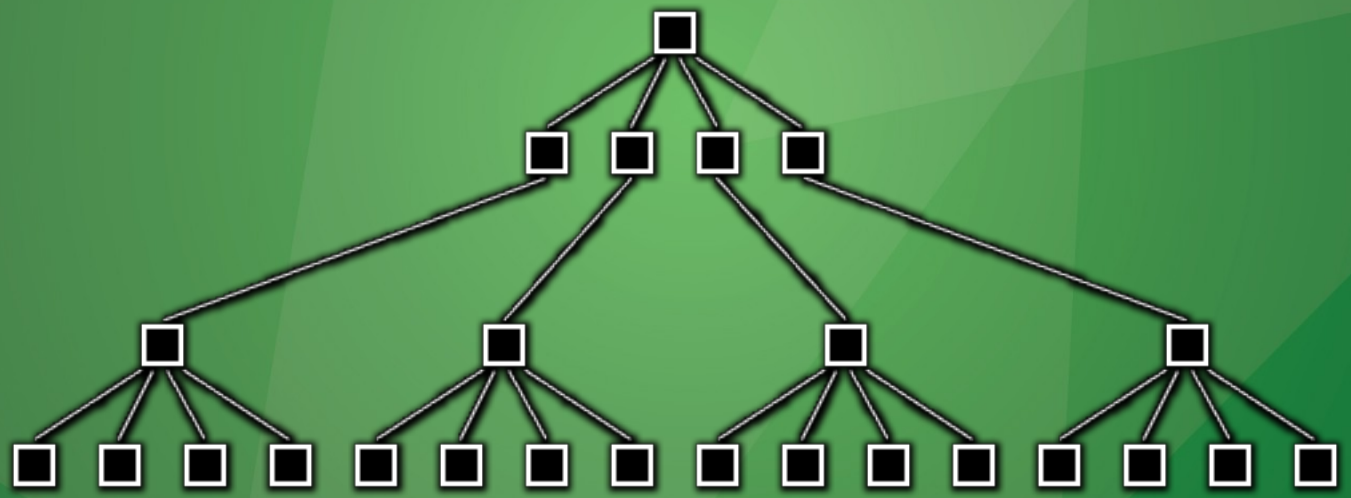
Quadfák



Kiss Krisztián, 512.

Mi az a quadfa?

Mi az a quadfa?

- **Quad** → négy

Mi az a quadfa?

- **Quad** → négy
- **Fa** → olyan adatszerkezet, amely csomópontok hierarchiájából áll

Mi az a quadfa?

- **Quad** → négy
- **Fa** → olyan adatszerkezet, amely csomópontok hierarchiájából áll
  - → mint a bináris fa

Mi az a quadfa?

- **Quad** → négy
- **Fa** → olyan adatszerkezet, amely csomópontok hierarchiájából áll
  - → mint a bináris fa
- *Egy quadfa vagy üres, vagy van 4 leszármazottja, amik szintén quadfák.*

Mi az a quadfa?

- **Quad** → négy
- **Fa** → olyan adatszerkezet, amely csomópontok hierarchiájából áll
  - → mint a bináris fa
- *Egy quadfa vagy üres, vagy van 4 leszármazottja, amik szintén quadfák.*
- *Egyéb elnevezések:* magyarul négyágú fa, angolul quadtree.

Mire jó egy quadfa?

Mire jó egy quadfa?

- Kétdimenziós adatok tárolására
  - pl. pontok, alakzatok, területek, görbék

Mire jó egy quadfa?

- Kétdimenziós adatok tárolására
  - pl. pontok, alakzatok, területek, görbék
- Gyors beszúrás/törlés/lekérdezés műveletek megvalósítására ezekkel

Mit ábrázolnak a quadfa csomópontjai?

Mit ábrázolnak a quadfa csomópontjai?

- A kétdimenziós *tér* egy részét fedik le

Mit ábrázolnak a quadfa csomópontjai?

- A kétdimenziós *tér* egy részét fedik le
- A gyökér az adott teret teljességében

Mit ábrázolnak a quadfa csomópontjai?

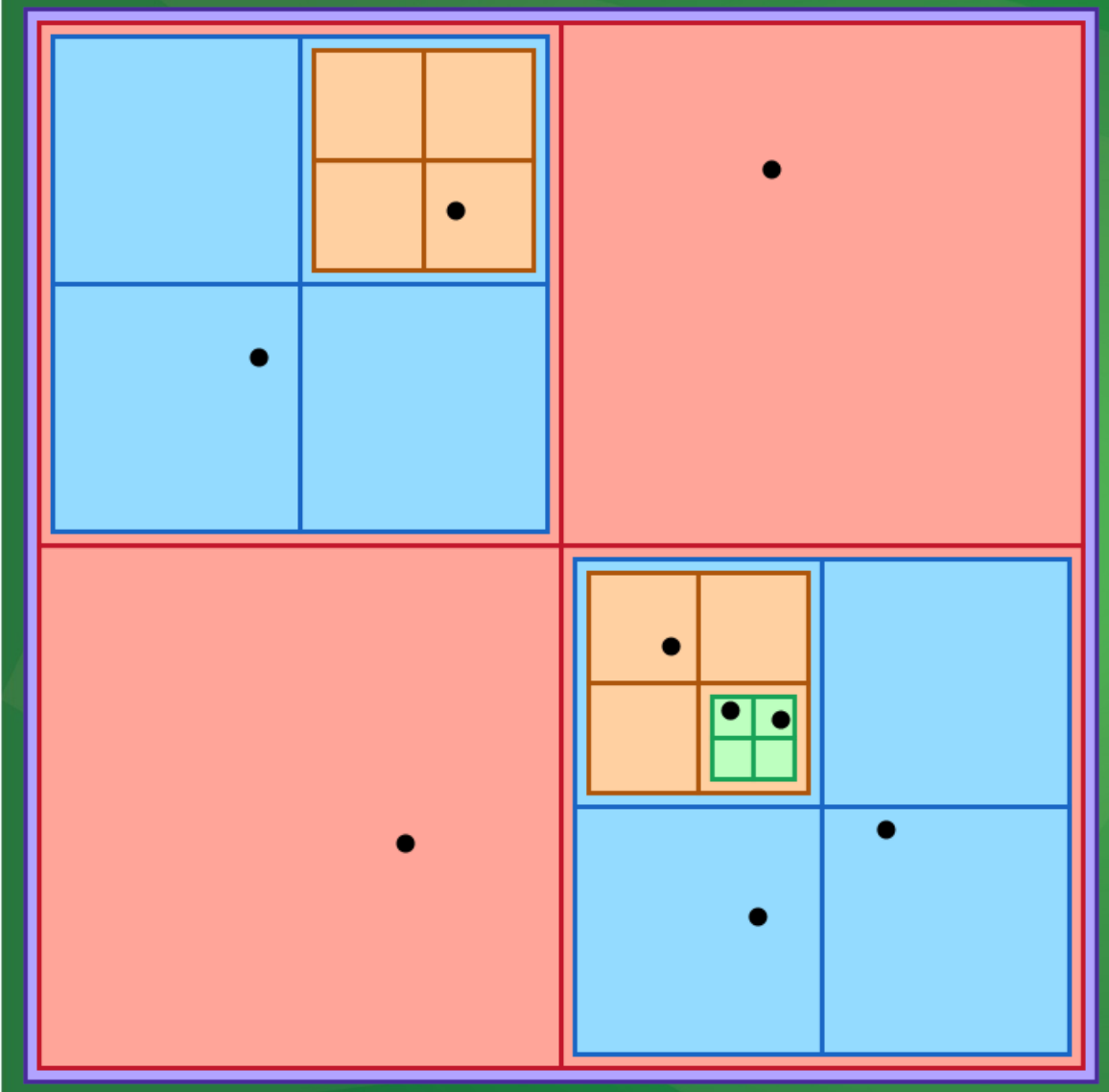
- A kétdimenziós *tér* egy részét fedik le
- A gyökér az adott teret teljességében
- A többi csomópont a szülő területének egy-egy *egyenlő* negyedét → quadrant

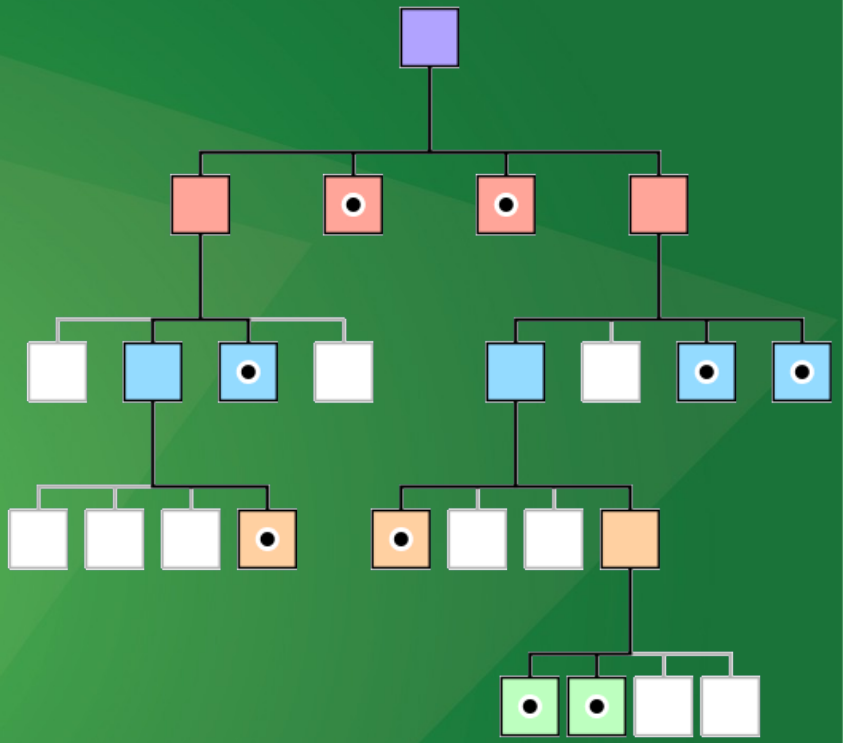
Mit ábrázolnak a quadfa csomópontjai?

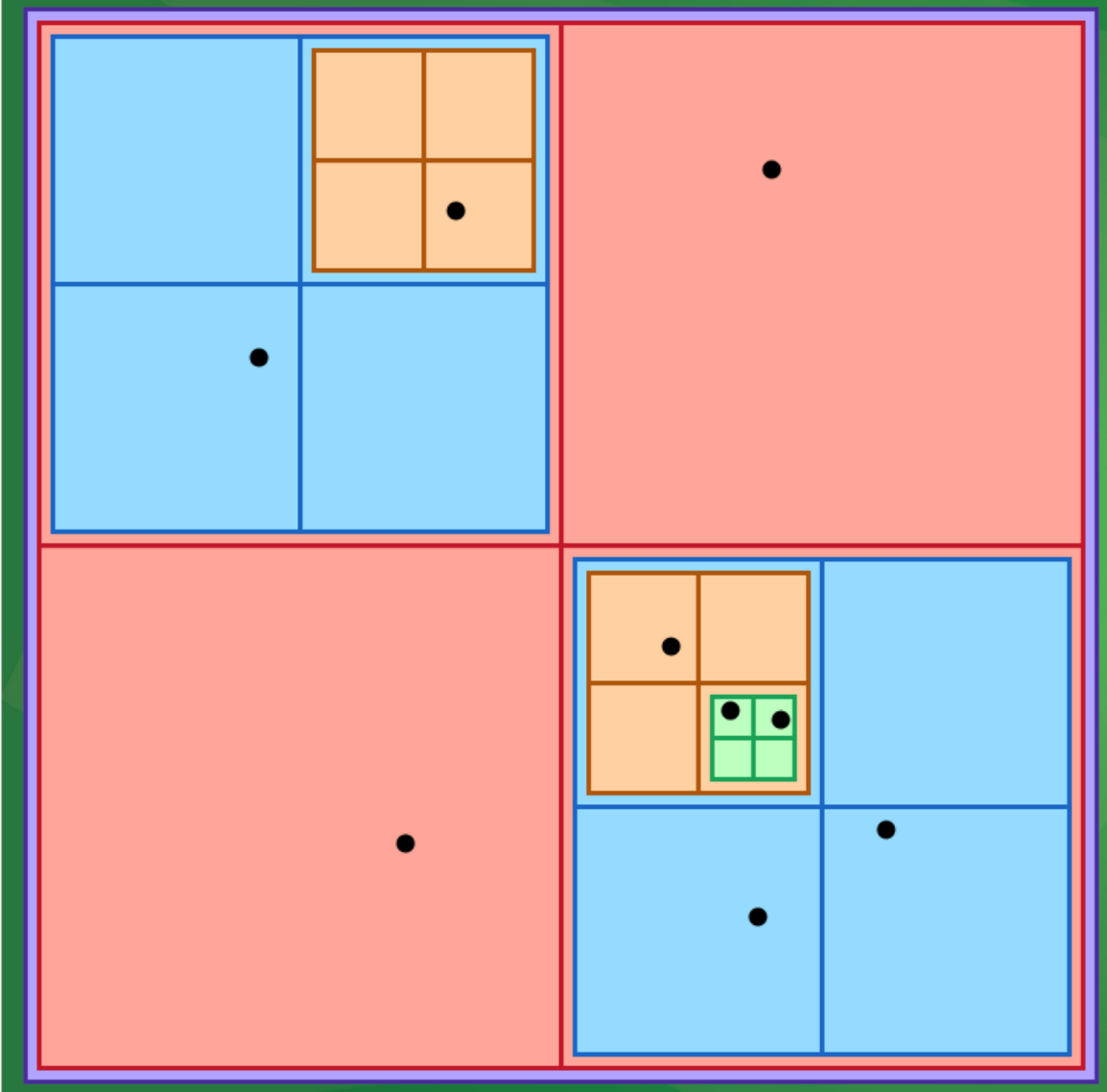
- A kétdimenziós *tér* egy részét fedik le
- A gyökér az adott teret teljességében
- A többi csomópont a szülő területének egy-egy *egyenlő* negyedét → quadrant
- A csomópontok tartalmazzák az általuk lefedett területen fekvő objektum (pont, alakzat, stb.) információit
  - koordináta/koordináták
  - méret, név, szín, stb.

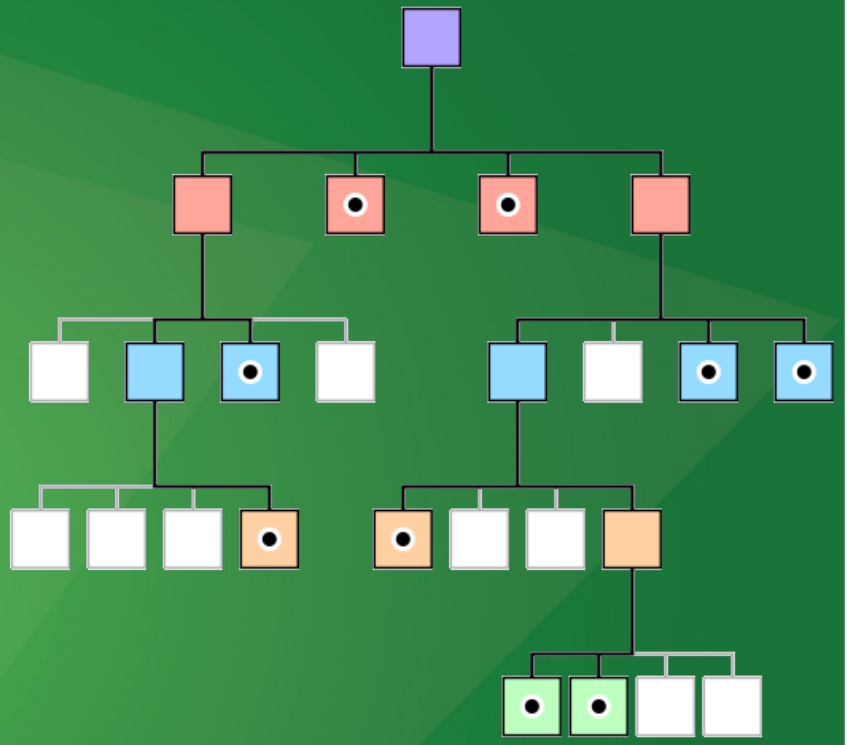
Mit ábrázolnak a quadfa csomópontjai?

- A kétdimenziós *tér* egy részét fedik le
- A gyökér az adott teret teljességében
- A többi csomópont a szülő területének egy-egy *egyenlő* negyedét → quadrant
- A csomópontok tartalmazzák az általuk lefedett területen fekvő objektum (pont, alakzat, stb.) információit
  - koordináta/koordináták
  - méret, név, szín, stb.
- Ahogy bináris fák esetén megkülönböztetjük a bal és jobb utódot, itt általában *NW, NE, SW, SE* elnevezéseket használunk (irányok angolul)

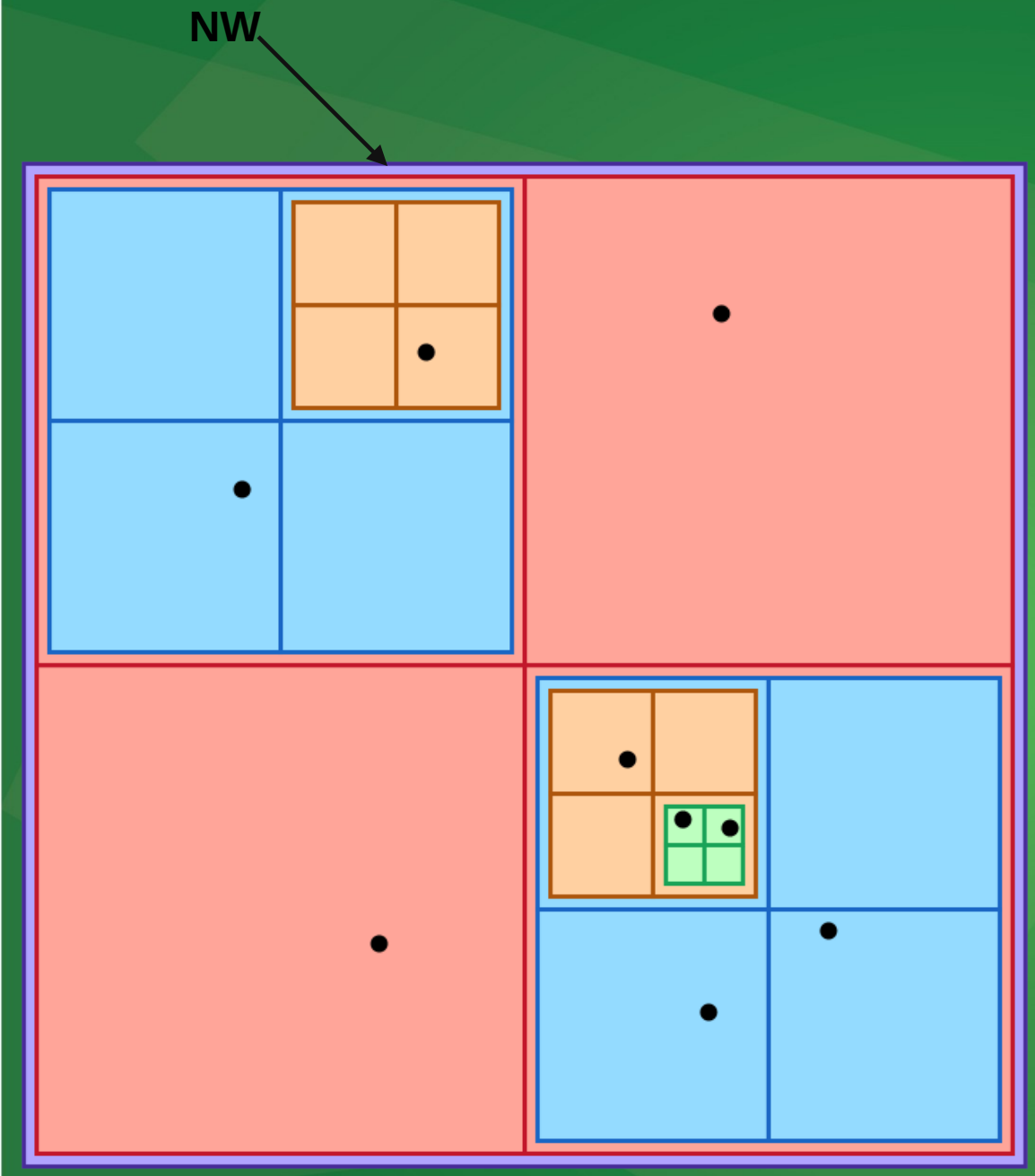


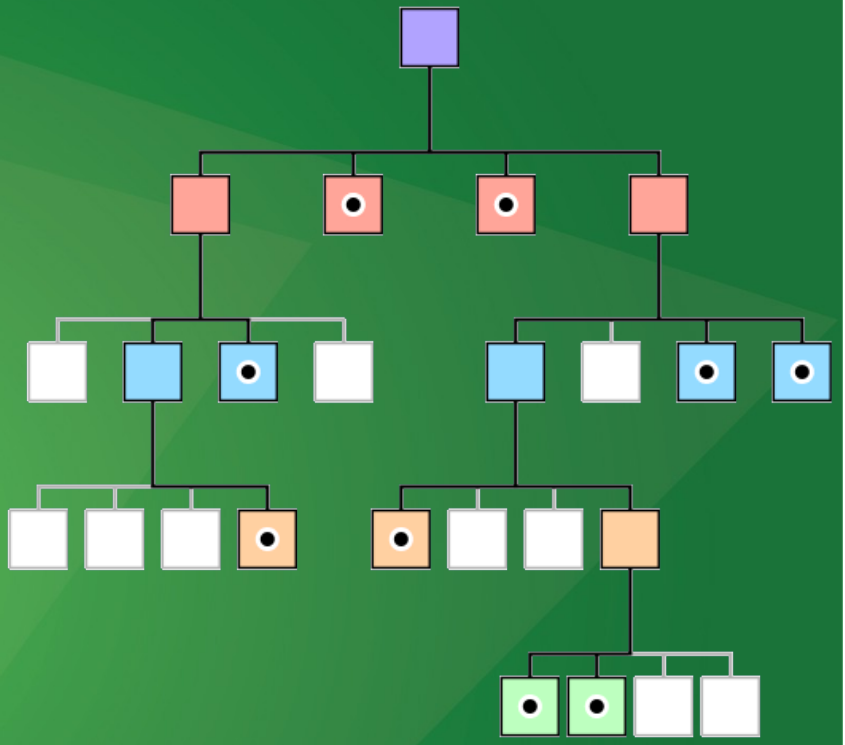




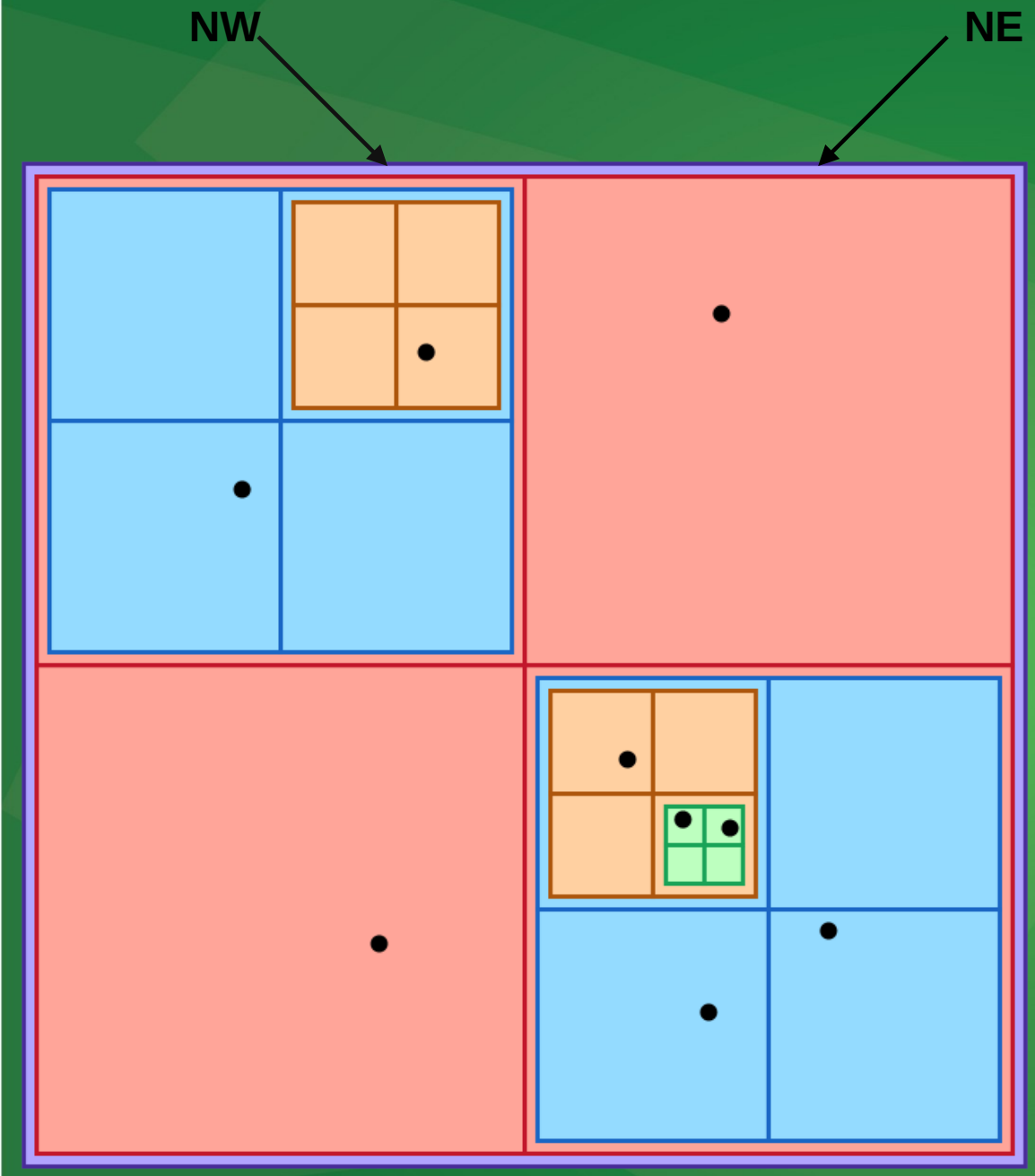


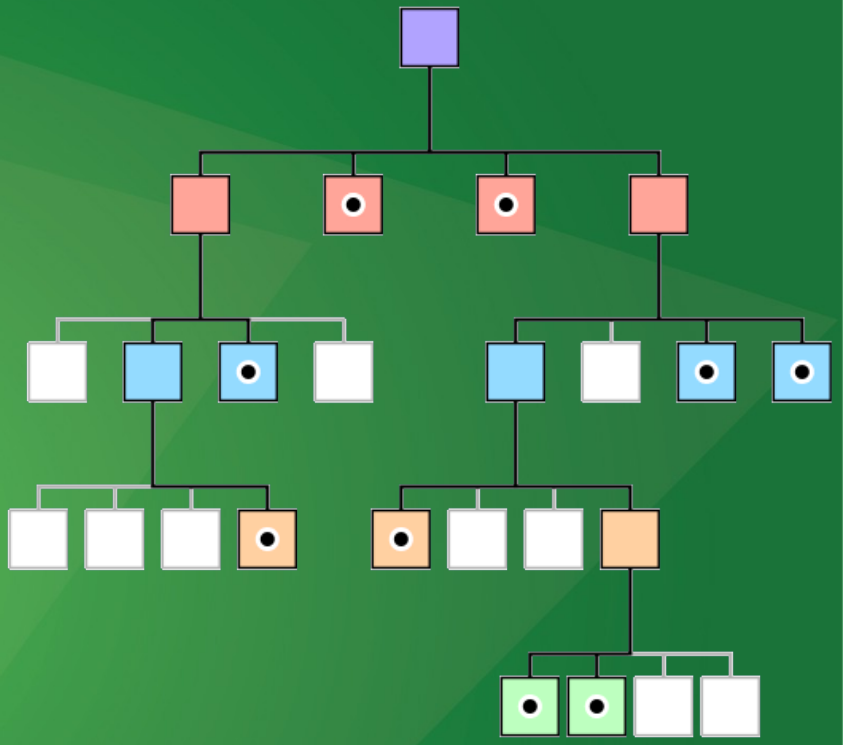
→ pontokat tartalmazó quadfa



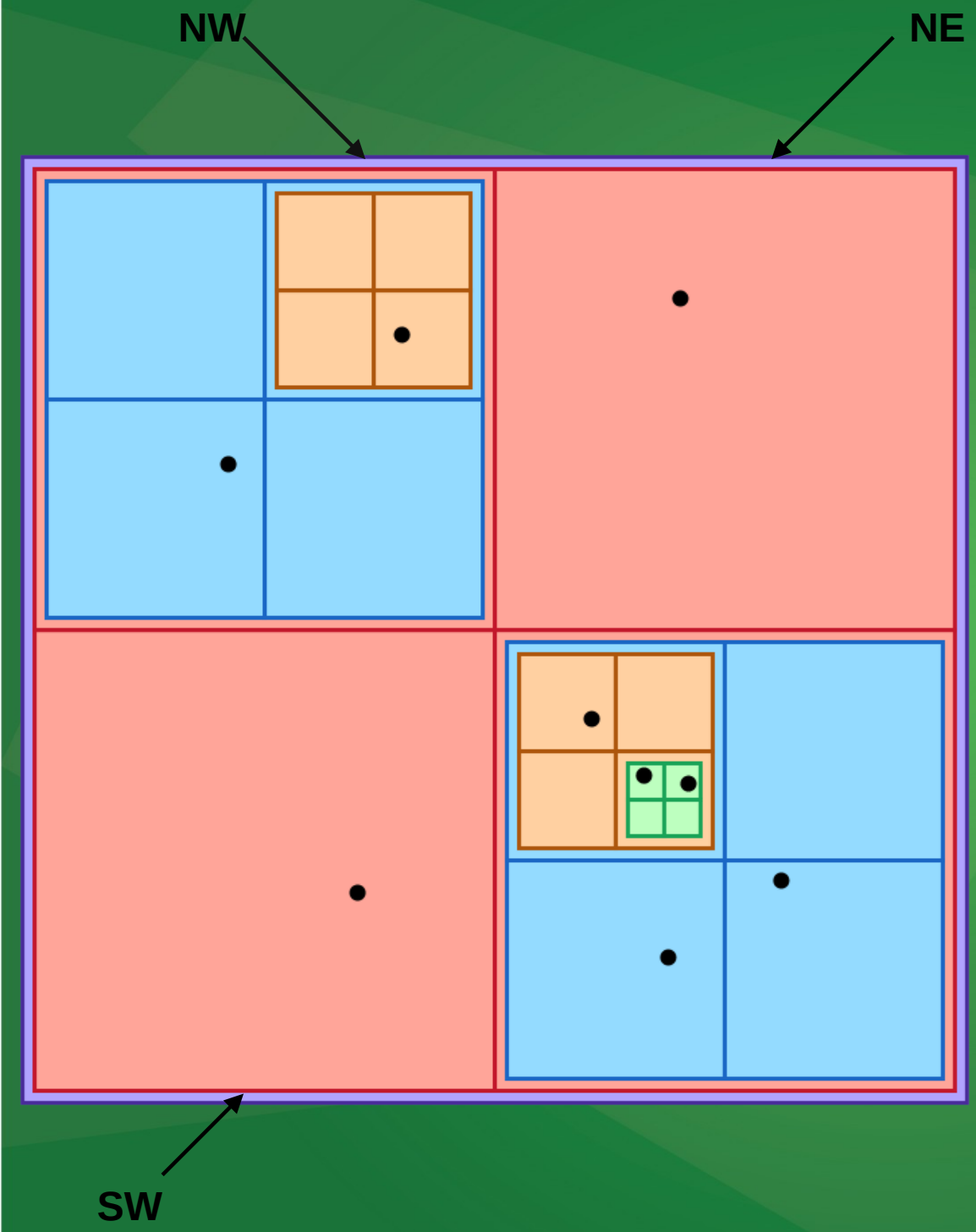


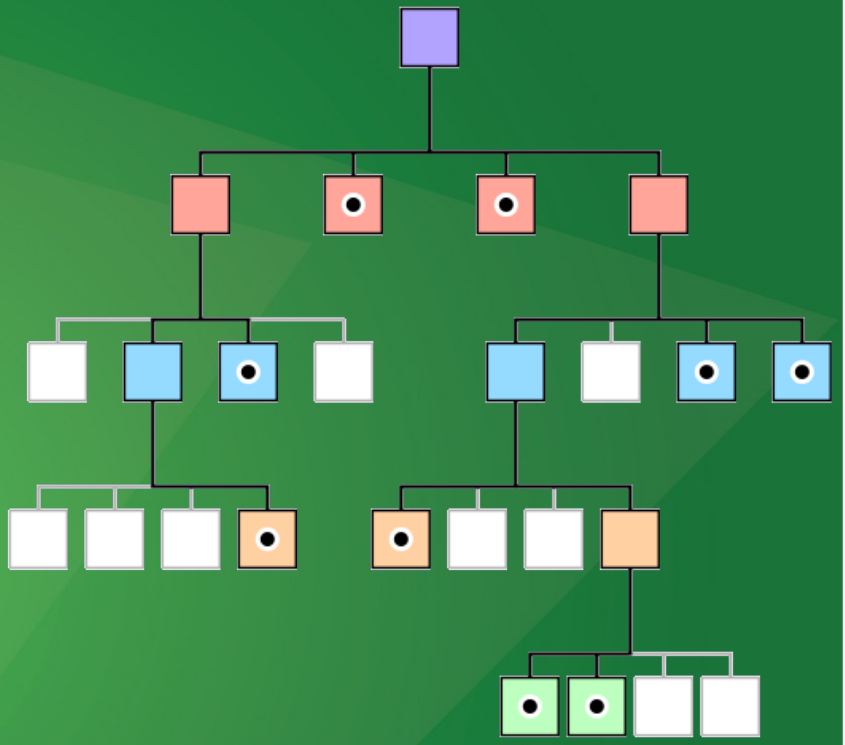
→ pontokat tartalmazó quadfa



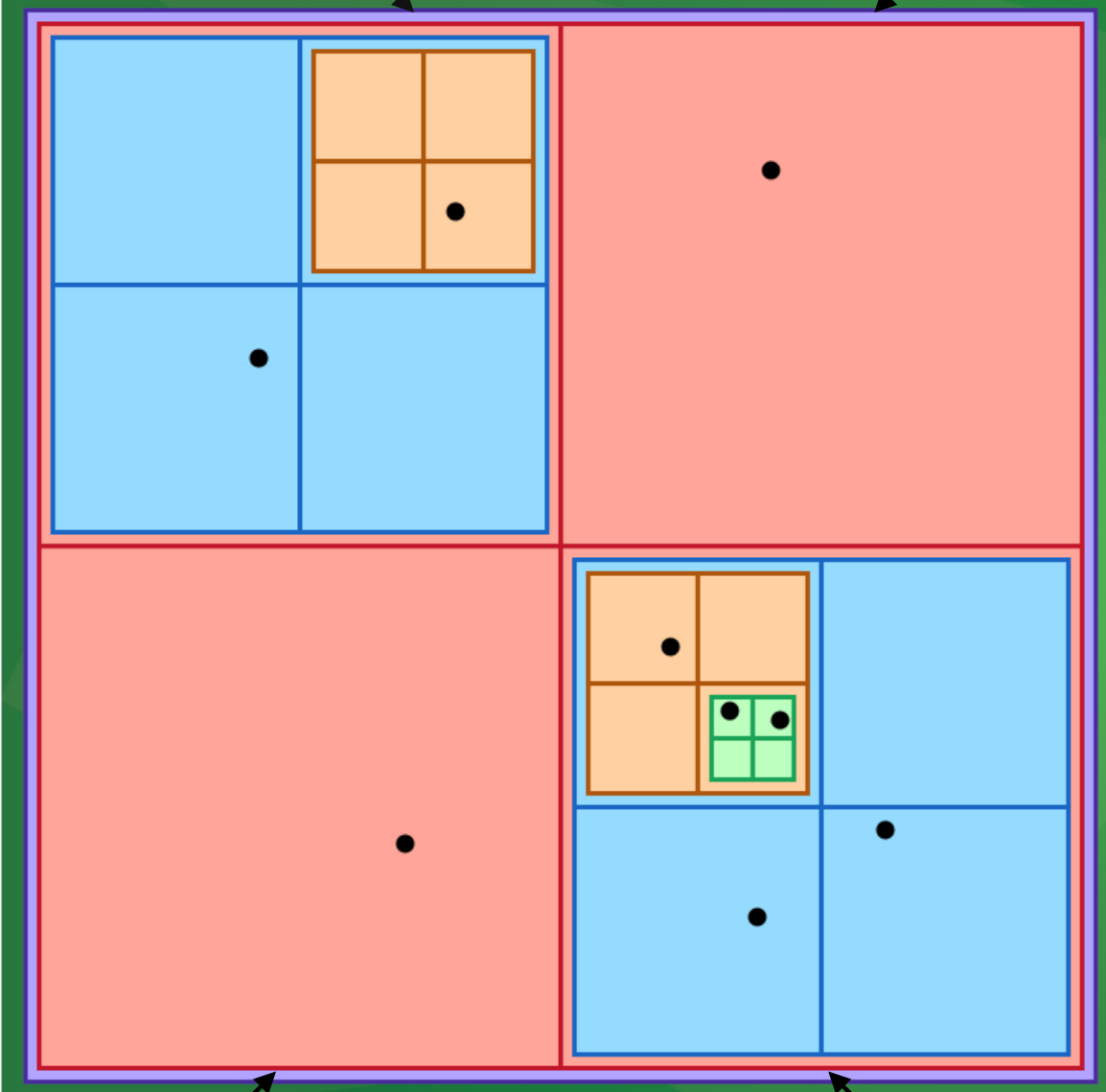


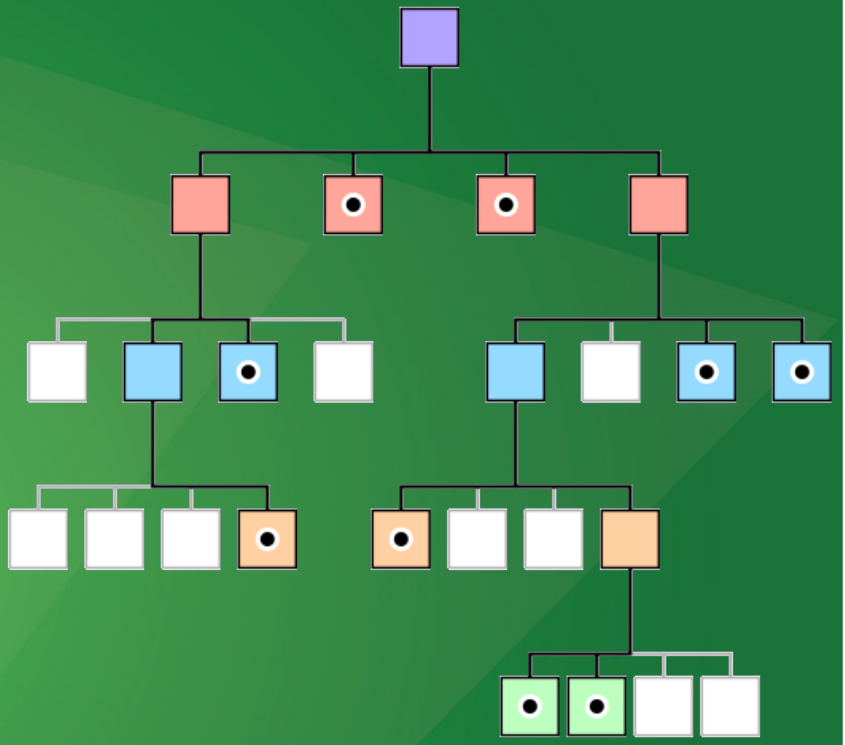
→ pontokat tartalmazó quadfa



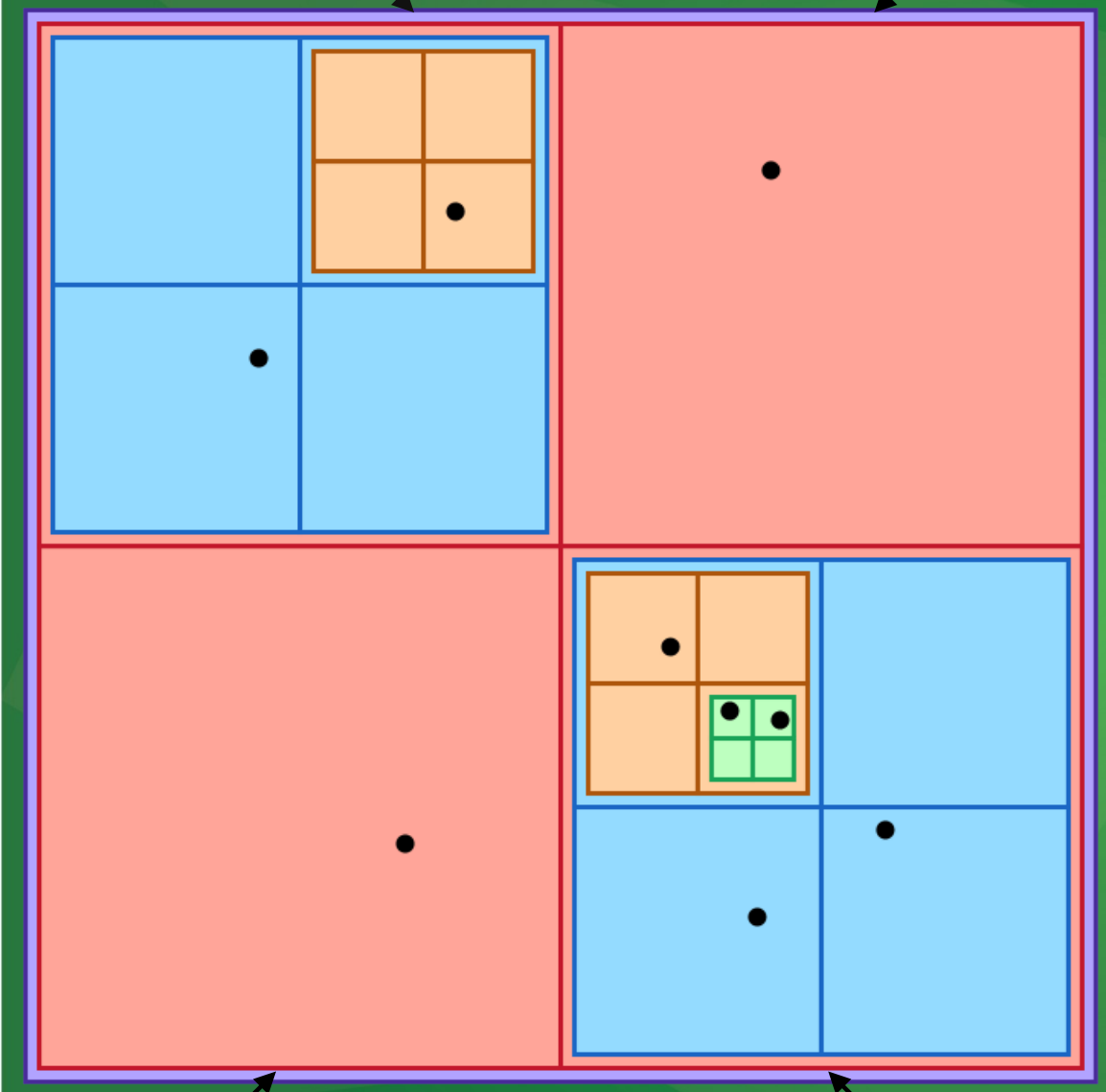


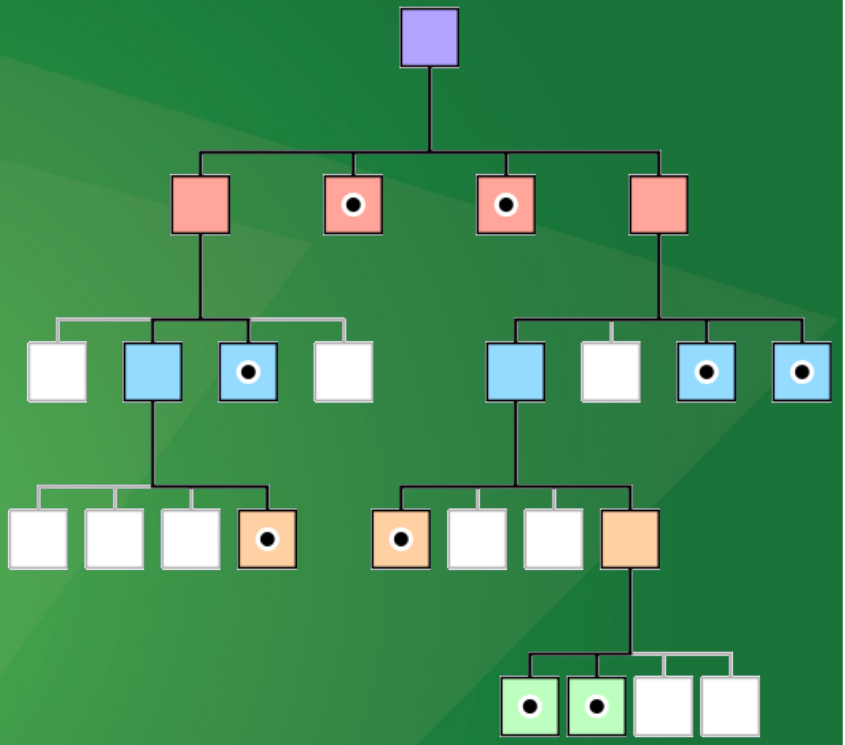
→ pontokat tartalmazó quadfa



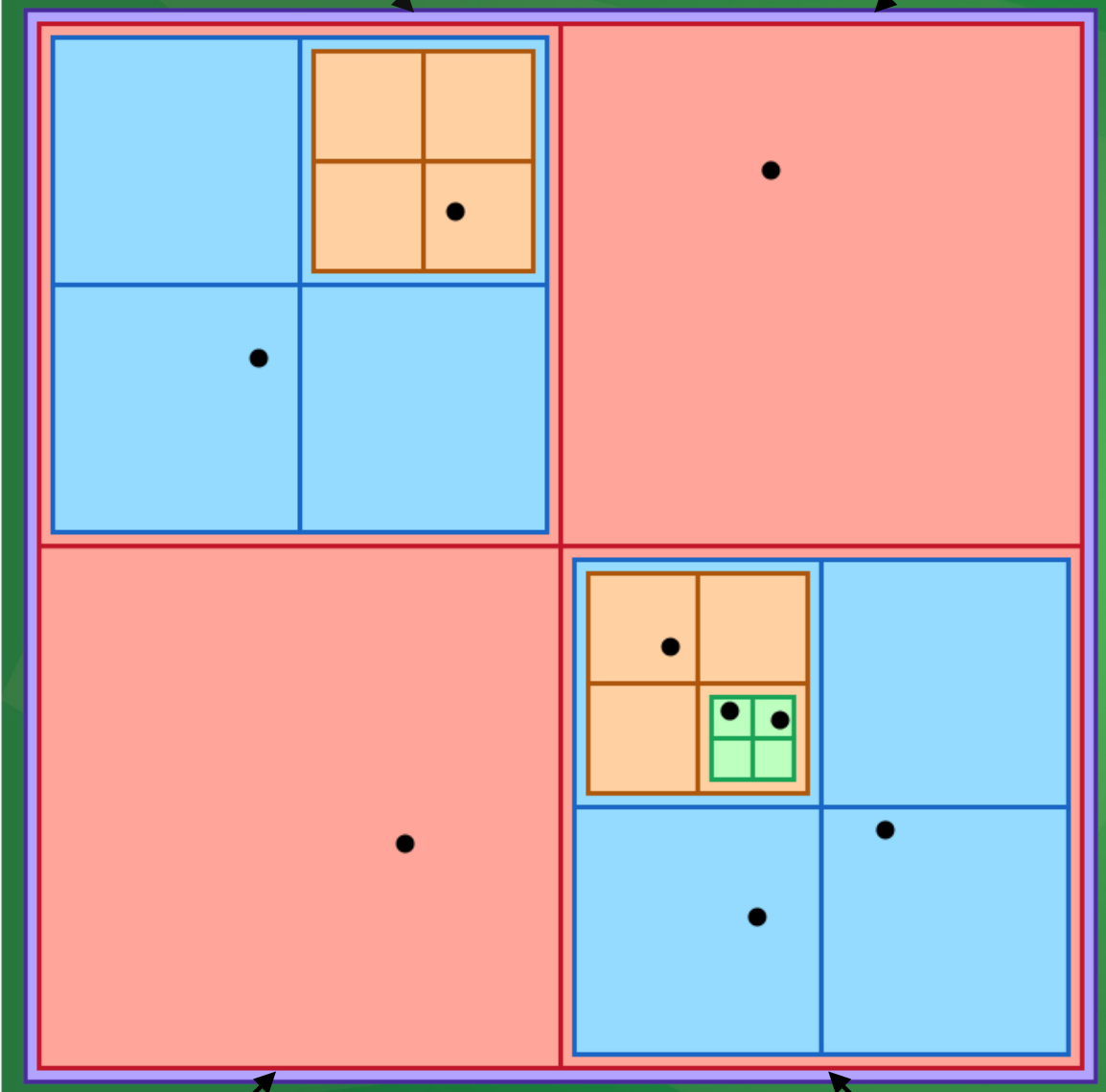


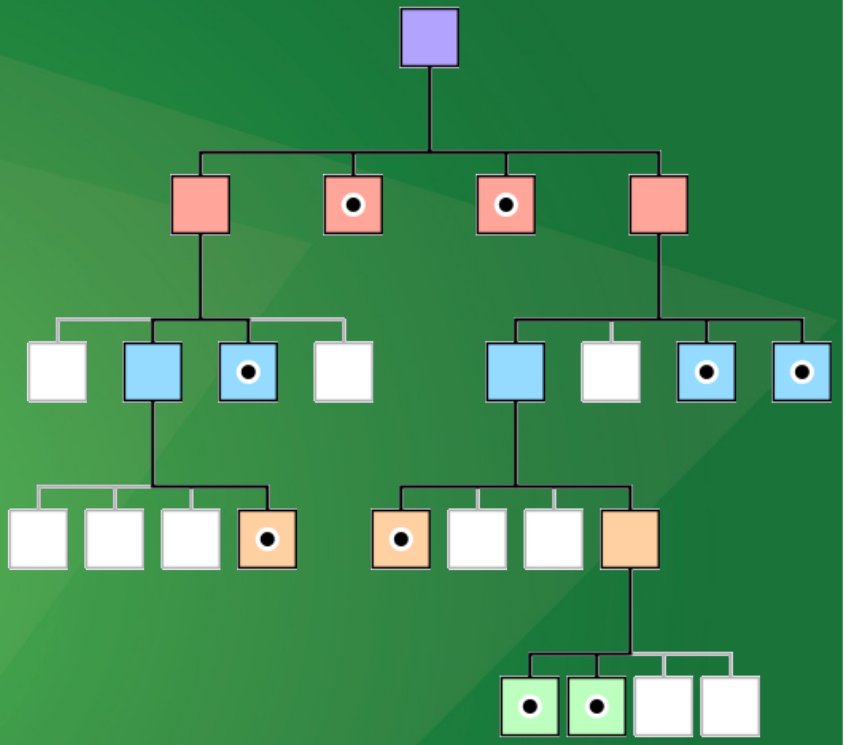
→ pontokat tartalmazó quadfa





→ pontokat tartalmazó quadfa
→ megfigyelhetjük, hogy a hasznos információ (pontok) mindig a levelekben helyezkednek el





→ pontokat tartalmazó quadfa
→ megfigyelhetjük, hogy a hasznos információ (pontok) mindig a levelekben helyezkednek el
→ függ a quadfa típusától és az implementációtól

Milyen típusú quadfák léteznek?

- ***Pont quadfa*** (point quadtree)

Milyen típusú quadfák léteznek?

- ***Pont quadfa*** (point quadtree)
  - méret nélküli, kétdimenziós koordinátával rendelkező objektumok (pontok) tárolása

Milyen típusú quadfák léteznek?

- ***Pont quadfa*** (point quadtree)
  - méret nélküli, kétdimenziós koordinátával rendelkező objektumok (pontok) tárolása
  - lásd előző ábra

Milyen típusú quadfák léteznek?

- ***Pont quadfa*** (point quadtree)
  - méret nélküli, kétdimenziós koordinátával rendelkező objektumok (pontok) tárolása
  - lásd előző ábra
  - általában van egy fix “ládaméret” (bucket size), aminek az elérése után a csomópont osztódik, és elosztja az utódai között a pontokat

Milyen típusú quadfák léteznek?

- ***Pont quadfa*** (point quadtree)
  - méret nélküli, kétdimenziós koordinátával rendelkező objektumok (pontok) tárolása
  - lásd előző ábra
  - általában van egy fix “ládaméret” (bucket size), aminek az elérése után a csomópont osztódik, és elosztja az utódai között a pontokat
    - → magyarázat az előző ábrán fellépő jelenségre (a ládaméret *1*)

Milyen típusú quadfák léteznek?

- ***Területi quadfa*** (region quadtree)

Milyen típusú quadfák léteznek?

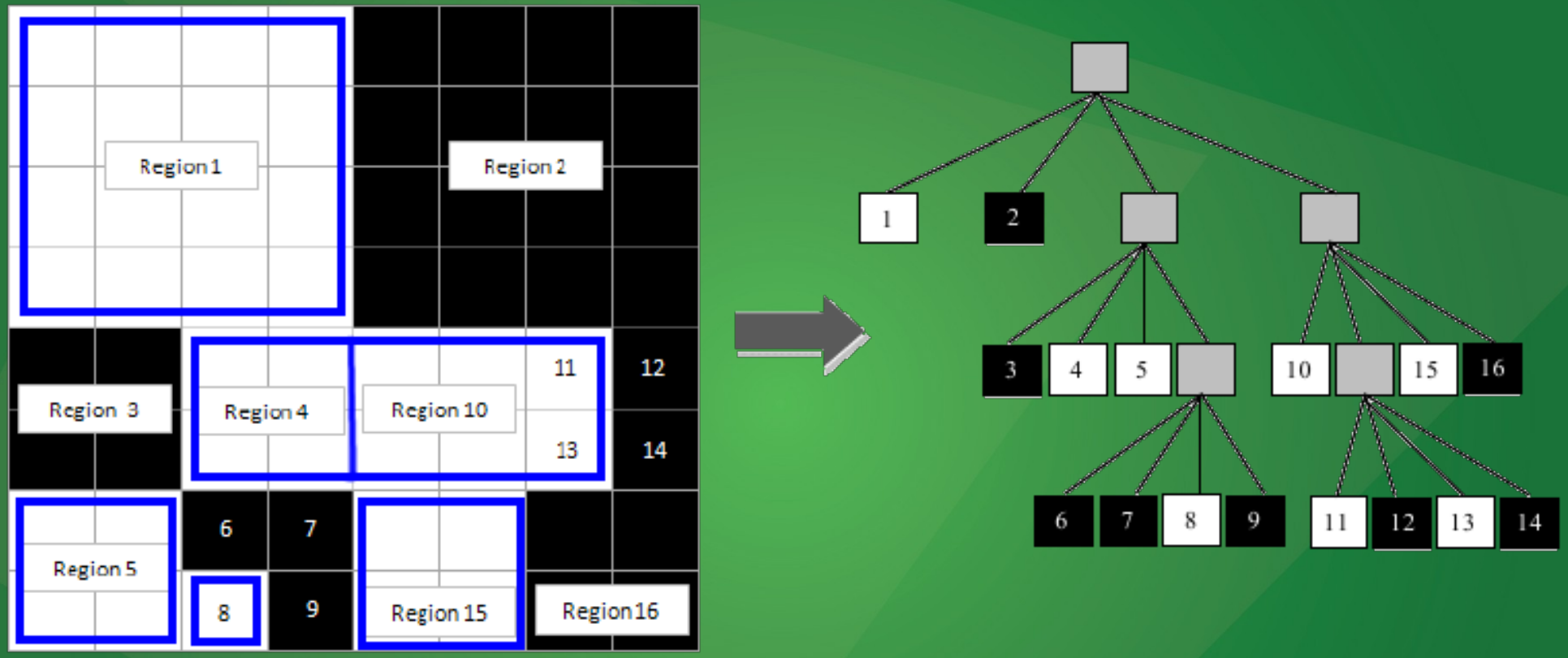
- ***Területi quadfa*** (region quadtree)
  - a quadronok nem tartalmaznak objektumokat, hanem magának a felosztásnak van jelentősége a tér egészére nézve

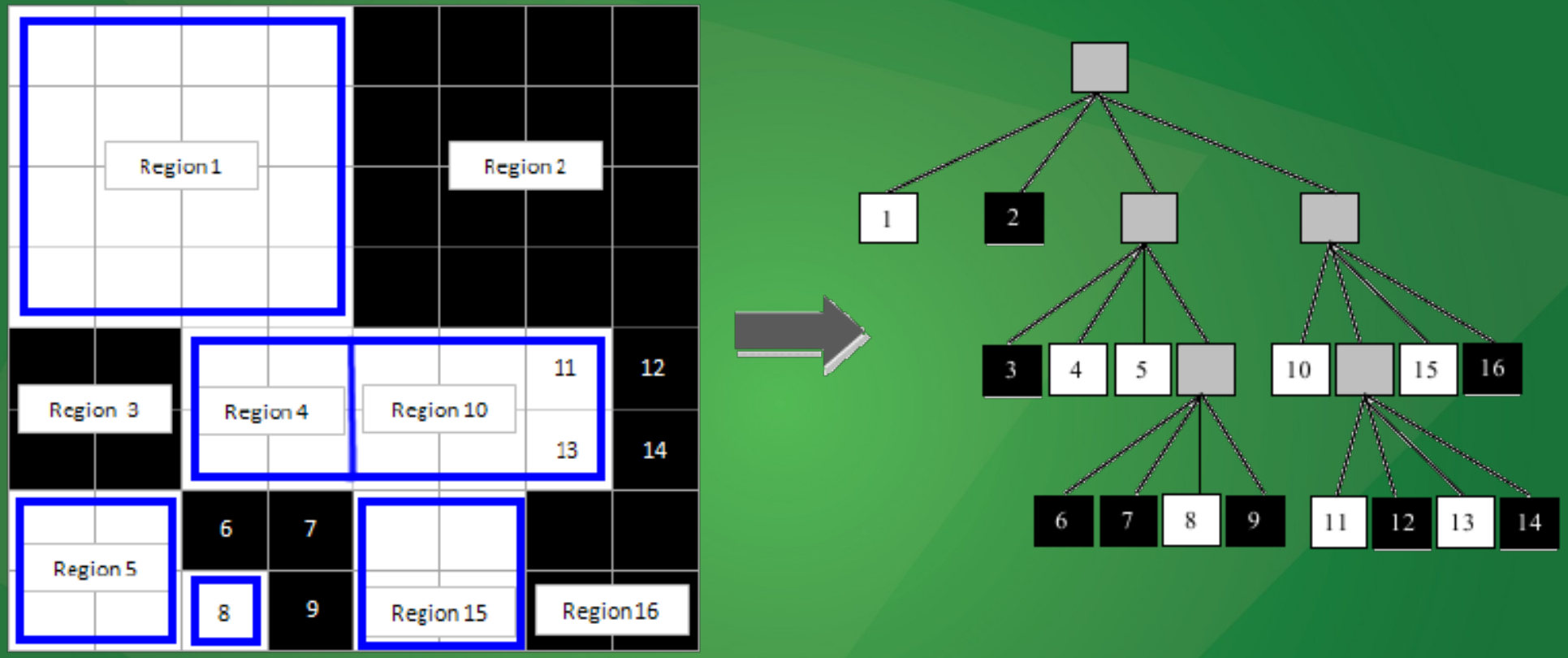
Milyen típusú quadfák léteznek?

- ***Területi quadfa*** (region quadtree)
  - a quadronok nem tartalmaznak objektumokat, hanem magának a felosztásnak van jelentősége a tér egészére nézve
  - előnyös 2-hatvány nagyságú területméretet választani

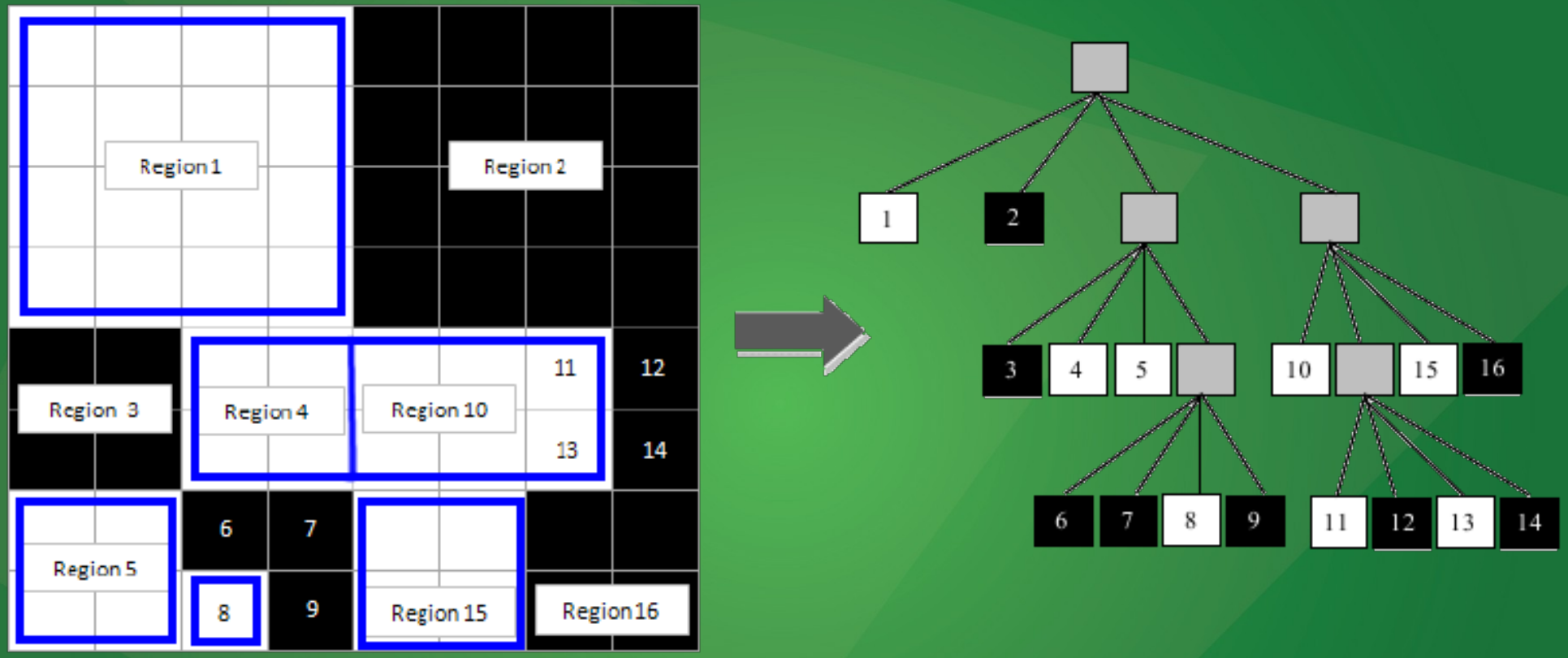
Milyen típusú quadfák léteznek?

- ***Területi quadfa*** (region quadtree)
  - a quadronok nem tartalmaznak objektumokat, hanem magának a felosztásnak van jelentősége a tér egészére nézve
  - előnyös 2-hatvány nagyságú területméretet választani
  - pl. képtömörítésben használják

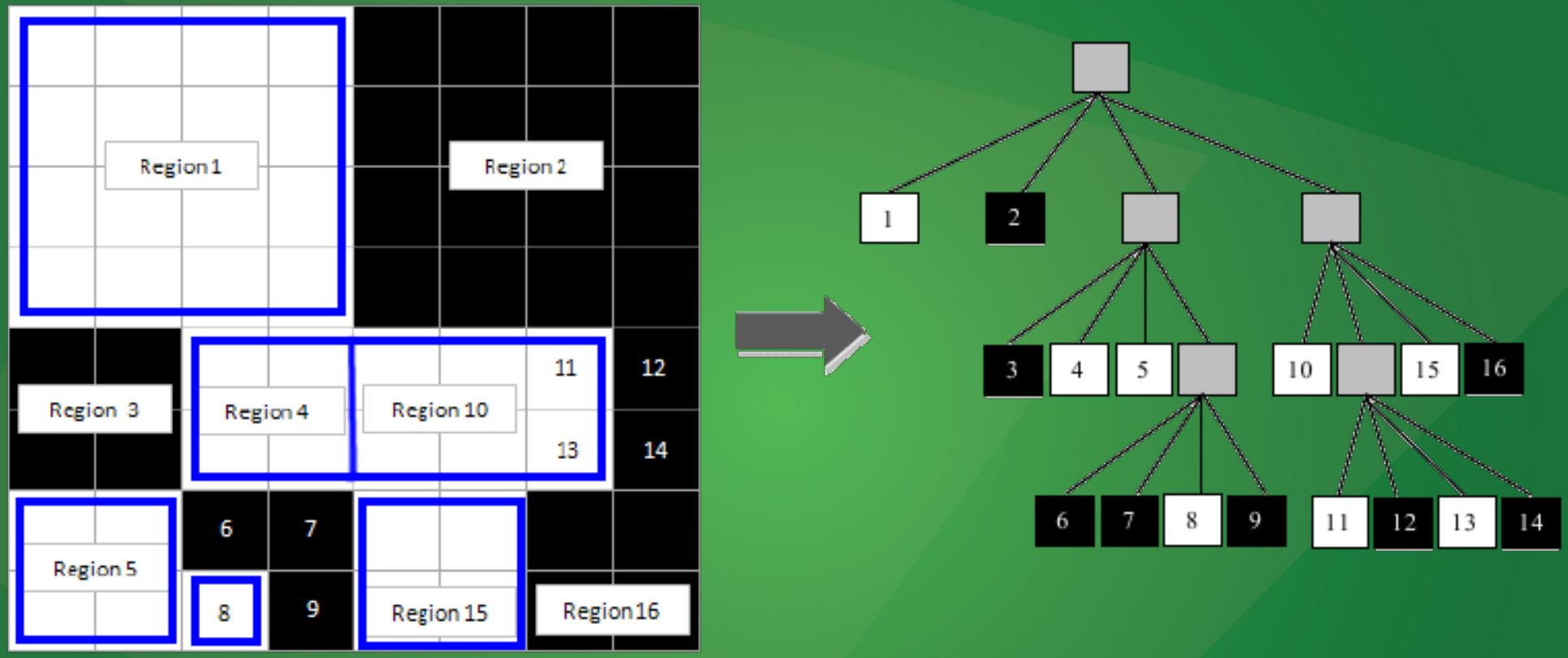

Region 1
Region 2
Region 3
Region 4
Region 10
11
12
13
14
Region 5
6
7
8
9
Region 15
Region 16
1
2
3
4
5
10
15
16
6
7
8
9
11
12
13
14



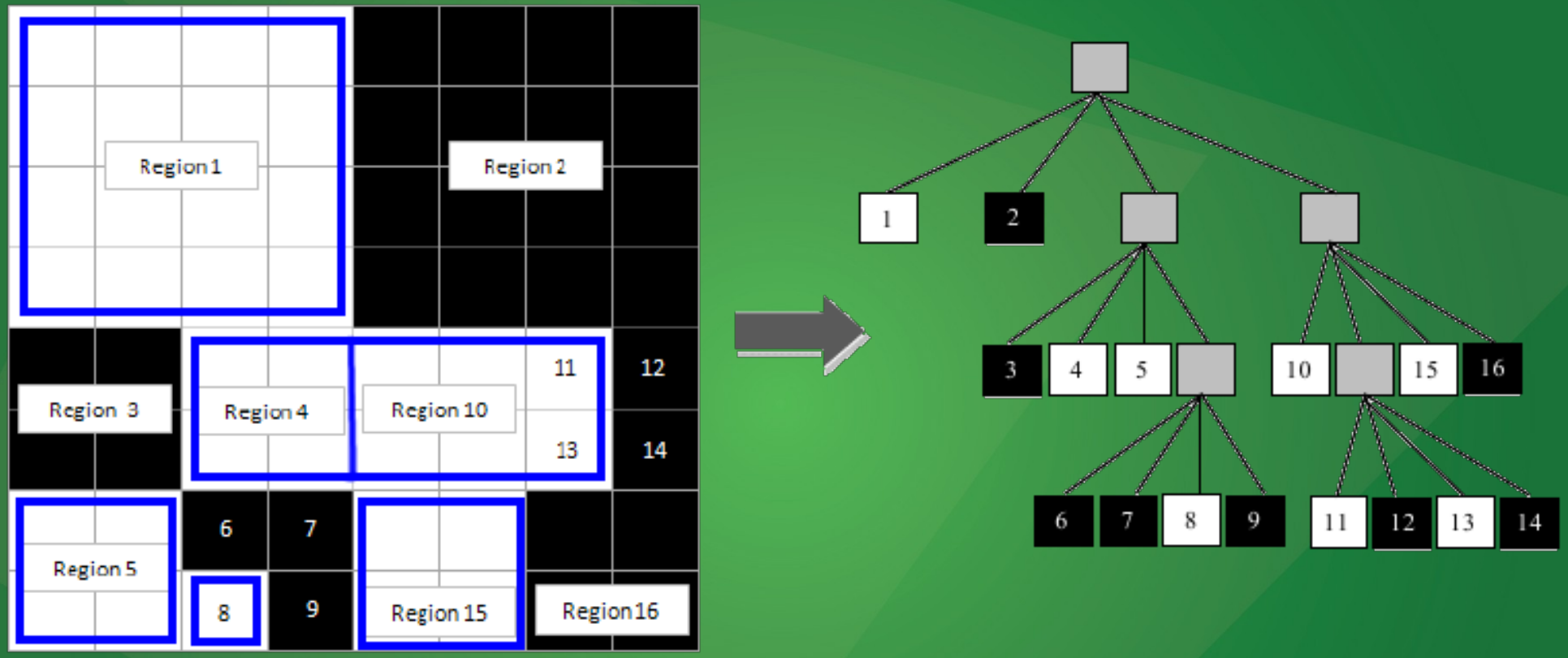
→ addig osztottuk a teret, amíg volt benne fehér és fekete pixel is



→ addig osztottuk a teret, amíg volt benne fehér és fekete pixel is
→ a levelek a régiók, vagyis a tiszta fehér vagy tiszta fekete részek



→ addig osztottuk a teret, amíg volt benne fehér és fekete pixel is
→ a levelek a régiók, vagyis a tiszta fehér vagy tiszta fekete részek
→ mikor hátrányos ez a modell?



→ addig osztottuk a teret, amíg volt benne fehér és fekete pixel is
→ a levelek a régiók, vagyis a tiszta fehér vagy tiszta fekete részek
→ mikor hátrányos ez a modell? → pl. “sakktábla” szerű kép

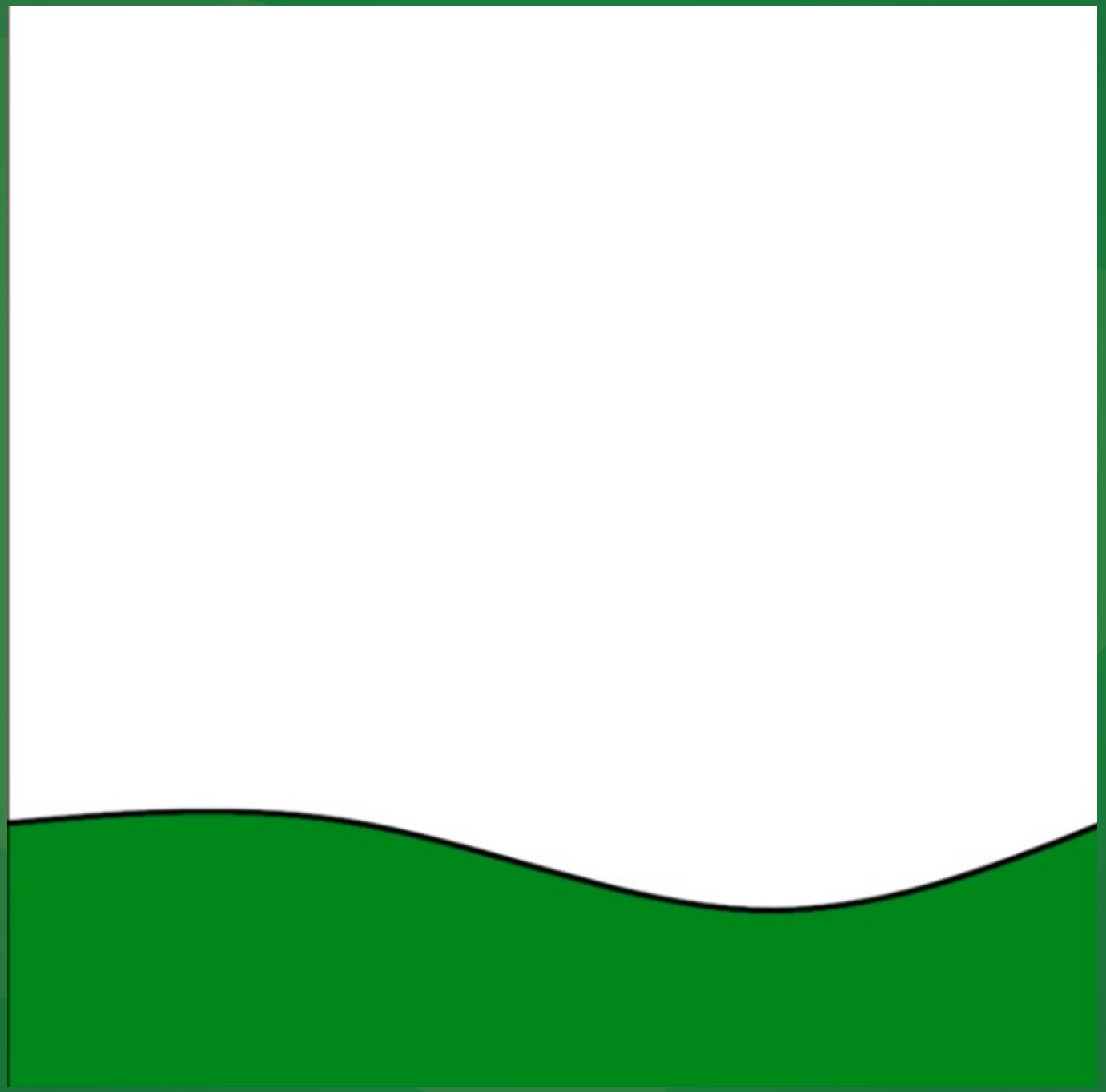
→ egy színes kép tömörítése

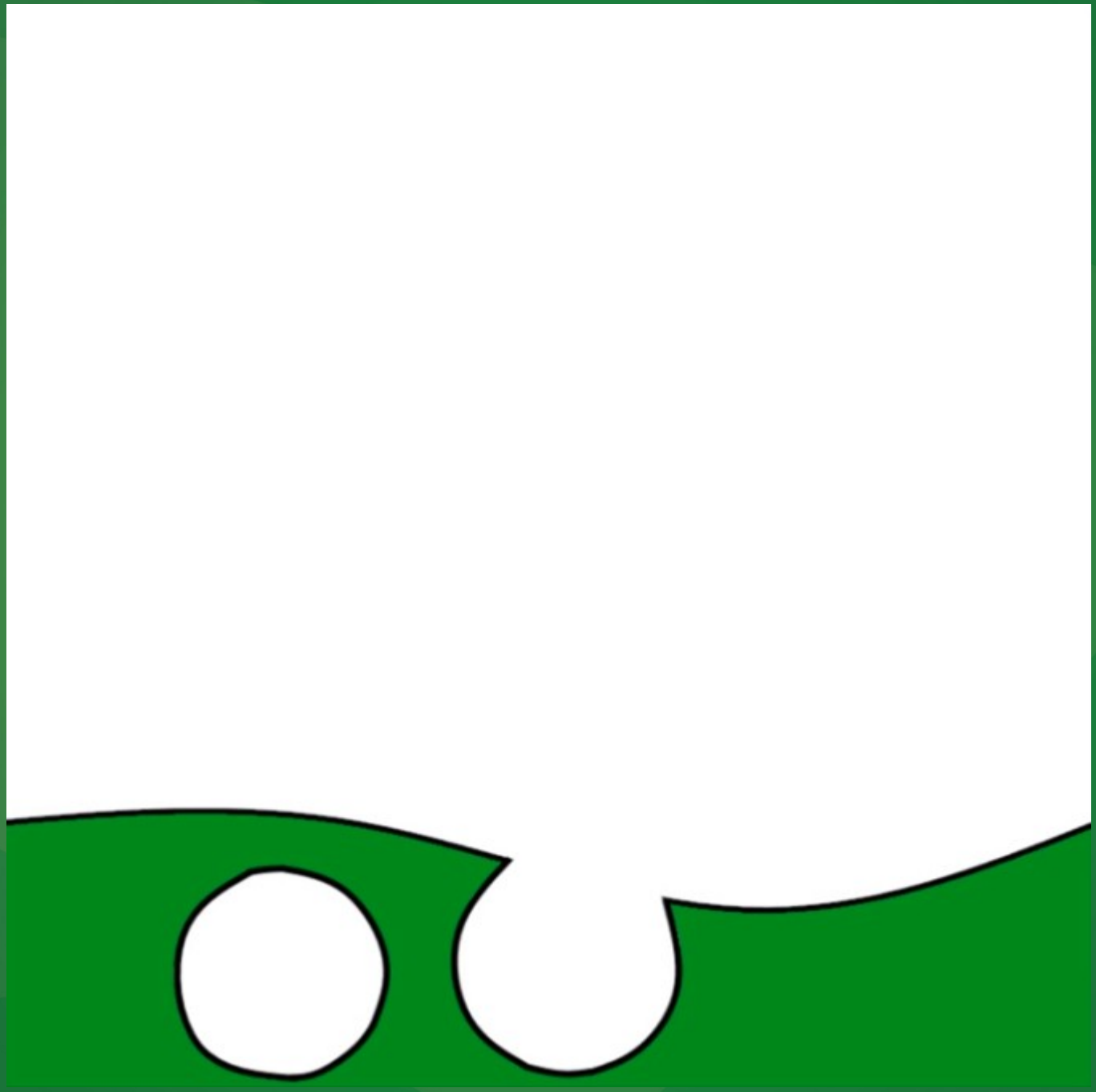
→ egy színes kép tömörítése
→ a felosztás leáll, amint a csomópont azonos, vagy csak “nagyon hasonló” színeket fed le

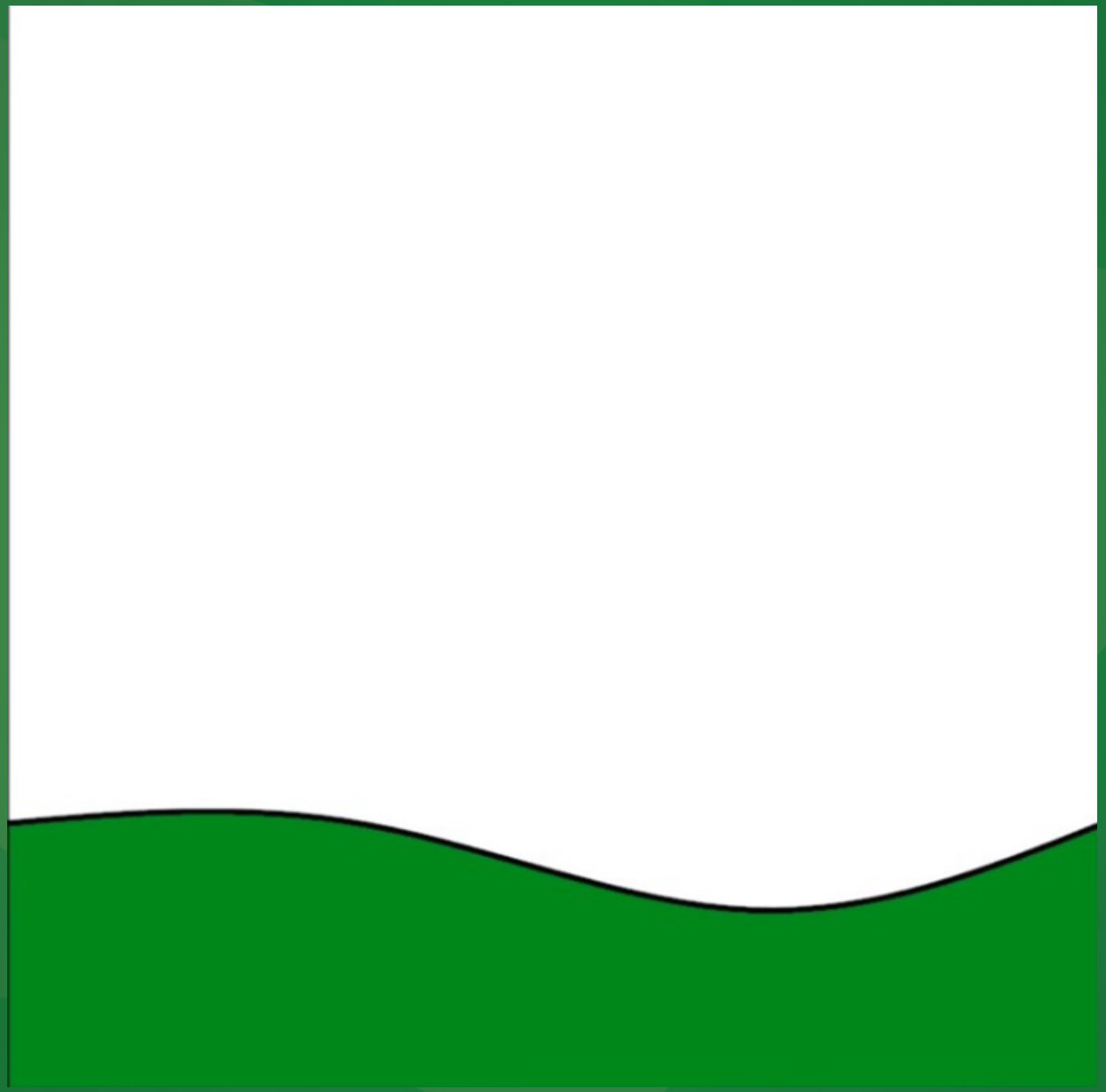
- Ezt a modellt játékokban is előszeretettel használják

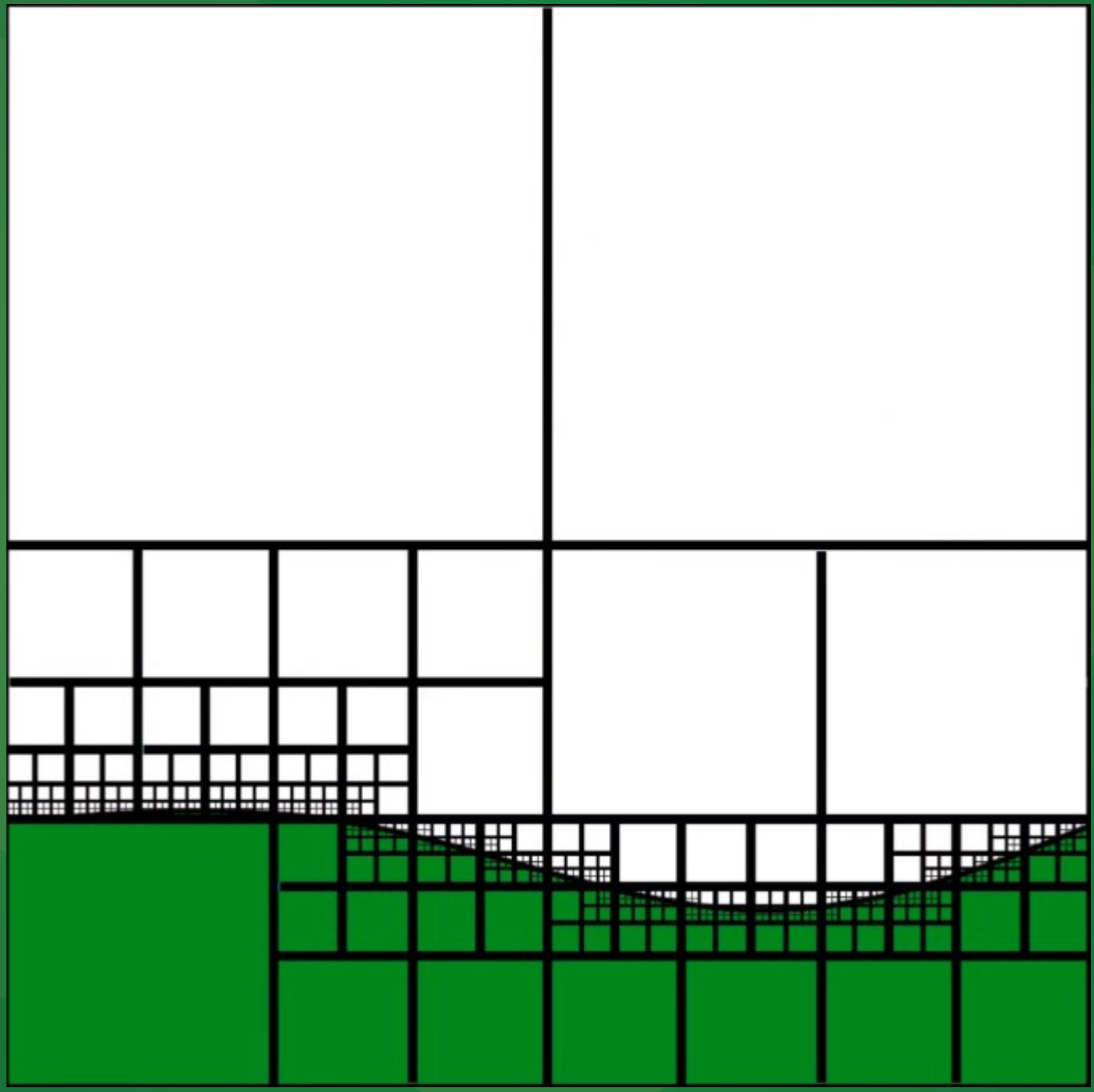
- Ezt a modellt játékokban is előszeretettel használják
  - → pl. *Worms* játéksorozat

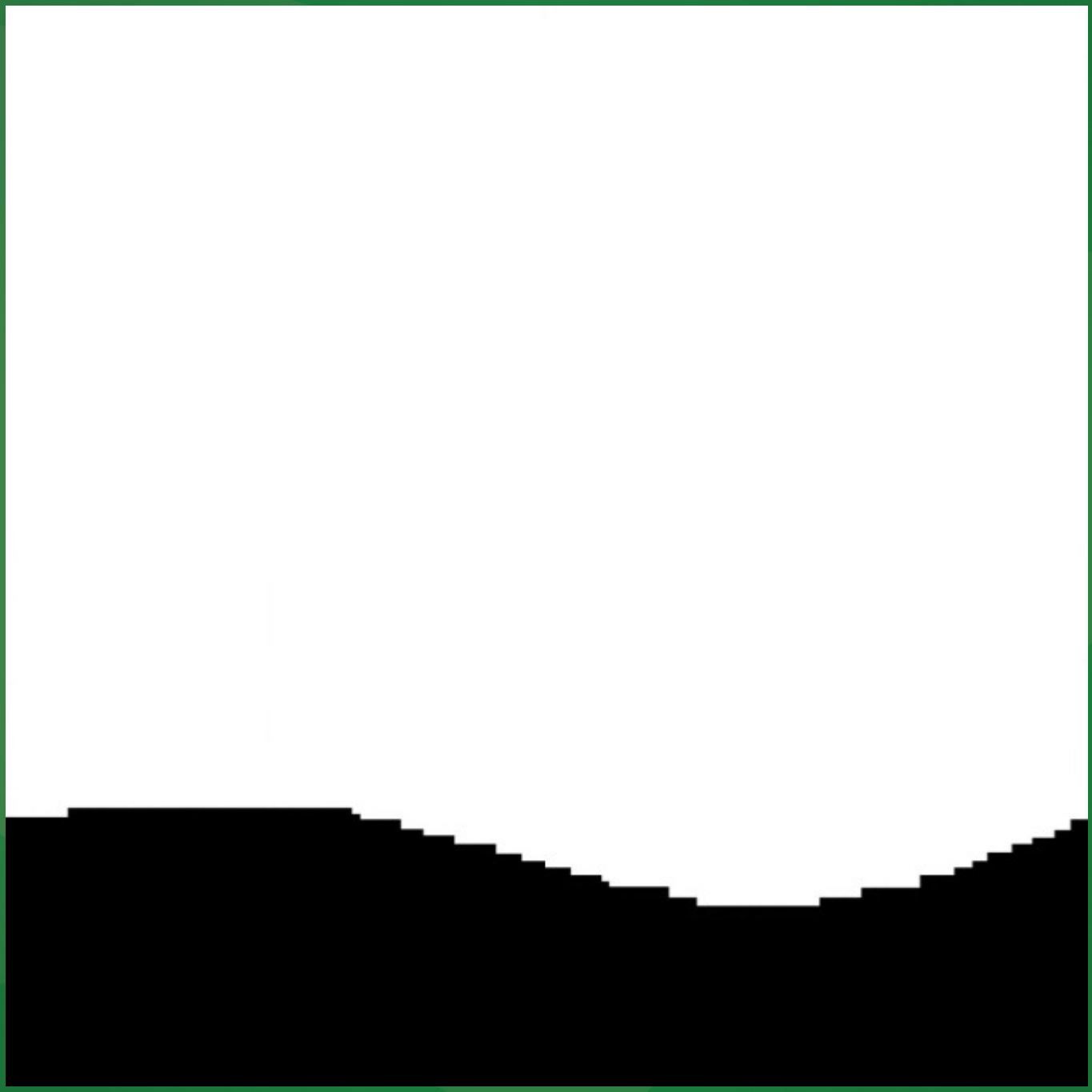
- Ezt a modellt játékokban is előszeretettel használják
  - → pl. *Worms* játéksorozat

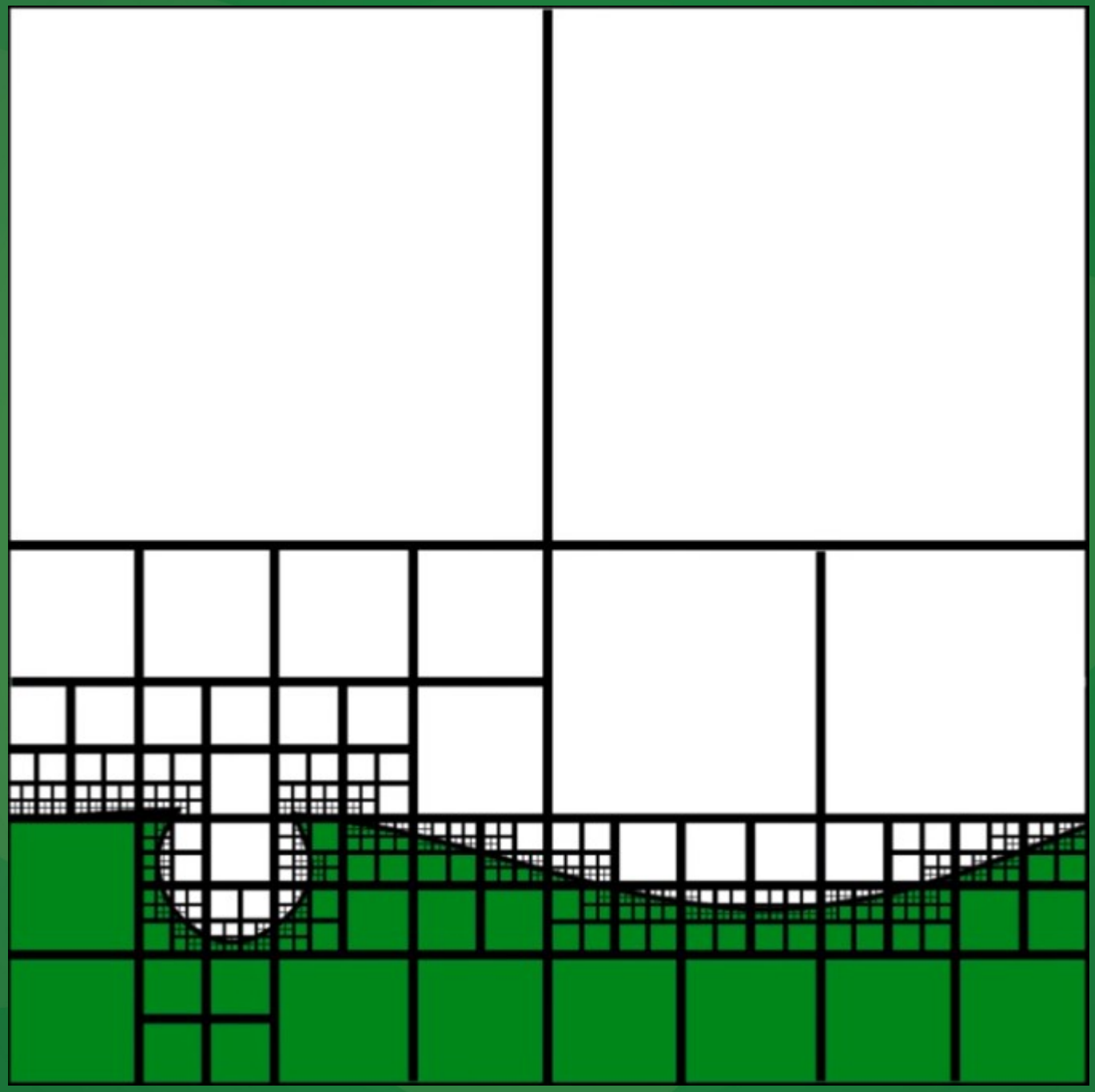












Milyen típusú quadfák léteznek?

- ***Perem quadfa*** (boundary quadtree)

Milyen típusú quadfák léteznek?

- ***Perem quadfa*** (boundary quadtree)
  - általában vonalakat tárolnak

Milyen típusú quadfák léteznek?

- ***Perem quadfa*** (boundary quadtree)
  - általában vonalakat tárolnak
  - görbék megközelítésére használják

Milyen típusú quadfák léteznek?

- ***Perem quadfa*** (boundary quadtree)
  - általában vonalakat tárolnak
  - görbék megközelítésére használják
    - → addig osztják a teret, míg a felbontás nagyon kicsi lesz, így vonalak sokaságaként lehet kezelni a görbét

Milyen típusú quadfák léteznek?

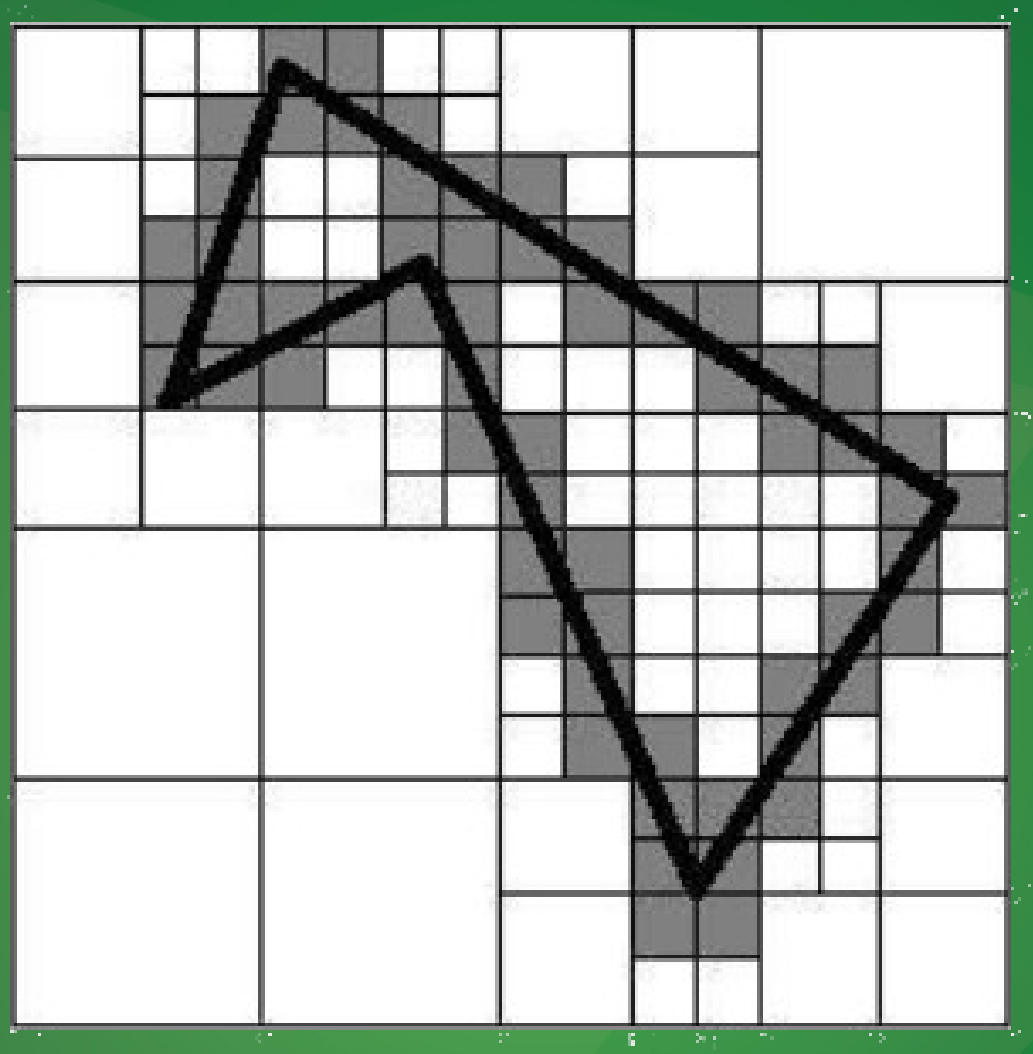
- ***Perem quadfa*** (boundary quadtree)
  - általában vonalakat tárolnak
  - görbék megközelítésére használják
    - → addig osztják a teret, míg a felbontás nagyon kicsi lesz, így vonalak sokaságaként lehet kezelni a görbét
  - hátrányai:

Milyen típusú quadfák léteznek?

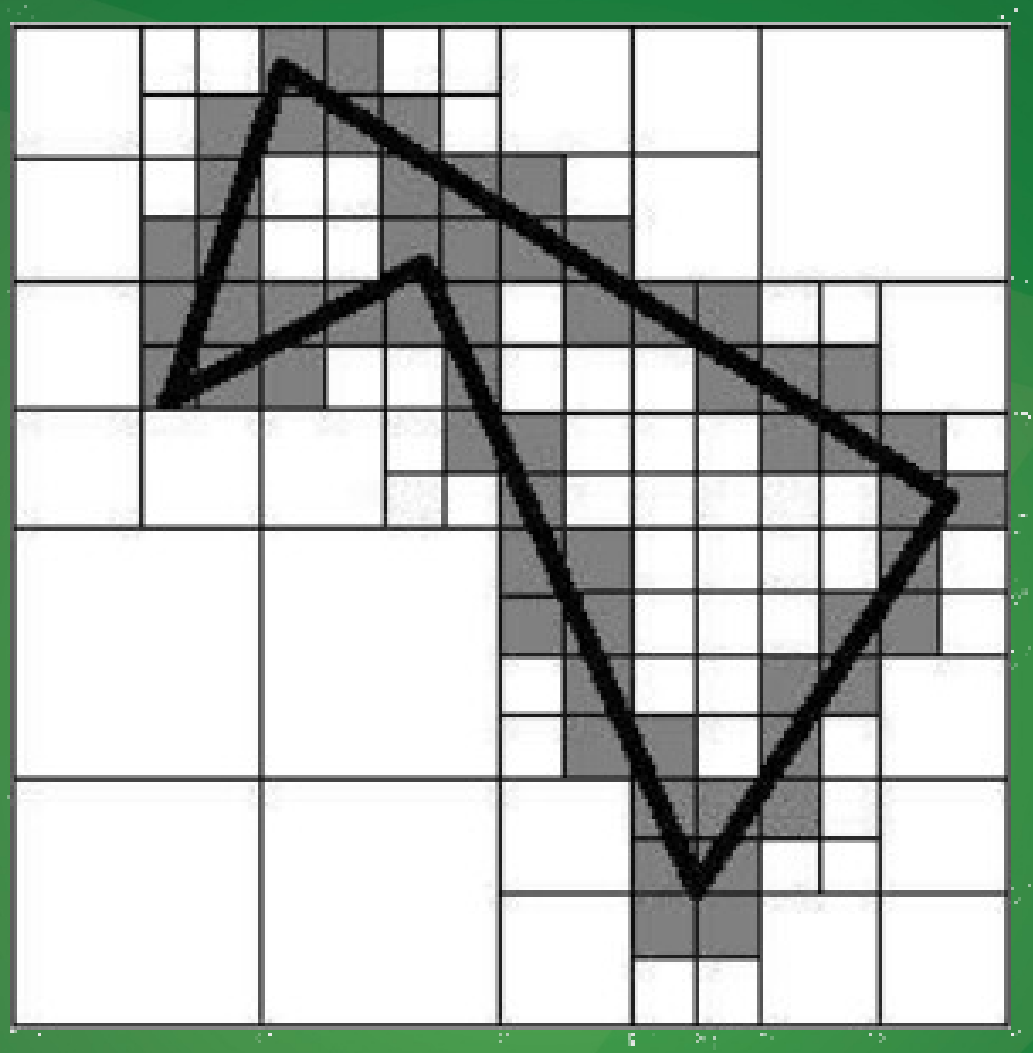
- ***Perem quadfa*** (boundary quadtree)
  - általában vonalakat tárolnak
  - görbék megközelítésére használják
    - → addig osztják a teret, míg a felbontás nagyon kicsi lesz, így vonalak sokaságaként lehet kezelni a görbét
  - hátrányai:
    - túl nagy lesz a fa mélysége

Milyen típusú quadfák léteznek?

- ***Perem quadfa*** (boundary quadtree)
  - általában vonalakat tárolnak
  - görbék megközelítésére használják
    - → addig osztják a teret, míg a felbontás nagyon kicsi lesz, így vonalak sokaságaként lehet kezelni a görbét
  - hátrányai:
    - túl nagy lesz a fa mélysége
    - legtöbb esetben egyáltalán nem lesz kiegyensúlyozott



→ ha csak a szürke négyzeteket nézzük, akkor is “össze tud állni” egy viszonylag kivehető kép az alakzatról



→ ha csak a szürke négyzeteket nézzük, akkor is “össze tud állni” egy viszonylag kivehető kép az alakzatról

→ minél nagyobb a felbontás, annál pontosabb képet kapunk

Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák***

Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák*** (rectangle quadree?)

Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák*** (rectangle quadree?)
  - használatuk:

Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák*** (rectangle quadree?)
  - használatuk:
    - egy-a-sokhoz ütközések lekérdezése

Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák*** (rectangle quadree?)
  - használatuk:
    - egy-a-sokhoz ütközések lekérdezése
    - → Pl. adott N téglalap a térben, valamint K másik téglalap, amit szintén el szeretnénk helyezni fix pozíciókra. Adjuk meg, hogy melyeket tudjuk elhelyezni!

Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák*** (rectangle quadree?)
  - használatuk:
    - egy-a-sokhoz ütközések lekérdezése
    - → Pl. adott N téglalap a térben, valamint K másik téglalap, amit szintén el szeretnénk helyezni fix pozíciókra. Adjuk meg, hogy melyeket tudjuk elhelyezni!
      - → **naív** megközelítés: **N** * **K** ütközés vizsgálata

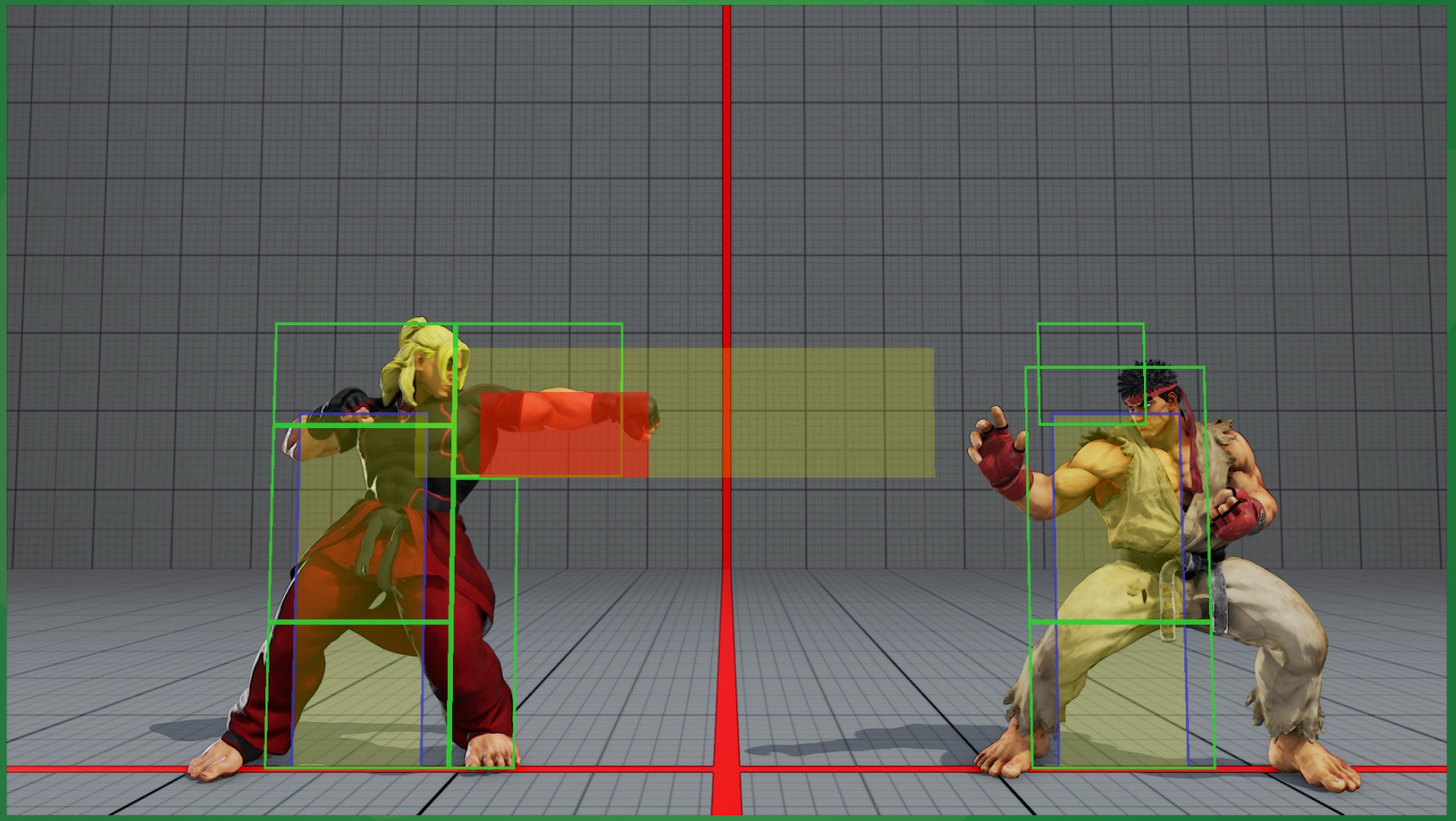
Milyen típusú quadfák léteznek?

- ***Téglalapok tárolására használt quadfák*** (rectangle quadree?)
  - használatuk:
    - egy-a-sokhoz ütközések lekérdezése
    - → Pl. adott N téglalap a térben, valamint K másik téglalap, amit szintén el szeretnénk helyezni fix pozíciókra. Adjuk meg, hogy melyeket tudjuk elhelyezni!
      - → **naív** megközelítés: **N** * **K** ütközés vizsgálata
      - → **quadfával**: sokkal **gyors**abban ki tudjuk zárni azokat, amikkel a kérdéses téglalapok biztosan nem ütköznek

- Szintén videójátékok esetén

- Szintén videójátékok esetén → *hitboxok!*

- Szintén videójátékok esetén → *hitboxok!*



Milyen AAT-kból épül fel a téglalap-quadfa modellje?

- ***XY típus***
  - egy koordinátát jelent
  - **x** komponens és **y** komponens

Milyen AAT-kból épül fel a téglalap-quadfa modellje?

- ***AABB típus***
  - egy téglalapot jelent
  - megadjuk a bal felső és a jobb alsó koordinátáit: **TL** és **BR** (XY típusúak)
  - lekérdező műveletek:
    - S.**ütközik**(T) → megadja, hogy az S téglalap ütközik-e a T téglalappal
      - → kommutatív művelet
    - S.**tartalmaz**(T) → megadja, hogy az S téglalap tartalmazza-e a T téglalapot
      - → *nem* kommutatív
    - → koordináták összehasonlításán alapulnak (nem részletezzük)

Milyen AAT-kból épül fel a téglalap-quadfa modellje?

- ***QuadNode típus***
  - a quadfa egy csomópontját jelenti
  - tartalmazza a lefedett területet, mint téglalap → **b** (AABB típus)
  - tartalmazza a szintjét: **depth** (egész típus)
  - tartalmaz 4 QuadNode típusú mutatót a gyerekeire: **NW, NE, SW, SE**
  - tartalmaz egy tárolót, amiben a saját területén fekvő téglalapokat tárolja: **objs[]**
  - legyen **MAXDEPTH** egy globális változó, ami megadja, hogy mekkora mélységig mehetünk a fa létrehozását illetően

Milyen AAT-kból épül fel a téglalap-quadfa modellje?

- Műveletek:
  - **létrehoz**(b, depth) → létrehoz egy üres quadfát, ami a **b** téglalapot fedi le és a **depth** szinttől kezdődik
  - Q.**beszúr**(obj) → beszúrja az **obj** téglalapot a **Q** quadfába
  - Q.**töröl**(b) → töröl minden olyan téglalapot a **Q** quadfából, ami érinti a **b** téglalapot
  - Q.**lekérdez**(b) → megad minden olyan téglalapot a **Q** quadfából, ami ütközik a **b** téglalappal

A létrehoz művelet

létrehoz(*b*, *depth*):
    legyen *új* egy QuadNode
    új.b = *b*
    új.depth = *depth*
    új.NW = *új*.NE = *új*.SW = *új*.SE = *NULL*
    új.objs = []
    visszatérít *új*

A **beszúr** *művelet*

```
Q.beszúr(obj):
    minden qnext-re Q.{NW, NE, SW, SE}-ből:
        legyen bnext a qnext területének megfelelő téglalap
        ha bnext.tartalmaz(obj):
            ha qnext gyerek létezik:
                qnext.beszúr(obj)
                vége
            különben ha Q.depth < MAXDEPTH – 1
                qnext = létrehoz(bnext, Q.depth + 1)
                qnext.beszúr(obj)
                vége
            vége ha
        vége ha
    vége minden
    Q.objs.hozzáad(obj)
vége
```

A **töröl** *művelet*

```
Q.töröl(b):
    minden obj-ra Q.objs[]-ből:
        ha b.ütközik(obj):
            eltávolít obj
        vége ha
    vége minden
    minden qnext-re Q.{NW, NE, SW, SE}-ből:
        legyen bnext a qnext területének megfelelő téglalap
        ha b.ütközik(bnext):
            qnext.töröl(b)
        vége ha
    vége minden
vége
```

A ***lekérdez*** *művelet*

```
Q.lekérdez(b):
    legyen találat[]
    minden obj-ra Q.objs[]-ből:
        ha b.ütközik(obj):
            találat.add(obj)
        vége ha
    vége minden
    minden qnext-re Q.{NW, NE, SW, SE}-ből:
        legyen bnext a qnext területének megfelelő téglalap
        ha b.ütközik(bnext):
            találat.add(qnext.lekérdez(b))
        vége ha
    vége minden
visszatérít találat
```

Hogyan lehet a műveleteket hatékonyabbá tenni?

Hogyan lehet a műveleteket hatékonyabbá tenni?

- felhasználjuk a **tartalmaz** műveletet a **töröl** és **lekérdez** műveleteknél, hogy ütközés-ellenőrzéseket spóroljunk meg

Hogyan lehet a műveleteket hatékonyabbá tenni?

- felhasználjuk a **tartalmaz** műveletet a **töröl** és **lekérdez** műveleteknél, hogy ütközés-ellenőrzéseket spóroljunk meg
- bevezetünk egy **levél** adattagot, ami igaz ha a csomópont levél, hamis különben. Így megspórolhatjuk feltételek kiértékelését egyes helyeken.

Hogyan lehet a műveleteket hatékonyabbá tenni?

- felhasználjuk a **tartalmaz** műveletet a **töröl** és **lekérdez** műveleteknél, hogy ütközés-ellenőrzéseket spóroljunk meg
- bevezetünk egy **levél** adattagot, ami igaz ha a csomópont levél, hamis különben. Így megspórolhatjuk feltételek kiértékelését egyes helyeken.
- implementációban:

Hogyan lehet a műveleteket hatékonyabbá tenni?

- felhasználjuk a **tartalmaz** műveletet a **töröl** és **lekérdez** műveleteknél, hogy ütközés-ellenőrzéseket spóroljunk meg
- bevezetünk egy **levél** adattagot, ami igaz ha a csomópont levél, hamis különben. Így megspórolhatjuk feltételek kiértékelését egyes helyeken.
- implementációban:
  - a csomópontokban nem tároljuk el magukat az objektumokat, csak rájuk mutató pointereket

Hogyan lehet a műveleteket hatékonyabbá tenni?

- felhasználjuk a **tartalmaz** műveletet a **töröl** és **lekérdez** műveleteknél, hogy ütközés-ellenőrzéseket spóroljunk meg
- bevezetünk egy **levél** adattagot, ami igaz ha a csomópont levél, hamis különben. Így megspórolhatjuk feltételek kiértékelését egyes helyeken.
- implementációban:
  - a csomópontokban nem tároljuk el magukat az objektumokat, csak rájuk mutató pointereket
  - rekurzió helyett iteratív megközelítést alkalmazunk

Hogyan lehet a műveleteket hatékonyabbá tenni?

- felhasználjuk a **tartalmaz** műveletet a **töröl** és **lekérdez** műveleteknél, hogy ütközés-ellenőrzéseket spóroljunk meg
- bevezetünk egy **levél** adattagot, ami igaz ha a csomópont levél, hamis különben. Így megspórolhatjuk feltételek kiértékelését egyes helyeken.
- implementációban:
  - a csomópontokban nem tároljuk el magukat az objektumokat, csak rájuk mutató pointereket
  - rekurzió helyett iteratív megközelítést alkalmazunk
- a hatékonyság függ a **MAXDEPTH** változó megválasztásától is

Kérdések…?

Köszönöm a figyelmet!