική διάταξη που αλλάζει σε κάθε επίπεδο. Κάθε "γειτονιά" είναι μοναδική αλλά ακολουθεί το ίδιο βασικό μοτίβο οργάνωσης!

3. Computer Game - Φιδάκι

Επισκόπηση

Το Snake Game αποτελεί μια κλασική υλοποίηση του παραδοσιακού παιχνιδιού φιδιού με σύγχρονα χαρακτηριστικά. Το παιχνίδι αναπτύχθηκε με HTML5, CSS3 και τη βιβλιοθήκη p5.js για την γραφική αποτύπωση και διαχείριση του gameplay. Ο παίκτης ελέγχει ένα φίδι που κινείται σε πλέγμα, συλλέγει τροφή για να μεγαλώσει και αποφεύγει τα εμπόδια και τον εαυτό του.

Δομή Αρχείου

HTML Structure

```
<!DOCTYPE html>
<html lang="el">
  <head>
    - Meta tags για charset και viewport - Εισαγωγή p5.js από CDN - CSS
styling
  </head>
  <body>
    - Container για κεντράρισμα - Info panel με σκορ και ταχύτητα - Game
canvas
    container - Οδηγίες ελέγχου
  </body>
</html>
```

Global Variables

Θέση και Κίνηση Φιδιού

- `snakeX, snakeY`: Συντεταγμένες κεφαλιού φιδιού (pixels)
- `step`: Μέγεθος κελιού πλέγματος (20 pixels)
- `fwd_x, fwd_y`: Διανύσματα κίνησης (-step, 0, +step)
- `snakeBody[]`: Array με τα τμήματα του σώματος φιδιού
- `snakeLength`: Τρέχον μήκος φιδιού (αρχικά 3)

Στοιχεία Παιχνιδιού

- `food[]`: Array με θέσεις τροφής
- `totalFood`: Συνολικός αριθμός τροφής στο παιχνίδι (80)
- `buildings[]`: Array με εμπόδια-κτήρια
- `totalBuildings`: Αριθμός κτηρίων (6)

Game State Variables

- `score`: Βαθμολογία παίκτη
- `gameRunning`: Boolean για κατάσταση παιχνιδιού
- `frameCounter`: Μετρητής frames για έλεγχο ταχύτητας

- `gameSpeed`: Τρέχουσα ταχύτητα (frames ανά ενημέρωση)
- `baseSpeed`: Αρχική ταχύτητα (8 frames)

Κύριες Συναρτήσεις

`setup()`

Σκοπός: Αρχικοποίηση παιχνιδιού και δημιουργία αρχικών στοιχείων

Βήματα εκτέλεσης:

1. **Δημιουργία Canvas**: 800x600 pixels, προσάρτηση στο DOM
2. **Υπολογισμός αρχικής θέσης**: Κεντράρισμα φιδιού στο πλέγμα
3. **Αρχικοποίηση σώματος φιδιού**: Δημιουργία 3 τμημάτων σε οριζόντια γραμμή
4. **Δημιουργία στοιχείων**: Κλήση `createFood()` και `createBuildings()`
5. **Ενημέρωση UI**: Κλήση `updateScore()` για αρχικό UI

`createFood()`

Σκοπός: Δημιουργία και τοποθέτηση τροφής στο παιχνίδι

Αλγοριθμική Προσέγγιση:

1. **Εκκαθάριση υπάρχουσας τροφής**: `food = []`
2. **Loop δημιουργίας** (80 επαναλήψεις):

```
Επανάληψη μέχρι εύρεση έγκυρης θέσης:
  - Τυχαίες συντεταγμένες στο πλέγμα
  - Κεντράρισμα στο κελί (+step/2)
  - Έλεγχος μη-επικάλυψης με κτήρια
```

3. **Προσθήκη στο array**: `food.push({x, y})`

Validation Algorithm:

```
while (!validPosition) {
  // Τυχαία θέση στο πλέγμα
  foodX = floor(random(0, width/step)) * step + step/2
  foodY = floor(random(0, height/step)) * step + step/2

  // Έλεγχος για κάθε κτήριο
  for building in buildings:
    if food overlaps building:
      validPosition = false
}
```

`createBuildings()`

Σκοπός: Στατική δημιουργία εμποδίων

Prede ned Buildings:

- 6 ορθογώνια κτήρια σε συγκεκριμένες θέσεις
- Διαφορετικές διαστάσεις για ποικιλομορφία
- Στρατηγική τοποθέτηση για ισορροπημένο gameplay

`draw()`

Σκοπός: Κύριος βρόχος απεικόνισης και λογικής παιχνιδιού

Frame Rate Control:

```
frameCounter++;
if (frameCounter % gameSpeed !== 0) return;
```

Rendering Pipeline:

1. **Background και Κτήρια**:
   - Λευκό φόντο
   - Μαύρα γεμάτα ορθογώνια για κτήρια
2. **Food Management**:

```
Για κάθε τροφή (αντίστροφη σειρά):
  Έλεγχος συλλογής (distance collision)
  Αν συλλέχθηκε:
    - Αφαίρεση από array
    - Αύξηση μήκους φιδιού
    - Ενημέρωση σκορ (+10)
    - Υπολογισμός νέας ταχύτητας
    - Δημιουργία νέας τροφής
  Αλλιώς:
    - Σχεδίαση ως σκούρος κύκλος
```

3. **Snake Movement Logic**:

```
Υπολογισμός νέας θέσης κεφαλιού:
  newX = snakeX + fwd_x
  newY = snakeY + fwd_y

Screen Wrapping:
  Αν εκτός ορίων → wrapping στην αντίθετη πλευρά

Building Collision Detection:
  Για κάθε κτήριο:
```

```
    Αν επικάλυψη → fwd_x = fwd_y = 0 (στάση)

Αν όχι κολλίσιο:
  - Ενημέρωση θέσης κεφαλιού
  - Self-collision check
  - Body update (unshift/pop)
```

4. **Snake Rendering**:

```
Για κάθε τμήμα σώματος:
  - Εξωτερικό περίγραμμα (σκούρο)
  - Εσωτερικό γέμισμα (ανοιχτό γκρι)
  - Εσωτερικό περίγραμμα (segmented look)
  - Κεντρικό γέμισμα (πολύ ανοιχτό)
```

`keyPressed()`

Σκοπός: Διαχείριση εισόδου από πληκτρολόγιο

Input Handling:

1. **Restart Command**:

```
if (key === 'r' || key === 'R'):
  restartGame()
  return false  // Prevent browser default
```

2. **Directional Controls**:

```
Arrow Keys Mapping:
  UP_ARROW    → fwd_x=0, fwd_y=-step (αν δεν πάει κάτω)
  DOWN_ARROW  → fwd_x=0, fwd_y=+step (αν δεν πάει πάνω)
  LEFT_ARROW  → fwd_x=-step, fwd_y=0 (αν δεν πάει δεξιά)
  RIGHT_ARROW → fwd_x=+step, fwd_y=0 (αν δεν πάει αριστερά)
```

Anti-Reverse Logic:

- Παρεμπόδιση κίνησης προς την αντίθετη κατεύθυνση
- Αποφυγή άμεσης αυτο-κολλίσιου

`gameOver()`

Σκοπός: Διαχείριση τερματισμού παιχνιδιού

Actions:

1. `gameRunning = false`
2. Εμφάνιση "GAME OVER" κειμένου
3. Οδηγίες restart ("Press R to restart")

`restartGame()`

Σκοπός: Επαναφορά παιχνιδιού στην αρχική κατάσταση

Reset Sequence:

1. **Θέση φιδιού**: Επαναφορά στο κέντρο
2. **Κίνηση**: `fwd_x = fwd_y = 0`
3. **Σώμα**: Νέο array με 3 τμήματα
4. **Game state**: `score = 0`, `gameSpeed = baseSpeed`
5. **Περιβάλλον**: Νέα τροφή, ενημέρωση UI

`updateScore()`

Σκοπός: Ενημέρωση UI στοιχείων

Updates:

- **Score Display**: `document.getElementById('score').textContent`
- **Speed Level**: Υπολογισμός και εμφάνιση επιπέδου ταχύτητας

Gameplay Systems

Σύστημα Σκοράρισματος

- **+10 πόντοι** ανά συλλογή τροφής
- **Αυτόματη αύξηση μήκους** φιδιού
- **Αντικατάσταση τροφής** με νέα στη θέση της παλιάς

Σύστημα Ταχύτητας

```
Αρχική ταχύτητα: 8 frames ανά ενημέρωση
Κάθε 50 πόντοι: gameSpeed = max(2, baseSpeed - floor(score/50))
Ελάχιστη ταχύτητα: 2 frames ανά ενημέρωση
```

Progression Table:

- 0-49 πόντοι: 8 frames (αργό)
- 50-99 πόντοι: 7 frames
- 100-149 πόντοι: 6 frames
- ...
- 300+ πόντοι: 2 frames (πολύ γρήγορο)

Collision Detection Systems

1. Food Collection

```
distance(snakeHead, food) < step / 2;
```

- Χρήση της `dist()` συνάρτησης του p5.js
- Tolerance για εύκολη συλλογή

2. Building Collision

```
AABB Collision Detection:
if (newX + step > building.x &&
    newX < building.x + building.w &&
    newY + step > building.y &&
    newY < building.y + building.h)
```

- Axis-Aligned Bounding Box collision
- Πρόληψη κίνησης αντί game over

3. Self Collision

```
Για i από 1 έως snakeBody.length:
  Αν κεφάλι == σώμα[i]: gameOver()
```

- Έλεγχος μόνο κατά την κίνηση
- Skip του κεφαλιού (index 0)

Screen Wrapping System

```
Wraparound Logic:
if (newX < 0) newX = width - step
if (newX >= width) newX = 0
if (newY < 0) newY = height - step
if (newY >= height) newY = 0
```

Τεχνικές Λεπτομέρειες

Grid-Based Movement

- **Διακριτό πλέγμα**: 20x20 pixel κελιά
- **Ακέραιες συντεταγμένες**: Πάντα πολλαπλάσια του step
- **Συγχρονισμένη κίνηση**: Όλα τα στοιχεία στο ίδιο grid

Performance Optimizations

- **Frame limiting**: Έλεγχος ταχύτητας με modulo
- **Reverse iteration**: Για food array modi cations
- **Minimal DOM updates**: Μόνο όταν αλλάζει το σκορ

Memory Management

- **Array operations**: E cient unshift/pop για σώμα φιδιού
- **Object pooling**: Επαναχρησιμοποίηση food objects
- **No memory leaks**: Proper cleanup στο restart

Visual Design

Color Palette

- **Background**: Άσπρο (#FFFFFF)
- **Buildings**: Μαύρο (#000000)
- **Food**: Σκούρο καστανό (100, 50, 50 RGB)
- **Snake Body**: Multi-layered γκρι

Snake Rendering Style

```
4-Layer Rendering:
1. Outer border (σκούρο γκρι #505050)
2. Main fill (ανοιχτό γκρι #C8C8C8)
3. Inner border (μεσαίο γκρι #787878)
4. Core fill (πολύ ανοιχτό #F0F0F0)
```

UI Elements

- **Monospace font**: Για retro αισθητική
- **Centered layout**: Flexbox κεντράρισμα
- **Minimal interface**: Εστίαση στο gameplay

Appendix html le

```
<!DOCTYPE html>
<html lang="el">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Snake Game</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.7.0/p5.min.js">
</script>
    <style>
      body {
        margin: 0;
```

```
        padding: 0;
        background-color: #f0f0f0;
        display: flex;
        justify-content: center;
        align-items: center;
        min-height: 100vh;
        font-family: monospace;
        color: black;
      }
      .container {
        text-align: center;
      }
      .info {
        margin: 10px 0;
        font-size: 14px;
      }
    </style>
  </head>
  <body>
    <div class="container">
      <div class="info">
        Score: <span id="score">0</span> | Speed: <span id="speed">1</span>
      </div>
      <div id="gameCanvas"></div>
      <div class="info">Arrow keys to move | R to restart</div>
    </div>

    <script>
      /* ========================================
         ΒΗΜΑ 1: ΔΗΛΩΣΗ GLOBAL VARIABLES
         Ορίζουμε όλες τις μεταβλητές που θα χρειαστούμε
         για τη διαχείριση του παιχνιδιού
         ======================================== */

      // --- Μεταβλητές θέσης και κίνησης φιδιού ---
      let snakeX, snakeY; // Συντεταγμένες κεφαλιού φιδιού
      let step = 20; // Μέγεθος κάθε κελιού του πλέγματος (20x20 pixels)
      let fwd_x = 0,
        fwd_y = 0; // Διανύσματα κίνησης (-step, 0, ή +step)
      let snakeBody = []; // Array που περιέχει όλα τα τμήματα του σώματος
      let snakeLength = 3; // Αρχικό μήκος φιδιού (3 τμήματα)

      // --- Στοιχεία παιχνιδιού (τροφή και εμπόδια) ---
      let food = []; // Array με όλες τις θέσεις τροφής
      let totalFood = 80; // Συνολικός αριθμός κομματιών τροφής στο
παιχνίδι
      let buildings = []; // Array με τα εμπόδια-κτήρια
      let totalBuildings = 6; // Αριθμός κτηρίων που θα δημιουργηθούν

      // --- Game state variables ---
      let score = 0; // Βαθμολογία παίκτη
      let gameRunning = true; // Boolean: true = παιχνίδι τρέχει, false =
game over
      let frameCounter = 0; // Μετρητής frames για έλεγχο ταχύτητας
```

```
      let gameSpeed = 8; // Τρέχουσα ταχύτητα (πόσα frames ανά ενημέρωση)
      let baseSpeed = 8; // Αρχική ταχύτητα παιχνιδιού

      /* ======================================
         ΒΗΜΑ 2: ΑΡΧΙΚΟΠΟΙΗΣΗ ΠΑΙΧΝΙΔΙΟΥ (setup)
         Δημιουργούμε το canvas και ετοιμάζουμε όλα
         τα στοιχεία για την έναρξη του παιχνιδιού
         ====================================== */
      function setup() {
        // Δημιουργία canvas 800x600 και προσάρτηση στο DOM
        let canvas = createCanvas(800, 600);
        canvas.parent("gameCanvas");

        // Υπολογισμός αρχικής θέσης φιδιού (κεντράρισμα στο πλέγμα)
        // Διαιρούμε τις διαστάσεις με το step και στρογγυλοποιούμε για
ακέραιες θέσεις
        snakeX = Math.floor(width / 2 / step) * step;
        snakeY = Math.floor(height / 2 / step) * step;

        // Δημιουργία αρχικού σώματος φιδιού (3 τμήματα σε οριζόντια
γραμμή)
        for (let i = 0; i < snakeLength; i++) {
          // Κάθε τμήμα τοποθετείται αριστερά από το προηγούμενο
          snakeBody.push({ x: snakeX - i * step, y: snakeY });
        }

        // Δημιουργία στοιχείων παιχνιδιού
        createFood(); // Τοποθέτηση τροφής
        createBuildings(); // Δημιουργία εμποδίων
        updateScore(); // Ενημέρωση UI με αρχικό σκορ
      }

      /* ======================================
         ΒΗΜΑ 3: ΔΗΜΙΟΥΡΓΙΑ ΤΡΟΦΗΣ (createFood)
         Τοποθετούμε τυχαία τροφή στο πλέγμα,
         αποφεύγοντας τα κτήρια
         ====================================== */
      function createFood() {
        food = []; // Εκκαθάριση υπάρχουσας τροφής

        // Δημιουργία 80 κομματιών τροφής
        for (let i = 0; i < totalFood; i++) {
          let foodX, foodY;
          let validPosition = false;

          // Loop μέχρι να βρούμε έγκυρη θέση (όχι πάνω σε κτήριο)
          while (!validPosition) {
            // Τυχαίες συντεταγμένες στο πλέγμα
            foodX = Math.floor(random(0, width / step)) * step + step / 2;
            foodY = Math.floor(random(0, height / step)) * step + step / 2;

            // Αρχικά θεωρούμε τη θέση έγκυρη
            validPosition = true;
```

```
            // Έλεγχος για επικάλυψη με κάθε κτήριο
            for (let building of buildings) {
              if (
                foodX >= building.x &&
                foodX < building.x + building.w &&
                foodY >= building.y &&
                foodY < building.y + building.h
              ) {
                // Αν η τροφή βρίσκεται μέσα σε κτήριο, η θέση δεν είναι
έγκυρη
                validPosition = false;
                break; // Διακοπή ελέγχου, δοκιμή νέας θέσης
              }
            }
          }

          // Προσθήκη έγκυρης θέσης τροφής στο array
          food.push({ x: foodX, y: foodY });
        }
      }

      /* ========================================
         ΒΗΜΑ 4: ΔΗΜΙΟΥΡΓΙΑ ΚΤΗΡΙΩΝ (createBuildings)
         Δημιουργούμε στατικά εμπόδια σε προκαθορισμένες θέσεις
         ======================================== */
      function createBuildings() {
        buildings = []; // Εκκαθάριση υπάρχοντων κτηρίων

        // Δημιουργία 6 ορθογώνιων κτηρίων με διαφορετικές διαστάσεις
        // Κάθε κτήριο ορίζεται από: {x, y, width, height}
        buildings.push({ x: 60, y: 40, w: 80, h: 60 }); // Πάνω αριστερά
        buildings.push({ x: 320, y: 140, w: 60, h: 40 }); // Κέντρο πάνω
        buildings.push({ x: 140, y: 480, w: 40, h: 40 }); // Κάτω αριστερά
        buildings.push({ x: 600, y: 200, w: 100, h: 80 }); // Δεξιά κέντρο
        buildings.push({ x: 480, y: 400, w: 60, h: 100 }); // Κάτω δεξιά
        buildings.push({ x: 200, y: 300, w: 120, h: 40 }); // Κέντρο
      }

      /* ========================================
         ΒΗΜΑ 5: ΚΥΡΙΟΣ ΒΡΟΧΟΣ ΠΑΙΧΝΙΔΙΟΥ (draw)
         Εδώ τρέχει όλη η λογική του παιχνιδιού:
         - Έλεγχος ταχύτητας
         - Απεικόνιση στοιχείων
         - Κίνηση φιδιού
         - Collision detection
         ======================================== */
      function draw() {
        frameCounter++; // Αύξηση μετρητή frames

        // ΕΛΕΓΧΟΣ ΤΑΧΥΤΗΤΑΣ: Ενημέρωση μόνο κάθε gameSpeed frames
        if (frameCounter % gameSpeed !== 0) {
          return; // Παράλειψη αυτού του frame
        }
```

```
        // Αν το παιχνίδι έχει σταματήσει, δεν κάνουμε τίποτα
        if (!gameRunning) {
          return;
        }

        // ΒΗΜΑ 5.1: ΑΠΕΙΚΟΝΙΣΗ ΦΟΝΤΟΥ ΚΑΙ ΚΤΗΡΙΩΝ
        background(255); // Λευκό φόντο

        // Σχεδίαση κτηρίων ως μαύρα ορθογώνια
        fill(0); // Μαύρο χρώμα
        noStroke(); // Χωρίς περίγραμμα
        for (let building of buildings) {
          rect(building.x, building.y, building.w, building.h);
        }

        // ΒΗΜΑ 5.2: ΔΙΑΧΕΙΡΙΣΗ ΤΡΟΦΗΣ
        fill(100, 50, 50); // Σκούρο καστανό χρώμα για τροφή
        noStroke();

        // Έλεγχος κάθε κομματιού τροφής (αντίστροφη σειρά για ασφαλή
αφαίρεση)
        for (let i = food.length - 1; i >= 0; i--) {
          let f = food[i];

          // COLLISION DETECTION: Έλεγχος αν το φίδι άγγιξε την τροφή
          if (dist(snakeX + step / 2, snakeY + step / 2, f.x, f.y) < step /
2) {
            // ΣΥΛΛΟΓΗ ΤΡΟΦΗΣ - Ενέργειες:
            food.splice(i, 1); // Αφαίρεση τροφής από το array
            snakeLength++; // Αύξηση μήκους φιδιού
            score += 10; // Προσθήκη 10 πόντων

            // ΑΥΞΗΣΗ ΤΑΧΥΤΗΤΑΣ: Κάθε 50 πόντοι μειώνουμε το gameSpeed
            gameSpeed = Math.max(2, baseSpeed - Math.floor(score / 50));

            updateScore(); // Ενημέρωση UI

            // ΔΗΜΙΟΥΡΓΙΑ ΝΕΑΣ ΤΡΟΦΗΣ για αντικατάσταση
            let newFoodX, newFoodY;
            let validPosition = false;

            // Εύρεση έγκυρης θέσης για νέα τροφή (όπως στην createFood)
            while (!validPosition) {
              newFoodX = Math.floor(random(0, width / step)) * step + step
/ 2;
              newFoodY = Math.floor(random(0, height / step)) * step + step
/ 2;

              validPosition = true;
              for (let building of buildings) {
                if (
                  newFoodX >= building.x &&
                  newFoodX < building.x + building.w &&
                  newFoodY >= building.y &&
```

```
                newFoodY < building.y + building.h
              ) {
                validPosition = false;
                break;
              }
            }
          }

          food.push({ x: newFoodX, y: newFoodY }); // Προσθήκη νέας
τροφής
          continue; // Συνέχεια με την επόμενη τροφή
        }

        // Σχεδίαση τροφής ως μικρός κύκλος
        circle(f.x, f.y, 4);
      }

      // ΒΗΜΑ 5.3: ΚΙΝΗΣΗ ΦΙΔΙΟΥ
      // Υπολογισμός νέας θέσης κεφαλιού
      let newX = snakeX + fwd_x;
      let newY = snakeY + fwd_y;

      // SCREEN WRAPPING: Αν βγει εκτός ορίων, εμφάνιση στην αντίθετη
πλευρά
      if (newX < 0) newX = width - step; // Αριστερά → δεξιά
      if (newX >= width) newX = 0; // Δεξιά → αριστερά
      if (newY < 0) newY = height - step; // Πάνω → κάτω
      if (newY >= height) newY = 0; // Κάτω → πάνω

      // ΒΗΜΑ 5.4: BUILDING COLLISION DETECTION
      let collisionDetected = false;
      for (let building of buildings) {
        // AABB (Axis-Aligned Bounding Box) collision detection
        if (
          newX + step > building.x && // Δεξιά άκρη φιδιού > αριστερή
άκρη κτηρίου
          newX < building.x + building.w && // Αριστερή άκρη φιδιού <
δεξιά άκρη κτηρίου
          newY + step > building.y && // Κάτω άκρη φιδιού > πάνω άκρη
κτηρίου
          newY < building.y + building.h // Πάνω άκρη φιδιού < κάτω άκρη
κτηρίου
        ) {
          // ΑΝΤΙΔΡΑΣΗ ΣΕ ΚΟΛΛΙΣΗ: Στάση φιδιού (μηδενισμός κίνησης)
          fwd_x = 0;
          fwd_y = 0;
          collisionDetected = true;
          break; // Διακοπή ελέγχου, βρέθηκε κολλίση
        }
      }

      // ΒΗΜΑ 5.5: ΕΝΗΜΕΡΩΣΗ ΘΕΣΗΣ ΚΑΙ ΣΩΜΑΤΟΣ ΦΙΔΙΟΥ
      if (!collisionDetected) {
        // Ενημέρωση θέσης κεφαλιού μόνο αν δεν υπάρχει κολλίση
```

```
    snakeX = newX;
    snakeY = newY;

    // SELF-COLLISION DETECTION: Έλεγχος αν το κεφάλι άγγιξε το σώμα
    for (let i = 1; i < snakeBody.length; i++) {
      if (snakeX === snakeBody[i].x && snakeY === snakeBody[i].y) {
        gameOver(); // Τερματισμός παιχνιδιού
        return;
      }
    }

    // ΕΝΗΜΕΡΩΣΗ ΣΩΜΑΤΟΣ: Μόνο αν το φίδι κινείται
    if (fwd_x !== 0 || fwd_y !== 0) {
      // Προσθήκη νέου κεφαλιού στην αρχή του array
      snakeBody.unshift({ x: snakeX, y: snakeY });

      // Αφαίρεση ουράς αν το μήκος ξεπερνά το επιθυμητό
      if (snakeBody.length > snakeLength) {
        snakeBody.pop();
      }
    }
  }

  // ΒΗΜΑ 5.6: ΑΠΕΙΚΟΝΙΣΗ ΦΙΔΙΟΥ
  // Σχεδίαση με 4-layer στυλ για 3D εφέ
  for (let i = 0; i < snakeBody.length; i++) {
    let segment = snakeBody[i];

    // Layer 1: Εξωτερικό περίγραμμα (σκούρο γκρι)
    fill(80);
    noStroke();
    rect(segment.x, segment.y, step, step);

    // Layer 2: Κύριο γέμισμα (ανοιχτό γκρι)
    fill(200);
    rect(segment.x + 2, segment.y + 2, step - 4, step - 4);

    // Layer 3: Εσωτερικό περίγραμμα (μεσαίο γκρι)
    fill(120);
    rect(segment.x + 4, segment.y + 4, step - 8, step - 8);

    // Layer 4: Κεντρικό γέμισμα (πολύ ανοιχτό γκρι)
    fill(240);
    rect(segment.x + 6, segment.y + 6, step - 12, step - 12);
  }
}

/* ========================================
   ΒΗΜΑ 6: ΔΙΑΧΕΙΡΙΣΗ ΕΙΣΟΔΟΥ (keyPressed)
   Έλεγχος πληκτρολογίου για κίνηση και restart
   ======================================== */
function keyPressed() {
  // RESTART COMMAND: R key
  if (key === "r" || key === "R") {
```

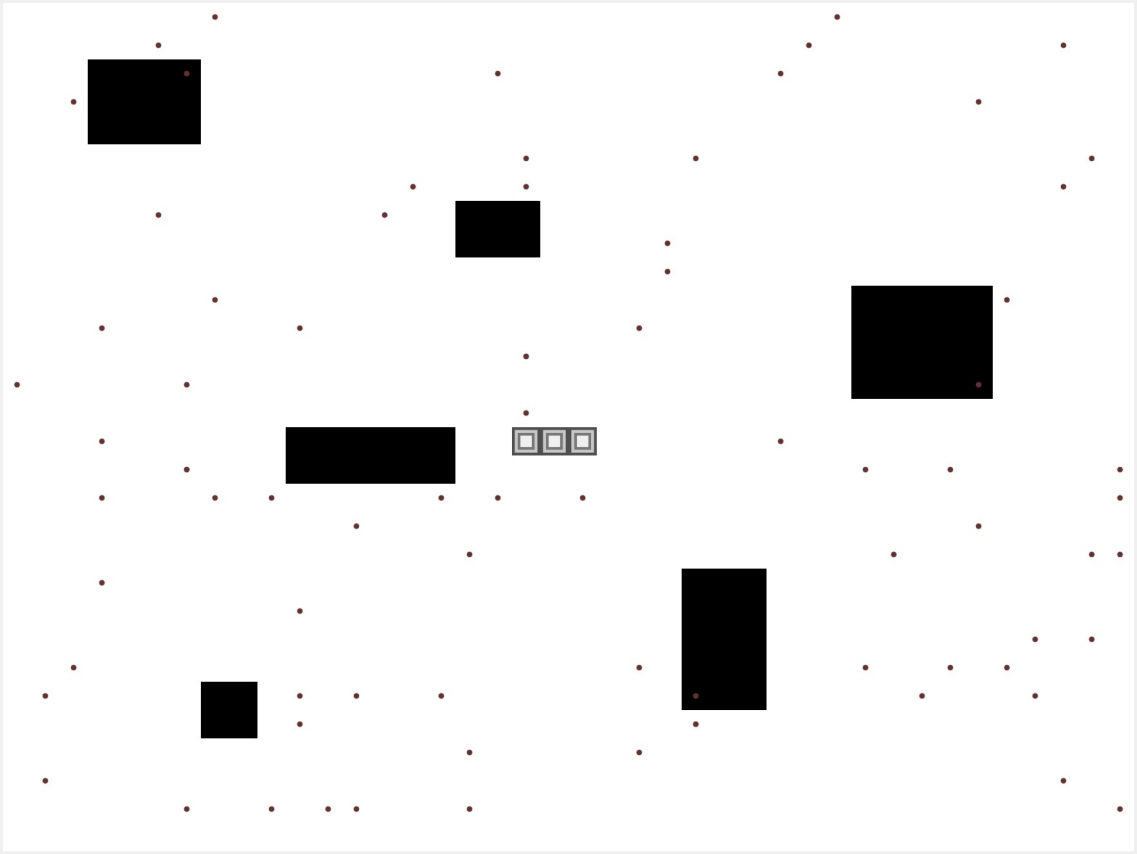
/

```
    restartGame();
    return false; // Αποτροπή default browser behavior
  }

  // Αν το παιχνίδι δεν τρέχει, αγνόηση άλλων πλήκτρων
  if (!gameRunning) return;

  // DIRECTIONAL CONTROLS: Arrow keys με anti-reverse logic
  if (keyCode === UP_ARROW) {
    // Κίνηση προς τα πάνω, αλλά όχι αν ήδη πάει κάτω
    if (fwd_y !== step) {
      fwd_x = 0;
      fwd_y = -step;
    }
    return false;
  } else if (keyCode === DOWN_ARROW) {
    // Κίνηση προς τα κάτω, αλλά όχι αν ήδη πάει πάνω
    if (fwd_y !== -step) {
      fwd_x = 0;
      fwd_y = step;
    }
    return false;
  } else if (keyCode === LEFT_ARROW) {
    // Κίνηση προς τα αριστερά, αλλά όχι αν ήδη πάει δεξιά
    if (fwd_x !== step) {
      fwd_x = -step;
      fwd_y = 0;
    }
    return false;
  } else if (keyCode === RIGHT_ARROW) {
    // Κίνηση προς τα δεξιά, αλλά όχι αν ήδη πάει αριστερά
    if (fwd_x !== -step) {
      fwd_x = step;
      fwd_y = 0;
    }
    return false;
  }
}

/* ========================================
   ΒΗΜΑ 7: ΤΕΡΜΑΤΙΣΜΟΣ ΠΑΙΧΝΙΔΙΟΥ (gameOver)
   Στάση παιχνιδιού και εμφάνιση μηνύματος
   ======================================== */
function gameOver() {
  gameRunning = false; // Στάση του game loop

  // Εμφάνιση "GAME OVER" μηνύματος
  fill(0); // Μαύρο κείμενο
  textAlign(CENTER); // Κεντραρισμένο κείμενο
  textSize(32); // Μεγάλο μέγεθος γραμματοσειράς
  text("GAME OVER", width / 2, height / 2);

  // Οδηγίες restart
  textSize(16); // Μικρότερο μέγεθος
```

/

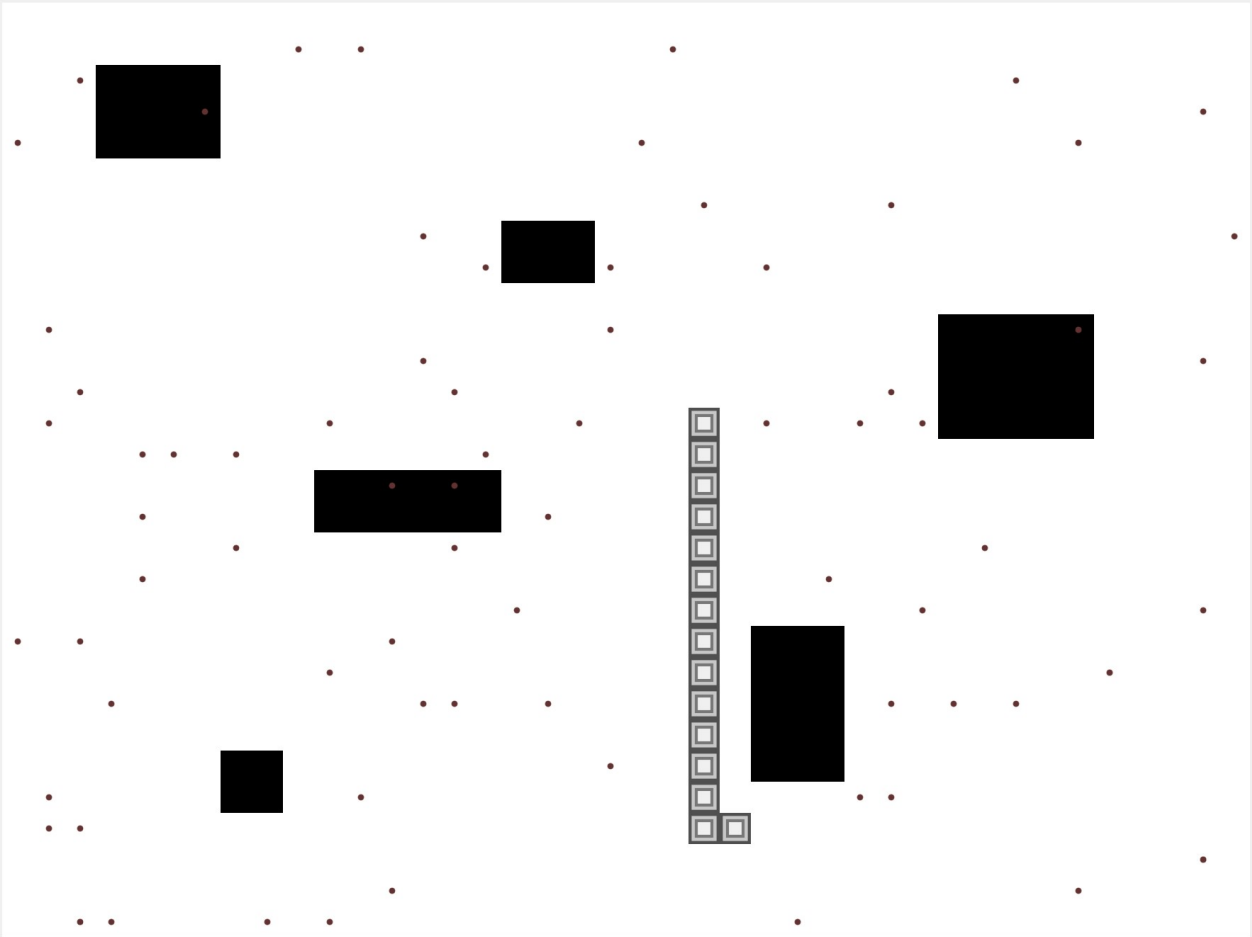
```
        text("Press R to restart", width / 2, height / 2 + 30);
      }

      /* ======================================
         ΒΗΜΑ 8: ΕΠΑΝΑΦΟΡΑ ΠΑΙΧΝΙΔΙΟΥ (restartGame)
         Reset όλων των μεταβλητών στις αρχικές τιμές
         ====================================== */
      function restartGame() {
        // Επαναφορά θέσης φιδιού στο κέντρο
        snakeX = Math.floor(width / 2 / step) * step;
        snakeY = Math.floor(height / 2 / step) * step;

        // Μηδενισμός κίνησης
        fwd_x = 0;
        fwd_y = 0;

        // Reset σώματος φιδιού
        snakeBody = [];
        snakeLength = 3;

        // Reset game state
        score = 0;
        gameSpeed = baseSpeed; // Επαναφορά στην αρχική ταχύτητα
        gameRunning = true; // Επανεκκίνηση παιχνιδιού

        // Δημιουργία νέου αρχικού σώματος
        for (let i = 0; i < snakeLength; i++) {
          snakeBody.push({ x: snakeX - i * step, y: snakeY });
        }

        // Αναδημιουργία τροφής και ενημέρωση UI
        createFood();
        updateScore();
      }

      /* ======================================
         ΒΗΜΑ 9: ΕΝΗΜΕΡΩΣΗ UI (updateScore)
         Ενημέρωση των στοιχείων σκορ και ταχύτητας
         ====================================== */
      function updateScore() {
        // Ενημέρωση σκορ στο DOM
        document.getElementById("score").textContent = score;

        // Υπολογισμός και εμφάνιση επιπέδου ταχύτητας
        let speedLevel = Math.floor(baseSpeed - gameSpeed + 1);
        document.getElementById("speed").textContent = speedLevel;
      }
    </script>
  </body>
</html>
```

Score: 0 | Speed: 1



Arrow keys to move | R to restart

Score: 120 | Speed: 3

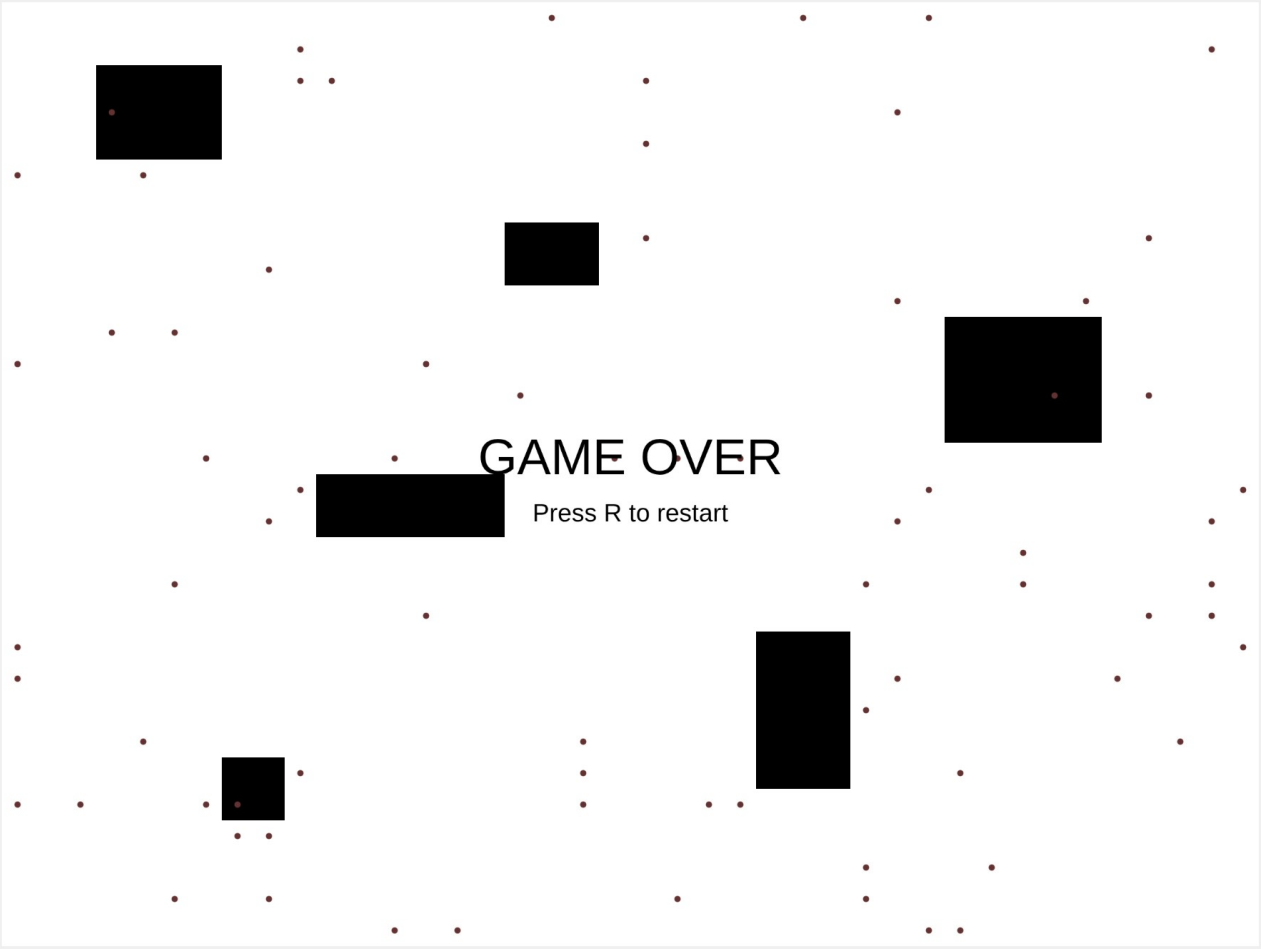


/

Arrow keys to move | R to restart

Arrow keys to move | R to restart

Score: 140 | Speed: 3



Arrow keys to move | R to restart