

Fakulta riadenia a informatiky  
Informatika

Semestrálna práca S1  
*Systém pre geodetov*  
2023

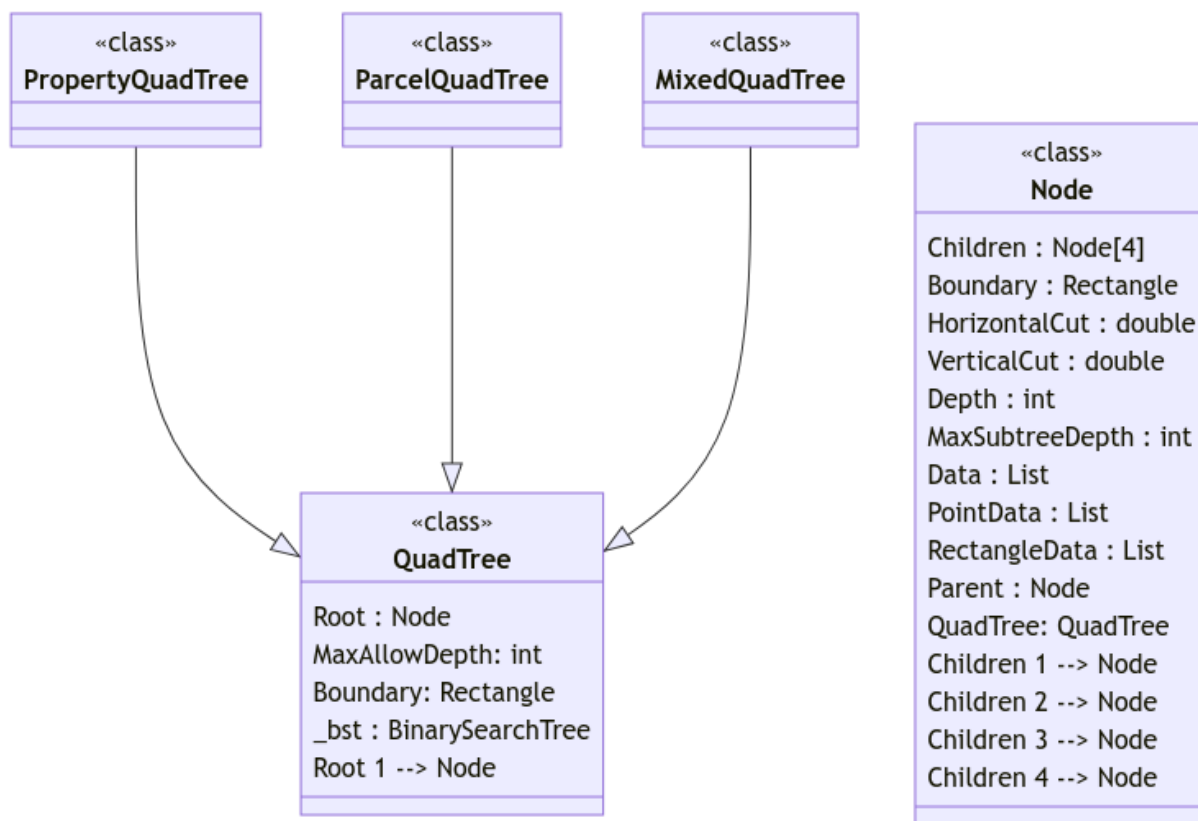
Ing. **Peter Jankovič**, PhD.  
PONDELOK 10, 11  
2023/2024

Maroš Gorný, 5ZIB11

## Obsah

Systém pre geodetov.....	1
2023 .....	1
Návrh aplikácie z pohľadu použitých údajových štruktúr .....	3
Uloženie parciel a nehnuteľností .....	3
QuadTree – Implementácia .....	5
Zdokonalenia.....	5
Optimalizácia.....	6
Výpočet zdravia.....	7
Diagram tried a ich popis.....	10
Aplikačná logika (Controller).....	10
GUI (View) .....	11
QuadTree údajová štruktúra (model) .....	12
Zložitosť operácií.....	13
QuadTree .....	13
Aplikačný systém.....	13
Intervalové hľadanie .....	14
Používateľská príručka .....	16
Vyhľadanie/vkladanie/vymazanie/odstránenie nehnuteľností/parcely .....	16
Import/export dát .....	17
Generovanie dát .....	17

## Návrh aplikácie z pohľadu použitých údajových štruktúr



### Uloženie parciel a nehnuteľností

Údajové štruktúry ktoré som použil boli **quad stromy**, **binárny vyhľadávací strom** a taktiež už implementované **listy** a **zásobníky** v programovacom jazyku **C#**. Vzhľadom na to, že sme potrebovali unikátnosť záznamov, vytvorili sme objekty v strome, ktoré niesli dvojicu **klúč – dáta**. Parcely a nehnuteľnosti sa do stromu pridávali na základe dvoch pozícií a to najviac ľavý dolný roh a najviac horný pravý roh, ktoré presne vymedzili priestor parcely alebo nehnuteľnosti. V strome teda vieme vyhľadávať pomocou daných súradníc.

Quad strom si pamätá maximálnu povolenú hĺbku, vymedzený priestor, kľúče a koreň stromu, ktorý si pamätá štyri deti - pod uzly, ktoré sú vlastne štyri kvadranty koreňa. Tieto uzly majú taktiež vymedzený priestor, ale pamätajú si aj rôzne iné veci, napríklad rez kvadrantov, svoju hĺbku, maximálnu hĺbku v podstrome, svojho otca, prípadne strom v ktorom sa nachádza. Taktiež sú v ňom uložené dáta, typu bod, alebo obdĺžnik.

### Quad strom

V aplikácii sme urobili **tri quad stromy**, jeden pre **parcely**, jeden pre **nehnuteľnosti** a jeden **spoločný**, kde sme pridávali aj parcely aj nehnuteľnosti. Vďaka tomuto rozdeleniu, sme mohli efektívnejšie vyhľadávať na základe typu objektu, buď parcely alebo

nehnutelnosti. Spoločný strom nám pomáha pri vyhľadávaní objektu podľa pozície, keď nám nezáleží na jeho type.

Tieto výhody však priniesli aj mierne zhoršenie komplexity pri vymazávaní, pretože vždy musíme objekt vymazať z dvoch stromov. Dopredu však vieme, že operácia vymazať sa bude vykonávať minimálne, preto sme boli ochotní podstúpiť toto mierne zhoršenie.

### Binárny vyhľadávací strom

Pre potreby obmedzenia duplicitnosti kľúčov, si quad strom pamätá všetky kľúče ktoré boli vložené do stromu a to práve v nami implementovanom binárnom vyhľadávacom strome. Na základe tejto informácie, vieme ešte pred vložením, prípadne vymazaním objektu zistiť, či je daný kľúč v strome, alebo nie.

Nakoľko je veľká pravdepodobnosť, že sa nové objekty budú vkladať postupne od najmenšieho kľúču po najväčší, alebo od najväčšieho kľúču po najmenší, môže sa stať, že náš strom zdegeneruje do lineárneho zoznamu. Preto by bolo v budúcnosti vhodné tento strom **zmeniť** napríklad na **Treap**, ktorý má oveľa menšiu šancu zdegenerovať na lineárny zoznam.

### List a zásobník

listy a zásobníky sme si neimplementovali a využili sme už vopred zadefinované implementácie jazyku C#.

Listy sme využívali väčšinou len ako úložný priestor a väčšinou sme v konečnom dôsledku museli prejsť v najhoršom prípade všetky objekty ktoré v ňom boli.

Zásobníky sme využili v prípadoch prehliadky, či už stromov, alebo podstromov.

# QuadTree – Implementácia

## Zdokonalenia

QuadTree, ktorý sme implementovali, sme implementovali tak, že sme **vopred nepoznali**, aké **dáta** v strome budú.

Neskôr sme sa však rozhodli pridať niektoré zdokonalenia, avšak je známe, že ak chceme niečo vylepšiť, je veľká pravdepodobnosť, že na druhej strane niečo zhoršíme. Rozhodli sme sa teda pre cestu, ktorá nám síce zaberie viac operačnej pamäte, ale zrýchli niektoré operácie a konkrétnejšie sme si vybrali implementáciu, ktorá zrýchli pridávanie a hľadanie priestorových objektov avšak naopak, mierne zhorší vymazávanie priestorových objektov.

## Kľúče v binárnom vyhľadávacom strome

Unikátne kľúče si strom drží v binárnom vyhľadávacom strome, ktorý je určite lepší ako obyčajný list, avšak bolo by vhodné, keby ho premeníme napríklad na **AVL** strom alebo **Treap** strom aby sme **zabránili degenerácii** na lineárny zoznam.

## Hĺbka uzla

Vzhľadom na to, že rýchlosť operácií v QuadTree závisí hlavne od výšky stromu, pridali sme do uzla vlastnosť, ktorá nám hovorí, v akej hĺbke sa uzol nachádza.

## Maximálna hĺbka podstromu

Okrem hĺbky uzla si vieme pozrieť aj to, aká je maximálna hĺbka podstromu daného uzla.

## Tri zoznamy dát

Pri vkladaní objektu do uzla, sme umožnili pridávať do **troch zoznamov**, jeden pre **body**, jeden pre **obdĺžniky** a jeden **spoločný** aj pre obdĺžniky aj pre body. Toto vylepšenie nám umožní rýchlejšie vyhľadávať objekt, pokiaľ vieme, aký typ má.

## Priestorové informácie uzla

Okrem vymedzeného priestoru, ktorý si uzol pamätá, si taktiež pamätá, svoj **rez horizontálneho** a **vertikálneho delenia**, ktoré nám výrazne pomáhajú zistiť rozdelenia kvadrantov.

## Efektívnejšie vymazávanie vnútorného uzla

Pri vymazávaní z vnútorného uzla, ktorý po vymazaní ostal prázdny, sme do tohto prázdneho uzla vložili objekt z podstromu, ktorý mal najväčšiu hĺbku, čím prispievame k zmenšeniu maximálnej výšky stromu.

## Zväčšenie/zmenšenie maximálnej výšky stromu

V prípade potreby, vieme zmenšiť alebo zväčšiť maximálnu výšku stromu. Pri zmenšení **nestrácame žiadne dáta**, len ich presúvame do posledného možného listu a pri zväčšení zase tieto dáta z listov presúvame nižšie.

## Prehliadky stromu s akciou

Naimplementovali sme tri prehliadky stromu, **in-order**, **pre-order** a **post-order**. Pri každej prehliadke sme taktiež umožnili spraviť **ľubovoľnú akciu nad daným uzlom**.

## Optimalizácia

Quad strom sme sa pokúsili optimalizovať na základe dát, ktoré sa v ňom nachádzali. Pri vedomosti aký typ prvkov v quad strome máme, sme lepšie vedeli prispôbiť rozdelenia kvadrantov, tak aby obsahovali čo najväčší počet prvkov a boli rovnomerné rozložené.

## Návrh

Vzhľadom na to, že pri quad strome sa obdĺžnik delí najprv na **vrchnú** a **spodnú polovicu** a potom na **právu** a **ľavú polovicu**, náš nápad bol, že sa pokúsime nájsť lepšie rozdelenie, pri ktorom rez pretína čo najmenej priestorových objektov.

Keďže sme teraz delili obdĺžnik na polovice, skúsili sme obdĺžnik deliť na **n-tiny**, pričom **n** sme pre celý strom určili rovnaké a rez ktorý sa preťal s čo najmenej objektmi, sme určili za hlavný. To isté sme spravili aj na vertikálnom aj horizontálnom reze. Týmto sme určili štyri kvadranty, na ktorých sme spravili to isté.

Pri delení sme však presúvali do daného kvadrantu len objekty ktoré sa tam naozaj dostanú a teda objekty, ktoré zostali na vyššej úrovni, sme nepočítali do počtu prierezov.

## Príklad

Pre vstup sme potrebovali **vymedzený priestor**, **objekty** ktoré sú v strome a určenú **porciu**, teda naše **n**. Ak **n** bolo napríklad 3, náš kvadrant sme **vertikálne rozdělili** na tri rovnaké časti. Následne sme spočítali ktorý rez sa pretína s najmenej objektami a tento **rez** sme určili **za hlavný**, to isté sme spravili aj **horizontálnym rozdelením**.

Keď sme mali určené rezy, spravili sme **4 kvadranty** podľa rezov a začali sme túto metódu **aplikovať** na naše **deti**, teda pod uzly. Do uzlov sme však pridávali len objekty, ktoré sa tam naozaj zmestia. Tie ktoré sa nezmestili, nechávame v uzle v ktorom sme.

## Zlepšenie rýchlostí pri operáciách

Pri testoch sme spustili **10 replikácií**, pričom v každej replikácii sme použili **10 000 objektov**, či už na operáciu **vlož**, **vymaž** alebo **nájdi**. Vždy sme vykonávali len jednu operáciu. Testy sme uskutočnili pomocou programu **dotTrace** od spoločnosti **JetBrains**.

### Vlož – Priemer za replikáciu

Pri operácii vložít sme dosiahli takmer **100%** zlepšenie.

QuadTree	Optimalizovaný QuadTree
5 531 ms	35 ms

### Vymaž – Priemer za replikáciu

Pri operácii vymaž sme dosiahli takmer **97%** zlepšenie.

QuadTree	Optimalizovaný QuadTree
4 507 ms	148 ms

### Nájdí – Priemer za replikáciu

Pri operácii nájdí sme dosiahli **93%** zlepšenie.

QuadTree	Optimalizovaný QuadTree
15 523 ms	1 235 ms

### Výsledok optimalizácie

**Výsledok optimalizácie** môžeme považovať **úspešný**, teda optimalizácia sa nám podarila a mala aj výrazný vplyv. Preto ak dopredu poznáme dáta, ktoré budú do stromu vkladané, môžeme použiť nami zvolenú optimalizáciu, ktorá celý **systém zrýchli**. Ak počítame všetky operácie dokopy, priemerne nám zlepšili operácie takmer o 95%. V prípade potreby, je pri zdrojovom kóde priložený aj **snapshot z profilera**.

### Výpočet zdravia

Pre výpočet zdravia sme dávali do úvahy dva parametre. Prvý bol **maximálna výška stromu** a druhý bol **počet objektov v uzly**.

#### Výpočet zdravia podľa počtu prvkov v uzly

Výsledok sa pohybuje v intervale **0 až 1**, pričom 1 je najlepší výsledok a 0 najhorší. V prípade, že je uzol list a neobsahuje žiadne objekty, do výpočtu tento uzol nerátame.

$$\frac{1.0}{1 + \left( \frac{\log(Data.Count)}{\log(2)} \right) * ScalingFactor}$$

- **Data.Count** je počet prvkov v uzle.

- **ScalingFactor** je škálovací faktor ktorý nám hovorí, ako veľmi výsledok ovplyvní výsledok zdravia. V prípade, že je väčší ako 1, zväčší to vplyv, ak je menší ako 1, ale väčší ako 0, vplyv sa zmenší, teda zdravie bude klesať pomalšie. My sme používali škálovací faktor **0.1**.

### Výpočet ideálnej výšky stromu

V QuadStrome vieme, že každý level pridávame do stromu  $4^n$  počet objektov, pričom  $n$  je výška stromu. Pri výške 0, pridáme 1 objekt, pri výške 1 pridáme 4 objekty atď. Vďaka tomuto si vieme odvodiť ideálnu výšku stromu, pokiaľ vieme počet prvkov ktorý sa v strome bude nachádzať. V prípade, že mal strom 20 objektov, začneme si počítať **kumulatívne počet objektov pre daný level** až pokiaľ nenarazíme na sumu väčšiu alebo rovnú ako je počet prvkov. Počítame teda  $1 + 4 + 16 = 21$ , čo je väčšie ako 20 a prešli sme cez level 0, 1 a 2. Ideálna výška je teda 2.

### Výpočet zdravia podľa výšky stromu

Výsledok sa pohybuje v intervale **0 až 1**, pričom 1 je najlepší výsledok a 0 najhorší. Výsledok sa počíta na základe rozdielu ideálnej výšky stromu s aktuálnou maximálnou výškou stromu.

$$\frac{1.0}{1 + \left( \frac{\log(\text{depthDifference})}{\log(2)} \right) * \text{ScalingFactor}}$$

- **depthDifference** je rozdiel maximálnej výšky stromu a ideálnej výšky stromu.
- **ScalingFactor** je škálovací faktor ktorý nám hovorí, ako veľmi výsledok ovplyvní výsledok zdravia. V prípade, že je väčší ako 1, zväčší to vplyv, ak je menší ako 1, ale väčší ako 0, vplyv sa zmenší, teda zdravie bude klesať pomalšie. My sme používali škálovací faktor **0.1**.

### Výpočet celkového zdravia

Pri výpočte sme brali do úvahy aj zdravie podľa počtu dát aj podľa výšky stromu. Tieto dve hodnoty sme spočítali a vydělili dvoma, čím nám vyšiel priemer týchto hodnôt.

### Zdravie neoptimalizovaného QuadTree

Zdravie sme vypočítali ako priemer **10 normálnych quad stromov**, pričom každý strom mal **10 000 náhodne generovaných prvkov**.

```
Normal tree average health over 10 runs:
(Combined: 0,7253049813636655, Data: 0,9940373407172318, Depth: 0,4565726220100988)
```

### Zdravie optimalizovaného QuadTree

Zdravie sme vypočítali ako priemer **10 optimalizovaných quad stromov**, pričom každý strom mal **10 000 náhodne generovaných prvkov**.

```
Optimized tree (portion 2) average health over 10 runs:
(Combined: 0,8139357032632979, Data: 0,7630786244818047, Depth: 0,8647927820447912)
```



Taktiež sme sa pokúsili porovnať zdravie aj pri rozdeľovaní na menšie porcie, kde sa však ukázalo, že **s väčším počtom sa nám zdravie len zhoršuje**.

```
Tree health test (Combined, Data, Depth)
Normal tree health: (0,725027888150418, 0,9940670708142255, 0,45598870548661047)
Optimized 2 portions: (0,8123936013884441, 0,7615992555239293, 0,8631879472529589)
Optimized 3 portions: (0,7467817518342823, 0,6989643938637502, 0,7945991098048143)
Optimized 4 portions: (0,719152097642842, 0,6789984286313682, 0,759305766654316)
Optimized 5 portions: (0,7101450981114024, 0,6696481023176618, 0,7506420939051428)
Optimized 6 portions: (0,6905356882636875, 0,662003377016559, 0,7190679995108161)
Optimized 7 portions: (0,6836613335863393, 0,657471637736888, 0,7098510294357905)
Optimized 8 portions: (0,6820271340384956, 0,6542032386412008, 0,7098510294357905)
Optimized 9 portions: (0,6785483764373239, 0,6552402905530551, 0,7018564623215926)
Optimized 10 portions: (0,6702881071153455, 0,6520398875185288, 0,6885363267121621)
Optimized 20 portions: (0,6529285885531912, 0,6531978030334282, 0,6526593740729544)
Optimized 40 portions: (0,6340829314312355, 0,6519780681306017, 0,6161877947318694)
Optimized 60 portions: (0,6178945177405528, 0,6425981899765175, 0,593190845504588)
Optimized 80 portions: (0,607784857627786, 0,6406032080951765, 0,5749665071603957)
Optimized 100 portions: (0,6038347658156251, 0,639484899714783, 0,5681846319164673)
```

### Výsledok zlepšenia

Na základe vykonaných výpočtov a výsledkov, ktoré sme dostali, môžeme prehlásiť, že výsledok zdravia sa pri optimalizovanom strome **pri delení na 2 porcie, zlepšil o približne 12%**.

Ak sa ale jedná o **výšku stromu**, tá sa v priemere zlepšila približne o **89%** a ak sa jedná o **počet dát v uzly**, zdravie sa v priemere **zhoršilo o -23%**.

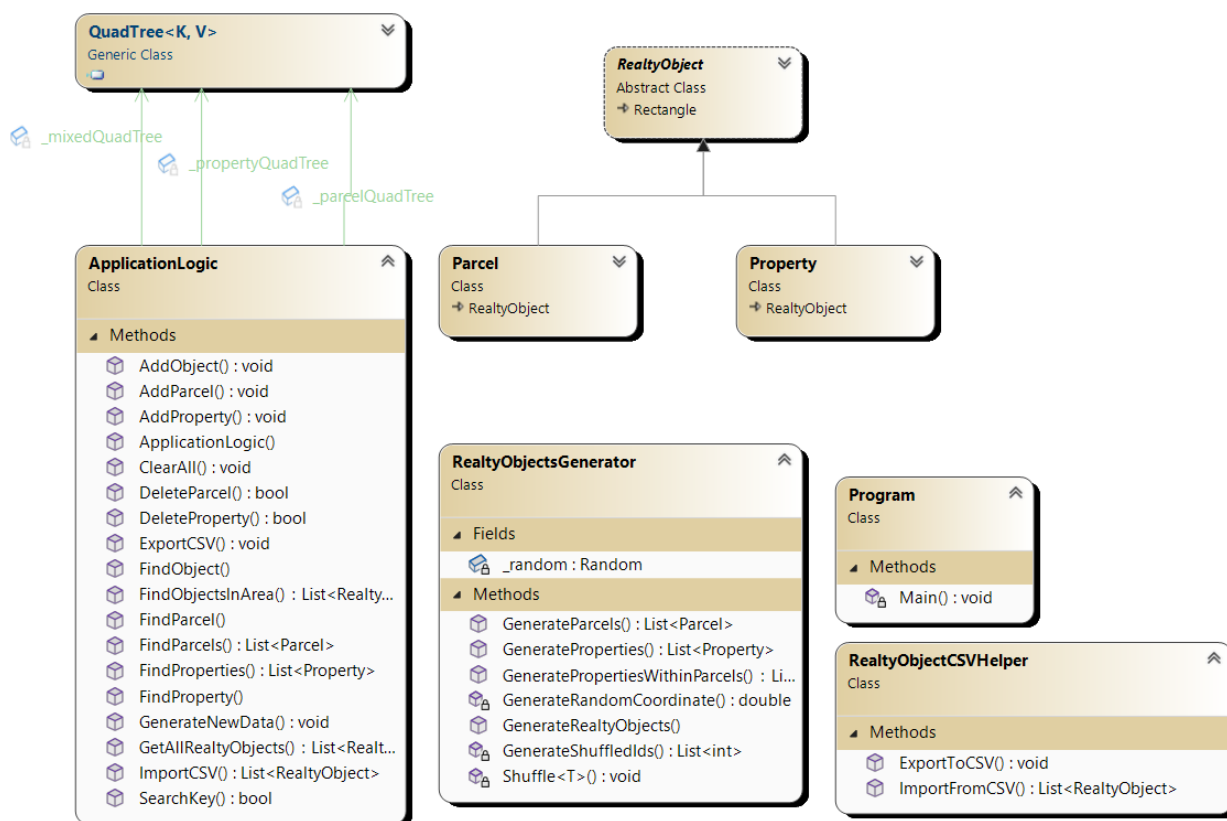
## Diagram tried a ich popis

Diagram tried si rozdelíme na tri časti z ktorých sa skladá naša aplikácia. Aplikácia je postavená na architektúre **MVC (Model – View – Controller)** a preto si postupne predstavíme práve tieto komponenty.

### Aplikačná logika (Controller)

Aplikačná logika je kontroler v našej architektúre. Predstavuje teda logiku aplikácie, ktorá spája grafické rozhranie a model. Nachádzajú sa v nej rôzne triedy, napríklad **ApplicationLogic**, v ktorej sme sa snažili držať hlavnú logiku programu, následne pomocné triedy na **generovanie**, **importovanie** alebo **exportovanie** dát a samotné typy pre nehnuteľnosti a parcely.

V aplikačnej logike máme aj triedy ktoré skúmajú výkon optimalizácie alebo testujú správnosť vytvorených nehnuteľností a parciel. Tieto triedy však pre používateľa nie sú dôležité, preto sme ich do UML diagramu nezahrnuli.



## GUI (View)

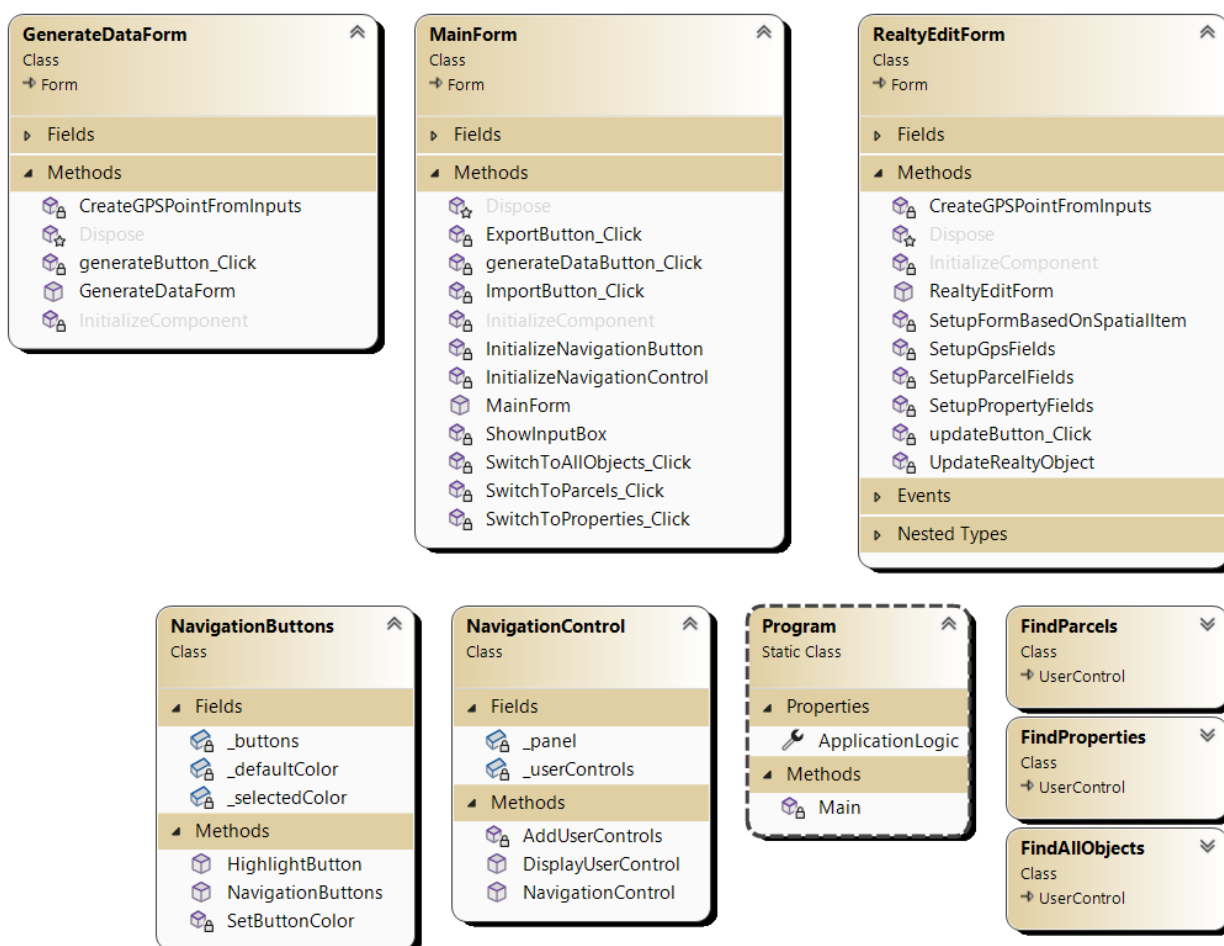
GUI je naše grafické rozhranie, takže náš **pohľad**, ktorý je zodpovedný za to, aby používateľovi ukázal dáta v korektnom formáte. GUI tiež spracováva používateľské **vstupy** a posielajú tieto **požiadavky kontroleru**.

V grafickom rozhraní máme tri krát **Form**, tri krát **UserControl** a následne triedy ktoré nám pomáhajú v navigácii alebo zvýraznení tlačidiel. Form je nové okno a UserControl je len nový obsah v danom okne. Následne tu máme triedu Program, ktorá cele GUI spúšťa.

**MainForm** je trieda, a teda aj okno, v ktorom beží jadro aplikácie, a z ktorého môžeme vykonávať ďalšie akcie, alebo otvárať nové okná, napríklad vyhľadávať nehnuteľnosti, exportovať a importovať dáta, alebo otvorenie okna na generovanie dát.

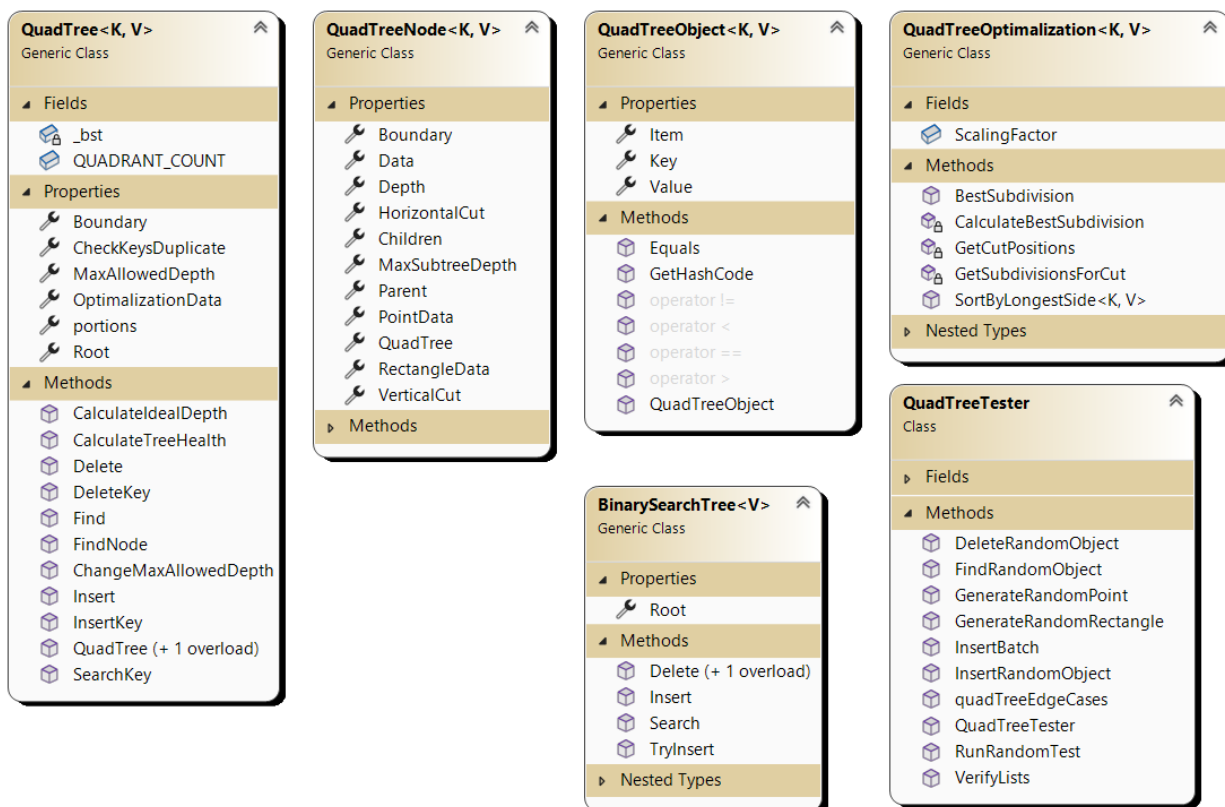
**GenerateDataForm** je trieda ktorá nám pomáha generovať dáta na základe určeného vymedzeného priestoru a taktiež počtu parciel alebo nehnuteľnosti.

**RealtyEditForm** je trieda určená len na úpravu objektu a teda po otvorení sa nám zobrazia jeho vlastnosti, ktoré máme možnosť upraviť.

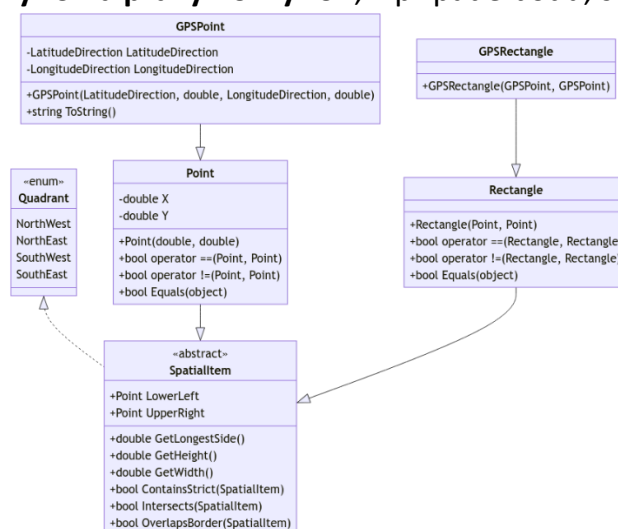


## QuadTree údajová štruktúra (model)

V projekte QuadTree, ktorý je náš **model**, spracováva logiku, pravidlá a dáta našej aplikácie. Naimplementovali sme si vlastný **QuadTree**, jeho **Node** a **Object** ktorý do neho vkladáme. Taktiež sme si museli implementovať **binárny vyhľadávací strom**, vďaka ktorému vieme efektívnejšie ukladať kľúče. Okrem iného máme aj triedu na **optimalizáciu** QuadTree alebo na overenie funkčnosti QuadTree metód.



QuadTree je však založený na **priestorových objektoch**, bod, alebo obdĺžnik. Vďaka týmto objektom vieme strom rozdeliť na **kvadranty**. Priestorový objekt si pamätá svoje dve pozície a to **pravý dolný roh** a **pravý horný roh**, v prípade bodu, sú tieto pozície rovnaké.



# Zložitosť operácií

## QuadTree

### Vlož

V prípade **vyváženosti** quad a binárneho vyhľadávacieho **stromu**

- $O(\log K + \log N)$
- K – Počet kľúčov v strome
- N – Počet prvkov v strome

### Vymaž

V prípade **vyváženosti** quad a binárneho vyhľadávacieho **stromu**

- $O(\log K + \log N + (\log N * D1 + D2))$
- K – Počet kľúčov v strome
- N – Počet prvkov v strome
- D1 – Počet dát daného typu v uzle (obdĺžnik alebo bod)
- D2 – Počet všetkých dát v uzle

### Nájdí

Silne závisí nad oblasťou, nad ktorou sa majú prvky hľadať a **nie je možné úplne vyčíslit** zložitosť, môžeme sa k nej ale **priblížiť**

- $O(N * D)$
- N – Počet uzlov do ktorého oblasť zasahuje
- D – Počet dát v prejdennom uzle

## Aplikačný systém

V reálnom svete, je veľmi **nepravdepodobné**, aby sme dosiahli **vyváženosť stromu**, preto vypočítané zložitosti budú pravdepodobne horšie, nevieme však povedať o koľko, nakoľko to závisí od typu dát v systéme.

### Vlož

**Približný odhad** zložitosti, pričom vložený objekt musíme spojiť s parcelou alebo nehnuteľnosťou

- $O(2 * \text{QuadTreeVlož} + \text{QuadTreeNájdí} + F)$
- QuadTreeVlož –  $O(\log K + \log N1)$
- QuadTreeNájdí –  $O(N2 * D)$
- F – Počet nájdených dát spojených s objektom vkladania
- K – Počet kľúčov v strome
- N1 – Počet prvkov v strome
- N2 – Počet uzlov do ktorého oblasť zasahuje
- D – Počet dát v prejdennom uzle

## Vymaž

**Približný odhad** zložitosti, pričom odstránenému objektu musíme vymazať spojitosti s parcelou alebo nehnuteľnosťou, pričom vymazávanie zo stromu, kde sú uložené len rovnaké typy vymazávaného objektu bude trvať kratšie ako vymazávanie zo stromu, kde sú všetky objekty

- $O(2 * \text{QuadTreeVymaž} + F)$
- $\text{QuadTreeVymaž} - O(\log K + \log N + (\log N * D1 + D2))$
- K – Počet kľúčov v strome
- N – Počet prvkov v strome
- D1 – Počet dát daného typu v uzle (obdĺžnik alebo bod)
- D2 – Počet všetkých dát v uzle
- F – Počet nájdených dát spojených s vymazaným objektom

## Nájdí nehnuteľnosť/parcelu

Hľadanie nám stačí uskutočniť len nad stromom, kde je uložený daný typ objektu

- $O(\text{QuadTreeNajdi})$
- $\text{QuadTreeNajdi} - O(N * D)$
- N – Počet uzlov do ktorého oblasť zasahuje
- D – Počet dát v prejdennom uzle

## Nájdí všetky objekty

Hľadanie nám stačí uskutočniť len nad stromom, kde sú uložené všetky objekty

- $O(\text{QuadTreeNajdi})$
- $\text{QuadTreeNajdi} - O(N * D)$
- N – Počet uzlov do ktorého oblasť zasahuje
- D – Počet dát v prejdennom uzle

## Uprav

V prípade, že operácia uprav mení **len popis** objektu, zložitosť je  **$O(\text{QuadTreeNajdi})$** , ak sa však mení identifikačné číslo, alebo pozícia objektu, objekt sa musí vymazať a následne na novo pridať, pre jednoduchosť, si napíšeme len skrátený výraz, ktorý už však bol rozložený vyššie

- $O(\text{AppVymaž} + \text{AppVlož})$

## Intervalové hľadanie

Zložitosť operácií nájdí sme si už vysvetlili, teraz si vysvetlíme postup, najprv na quad strome a neskôr postup v našej aplikácii.

## QuadTree

Pri **intervalovom vyhľadávaní** potrebujeme **vymedziť priestor**, v ktorom sa bude vyhľadávať, preto do metódy vstupuje ako parameter **obdĺžnik**, ktorý jasne vymedzuje priestor hľadania.

Keď máme vymedzený priestor, musíme si **nájsť všetky uzly**, ktoré do tohto priestoru **zasahujú**, to znamená, že uzol sa nachádza vnútri vymedzeného priestoru, alebo doň čiastočne zasahuje.

V prípade, že sa uzol nachádza úplne celý vo vymedzenom priestore, môžeme všetky jeho objekty ktoré sa v ňom nachádzajú, priradiť do množiny nájdených výsledkov a presunúť sa na deti, pri ktorých je kontrolovaná tá istá podmienka.

V prípade, že sa uzol nachádza len čiastočne vo vymedzenom priestore, musíme skontrolovať, či sa v tomto priestore nachádzajú aj všetky jeho objekty ktoré sa v uzle nachádzajú a pridať len tie objekty, ktoré do vymedzeného priestoru zasahujú. Následne môžeme pokračovať na deti, pri ktorých je kontrolovaná tá istá podmienka.

V neposlednom rade, ak je uzol mimo vymedzeného priestoru, môžeme uzol zahodiť a nemusíme ďalej hľadať údaje v tomto uzle.

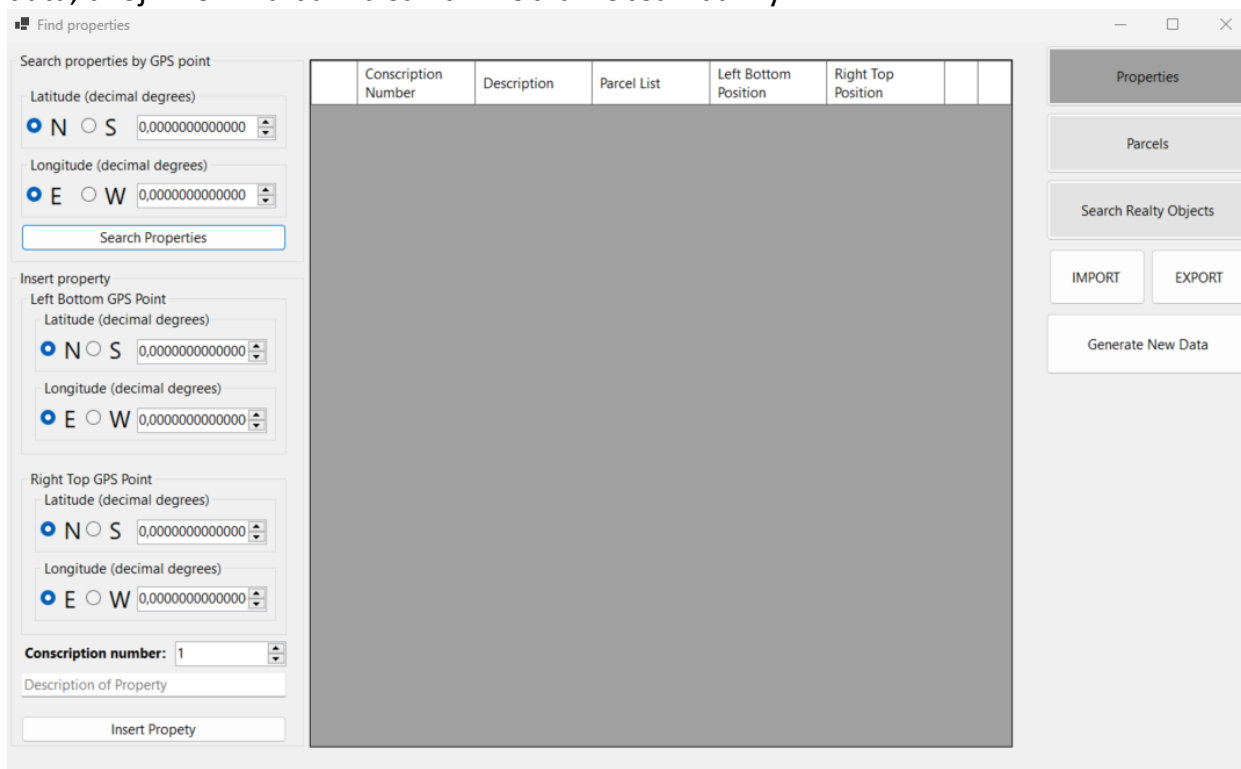
## Aplikácia

V našej aplikácii, nám priestor vymedzujú **dva GPS body**, ktoré sú súradnice a obsahujú **zemepisnú šírku a dĺžku**. Vďaka tomu, že vieme, či hľadáme parcelu, nehnuteľnosť, alebo všetky objekty, môžeme naše intervalové vyhľadávanie **obmedziť** buď **na jeden typ**, alebo vyhľadávať **všetky objekty**.

Vyhľadávanie sa však nemení, v prípade vyhľadávania daného typu, sa vyhľadáva len v strome, ktorý tento typ obsahuje a v prípade vyhľadávania všetkých objektov sa vyhľadáva v strome, kde sú všetky objekty.

## Používateľská príručka

Po spustení programu sa zobrazí **hlavné okno** kde na boku môžeme vyberať, či sa chceme venovať **nehnutelnostiam, parcelám, všetkým objektom**, alebo chceme **vygenerovať, exportovať, alebo importovať** dáta. Úvodné okno je tiež oknom pre prácu s nehnuteľnosťami. V prípade, že by sme po zobrazení dát v tabuľke nevideli kompletne dáta, dvojklikom na bunku sa nám zobrazí obsah bunky.







### Vyhľadanie/vkladanie/vymazanie/odstránenie nehnuteľností/parcely

V prípade **vyhľadávania** nehnuteľností/parcely stačí zadať **GPS pozíciu** a zobrazia sa nám všetky nehnuteľnosti/parcely ktoré sa na nej nachádzajú.

Ak chceme nehnuteľnosť/parcelu **pridať**, musíme zadať **2 GPS pozície**, pričom prvá je ľavý dolný roh vymedzeného priestoru a druhá je zas pravý horný roh.

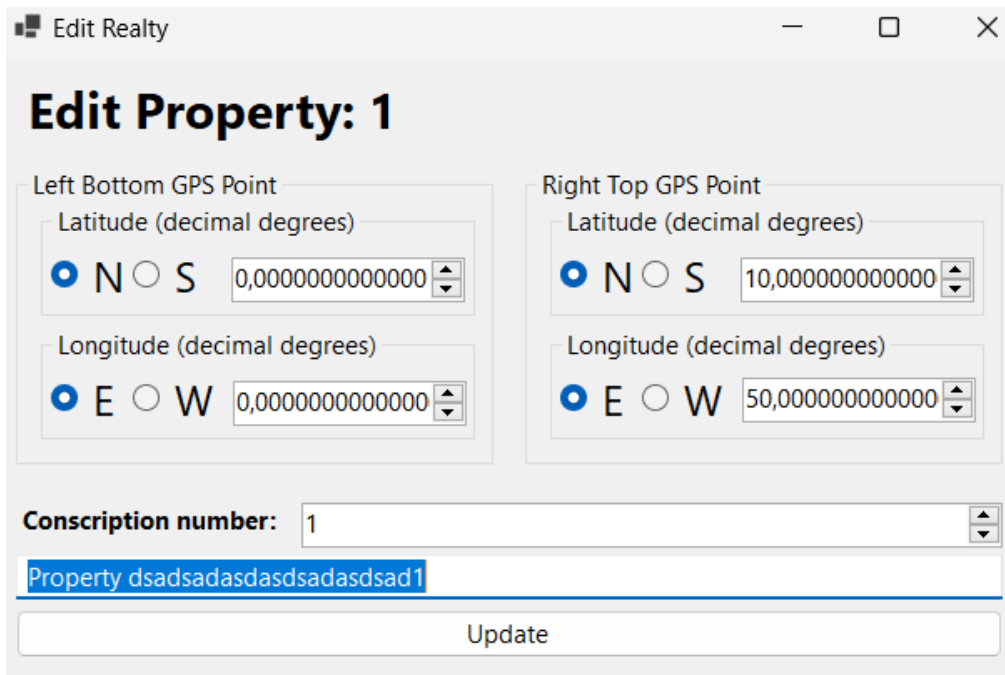
Ak chceme dáta **vymazať** alebo **upraviť**, najprv si ich musíme vyhľadať a následne kliknúť na **ikonku editácie** alebo **vymazania**.

	Conscription Number	Description	Parcel List	Left Bottom Position	Right Top Position		
▶ 1	1	Property dsadsa...	1	N 0, E 0	N 10, E 50		
2	11	Property 11	1	N 0, E 0	N 10, E 50		

Ak zvolíme vymazanie, aplikácia sa nás spýta, či objekt naozaj chceme odstrániť a ak odstránenie potvrdíme, objekt sa odstráni. Ak zvolíme editáciu, objaví sa nám ďalšie okno,



v ktorom môžeme vykonávať potrebné zmeny, avšak musíme ich potvrdiť tlačidlo na aktualizáciu.



**Edit Realty**

## Edit Property: 1

Left Bottom GPS Point

Latitude (decimal degrees)

☒ N ☐ S 0,00000000000000

Longitude (decimal degrees)

☒ E ☐ W 0,00000000000000

Right Top GPS Point

Latitude (decimal degrees)

☒ N ☐ S 10,00000000000000

Longitude (decimal degrees)

☒ E ☐ W 50,00000000000000

Conscription number: 1

Property dsadsadasdsadasdsad1

Update

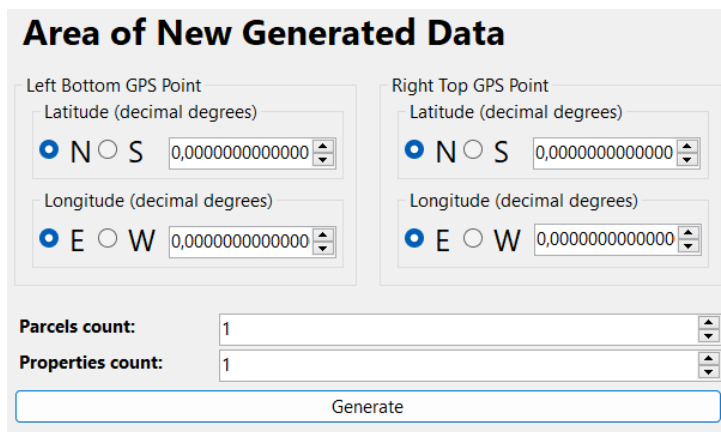
## Import/export dát

V prípade, že chceme **uložiť daný stav**, môžeme zvoliť **export** dát, v ktorom sa nám dáta uložia do **CSV** súboru. Lokáciu, kde sa má súbor uložiť a jeho názov si vyberieme v kontextovom menu.

Ak chceme dáta zase naspäť **načítať**, zvolíme **import** dát, kde musíme nájsť daný **CSV** súbor a potvrdiť výber.

## Generovanie dát

V prípade **generovaní** dát, si musíme **zvoliť oblasť**, v ktorej sa budú generovať a taktiež **počet** nehnuteľností a parciel ktoré chceme generovať.



## Area of New Generated Data

Left Bottom GPS Point

Latitude (decimal degrees)

☒ N ☐ S 0,00000000000000

Longitude (decimal degrees)

☒ E ☐ W 0,00000000000000

Right Top GPS Point

Latitude (decimal degrees)

☒ N ☐ S 0,00000000000000

Longitude (decimal degrees)

☒ E ☐ W 0,00000000000000

Parcels count: 1

Properties count: 1

Generate