Kierunek: Informatyka
Semestr: VIII
Rok akademicki: 2020/21
Przedmiot: Studio projektowe

Temat:
Smart Evacuation

Autorzy:
Grzegorz Kunc
Adam Szreter

Problem

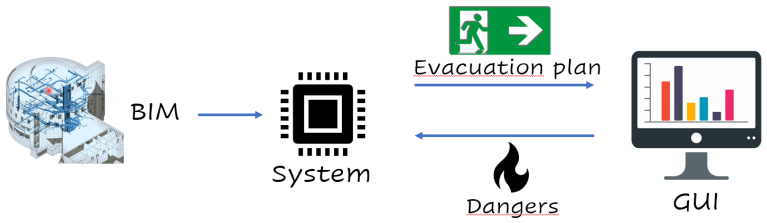
W obecnych czasach budynki stają się coraz bardziej skomplikowane. Poruszanie się po nich można porównać do chodzenia po labiryncie. Problem pojawia się, gdy w budynku pojawi się zagrożenie. Ewakuacja powinna przebiegać sprawnie, a każda osoba powinna udać się do wyznaczonego wyjścia. Jednak w wyniku paniki, ludzie mogą zapomnieć planów ewakuacyjnych. Same plany są zaś tworzone statycznie, z założeniem że wszystkie pomieszczenia i przejścia w budynku będą możliwe do pokonania w sytuacji zagrożenia - co nie musi być prawdą. Rozwiązaniem tego problemu może być inteligentny system ewakuacyjny, na bieżąco wyznaczający optymalne trasy ewakuacji z uwzględnieniem pojawiających się zagrożeń, wykrywanych dzięki coraz bardziej rozbudowanym sieciom czujników w budynkach.

Cel Projektu

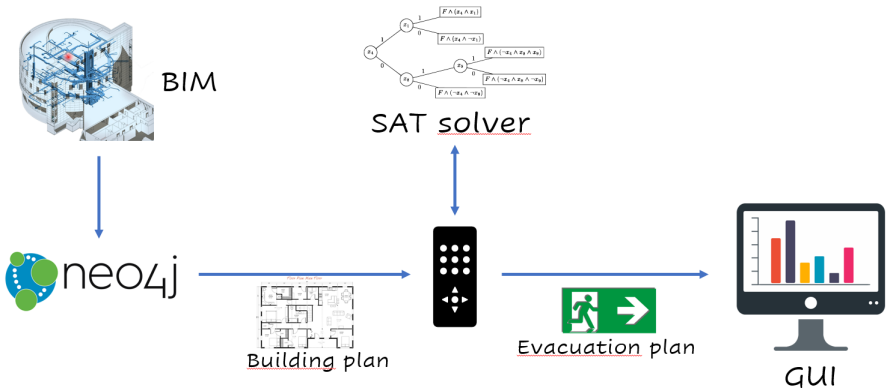
Celem projektu jest stworzenie aplikacji, która będzie w stanie:

- przyjąć od użytkownika plik IFC opisujący budynek i załadować go do grafowej bazy danych
- wygenerować i wyświetlić prosty plan budynku (opisujący logiczną strukturę pomieszczeń)
- wyznaczyć plan ewakuacji budynku za pomocą solvera SAT
- zaprezentować plan ewakuacji użytkownikowi
- przyjmować (poprzez dedykowane GUI) informację o symulowanych zagrożeniach i dostosowywać plan ewakuacji

Funkcjonalność aplikacji można przedstawić graficznie:



Jej architektura ma wyglądać następująco:

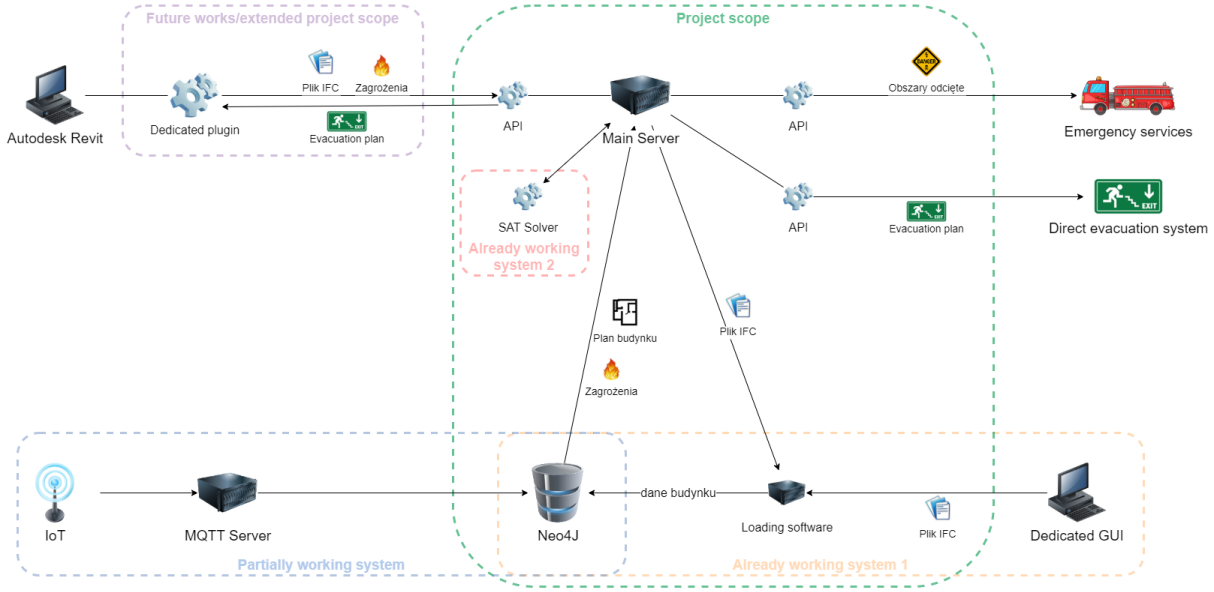


Szersza wizja systemu

Tworzona aplikacja ma w założeniu stworzyć podwaliny pod bardziej rozbudowany system o znacznie większym zastosowaniu praktycznym. Miałby on posiadać dwa tryby pracy: wsparcia projektowania budynku oraz monitorowania budynku i realizować w nich następujące dodatkowe funkcjonalności:

- w trybie projektowania budynku - realizacja przez stworzenie dedykowanych wtyczek do popularnych programów służących do edycji BIM (np. Autodesk Revit):
    - umożliwiać załadowanie pliku IFC do systemu bezpośrednio z poziomu programu do edycji BIM
    - wyświetlać w tym programie aktualny plan ewakuacji (jako nakładkę)
    - umożliwiać wprowadzanie zagrożeń bezpośrednio z poziomu programu do edycji BIM
- w trybie monitorowania budynku:
    - przyjmować dane pomiarowe z zamontowanej w budynku sieci czujników IoT
    - rozpoznawać na ich podstawie pojawiające się w budynku zagrożenia
    - udostępniać API dla urządzeń typu “inteligentna strzałka” w budynku prezentujących aktualny plan ewakuacji
    - udostępniać API dla służb ratunkowych pozwalające zlokalizować obszary budynku odcięte od dróg ewakuacji

Architektura rozszerzonego systemu mogłaby się prezentować następująco:



W dalszej perspektywie, interesujące mogłoby być rozwinięcie systemu o zdolność przewidywania rozchodzenia się zagrożeń w budynku (na użytek trybu monitorowania). Pozwoliłoby to znacznie lepiej zarządzać ewakuacją w przypadku zagrożeń, które z natury potrafią się przemieszczać/rozprzestrzeniać (przede wszystkim pożary, ale nie tylko - mogłoby to również dotyczyć np. ataków terrorystycznych). Pliki BIM zawierają niezwykle bogate i dokładne informacje o budynku, co pozwala mieć realną nadzieję że takie przewidywania mogłyby być dokładne.

Innym kierunkiem rozwoju mogłoby być inteligentne wsparcia użytkownika w zakresie planowania rozmieszczenia czujników w budynkach (w trybie projektowania). Można by sobie wyobrazić, że dla zadanej sieci czujników system przeprowadzi symulację zagrożeń powstających w budynku i momentu ich wykrycia oraz wpływu tego momentu na skuteczność ewakuacji (na podstawie wyznaczonego przez siebie planu ewakuacji). Takie zagrożenia mogłyby pojawiać się zupełnie losowo lub z rozkładem prawdopodobieństwa uwzględniającym opisaną w pliku BIM konstrukcję budynku (np. wycieki gazu i gwałtowne pożary mają większe szanse pojawić się w pobliżu instalacji gazowej budynku).

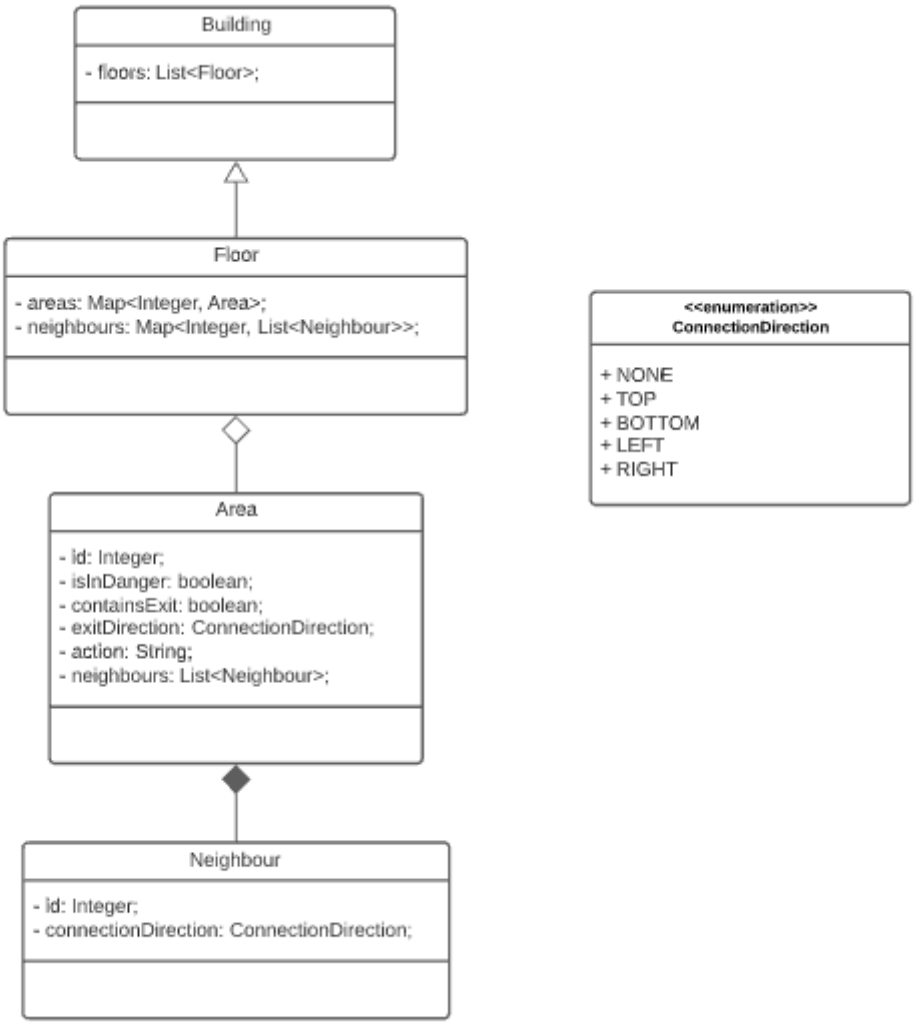
Na powyższym schemacie pokazana jest główna idea działania systemu. Mając dane budynku z pliku BIM jestesmy w stanie wygenerowac mape budynku. Za pomocą różnego rodzaju czujników (np. dymu, temperatury, jakości powietrze) system jest w stanie ustalić czy w dowolnym pomieszczeniu pojawiło się zagrożenie. Na tej podstawie droga ewakuacyjna jest aktualizowana, aby kierować ludzi do najbliższego możliwego wyjścia. W przypadku braku możliwości ewakuacji zostaje wyświetlona informacja o pozostaniu w aktualnym pomieszczeniu.

Starsze projekty

Projekt stanowi rozwinięcie dokonań naszych poprzedników - w szczególności warto zajrzeć do:

- Dolata W., Wawrzynów P. *“Adaptacyjny model ewakuacji”*, 2020 - stąd zaczerpnęliśmy obsługę solvera SAT i podstawowe wyświetlanie budynku
- Prażuch M. *“Integracja technologii BIM z Internetem Rzeczy”*, 2016 - świetne wprowadzenie w to czym w ogóle jest BIM i jak jest zbudowany plik IFC

Model budynku



Powyższy diagram klas jest odpowiedzialny za odpowiednia strukture budynku. Mozna zauwazyc, ze kazdy budynek jest zbudowany z pięter, w ktorych sklad wchodza pokoje, które posiadają pola odpowiedzialne za reprezentacje sąsiedztwa, możliwych akcji, a także informacji o wyjściu czy niebezpieczeństwie.

Tworzenie mapy budynku

Przy tworzeniu mapy budynku została wykorzystana biblioteka JavaFX. Pozwala ona na dynamiczne tworzenie widoku poprzez tworzenie odpowiednich komponentów w kodzie, a nie tylko widoku. Na wczesnym etapie projektu zostały zdefiniowane mocki kilku prostych budynków.

Budynek tworzony jest na podstawie zdefiniowania obiektów pokoi, które posiadają połączenia między sobą. Na tej podstawie rekurencyjny algorytm tworzenia budynku jest w stanie wygenerować pokoje w odpowiednim rozmieszczeniu.

Przykład prostego budynku, który posiada 4 pokoje i dwa piętra:

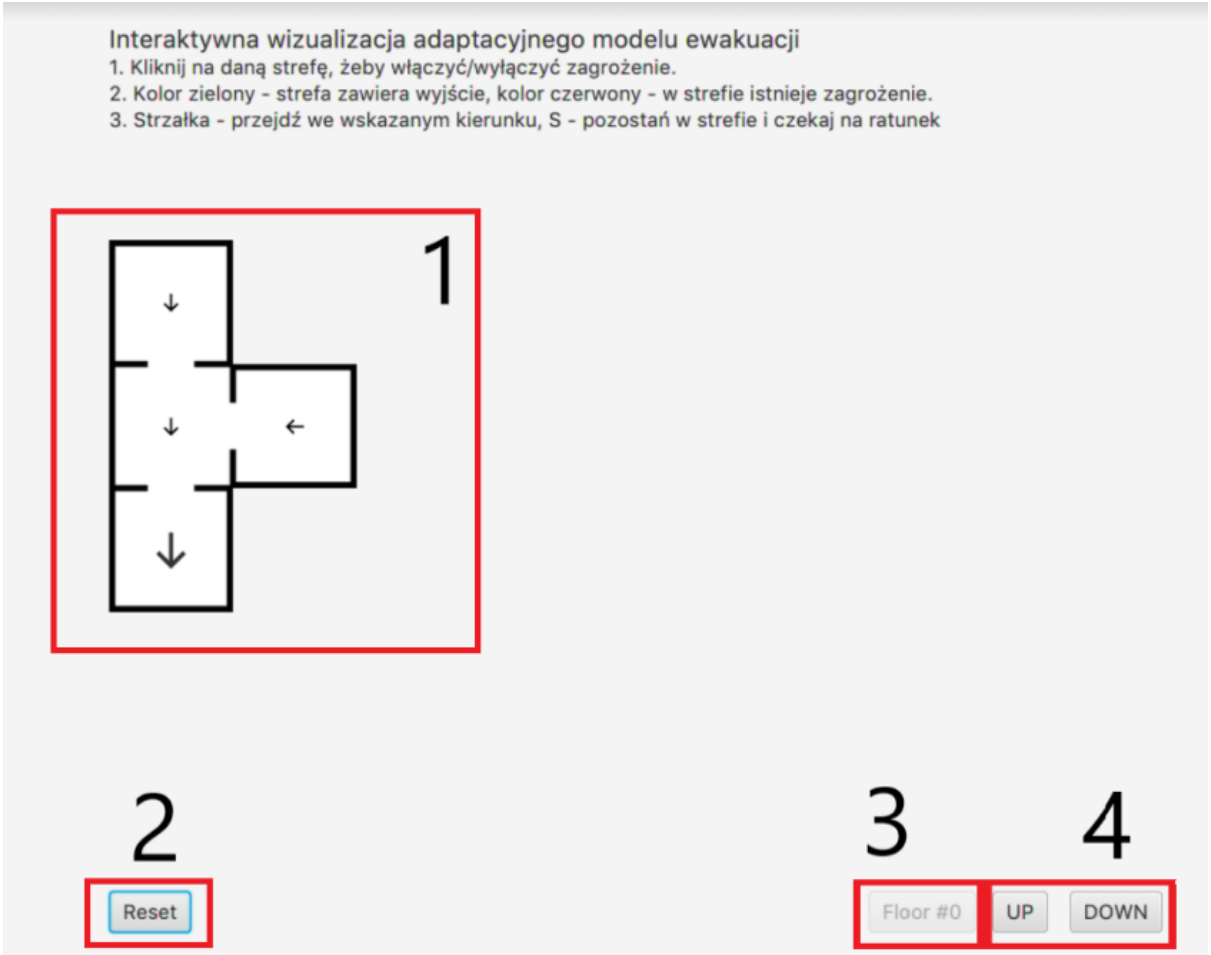
```
//      [0]
//      [1][2]
//      [3]
//
//       Exit in 3 on bottom wall
//       3 Floors
public static Building getSmallBuilding() {
    Building building = new Building();
    Floor floor = new Floor();

    for(int i = 0; i < 3; i++) {
        floor.addArea(i,  isInDanger: false);
    }
    floor.addArea( id: 3,  isInDanger: false, ConnectionDirection.BOTTOM);

    floor.createConnection( area1: 0,  area2: 1, ConnectionDirection.BOTTOM);
    floor.createConnection( area1: 1,  area2: 2, ConnectionDirection.RIGHT);
    floor.createConnection( area1: 1,  area2: 3, ConnectionDirection.BOTTOM);

    building.addFloor(floor);
    building.addFloor(floor);
    building.addFloor(floor);
    return building;
}
```

W wyniku odpalenie aplikacji z tym przykładem otrzymamy poniższy ekran:



1. Wizualizacja budynku
2. Przycisk do resetu symulacji
3. Numer aktualnie wyświetlanego piętra
4. Przyciski do przełączania między piętrami

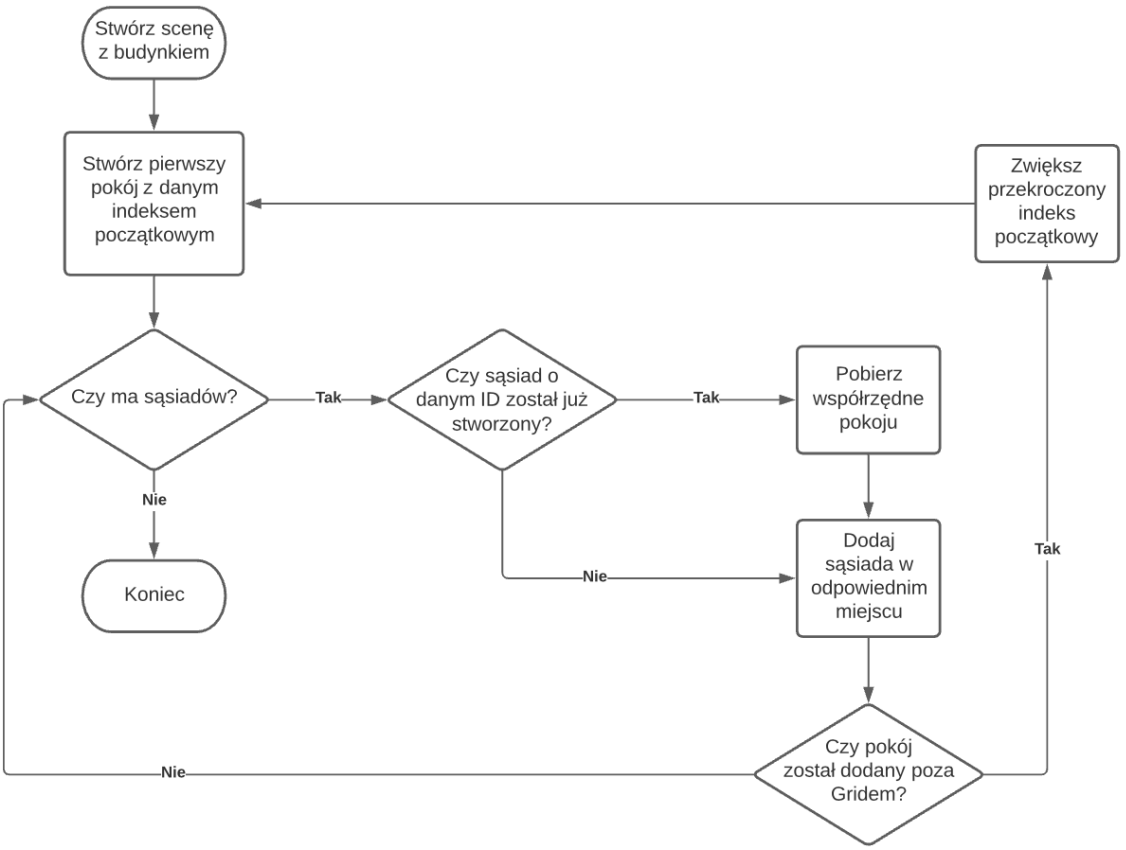


Diagram aktywności tworzenia dynamicznego ekranu budynku.

Generacja ścieżki ewakuacji

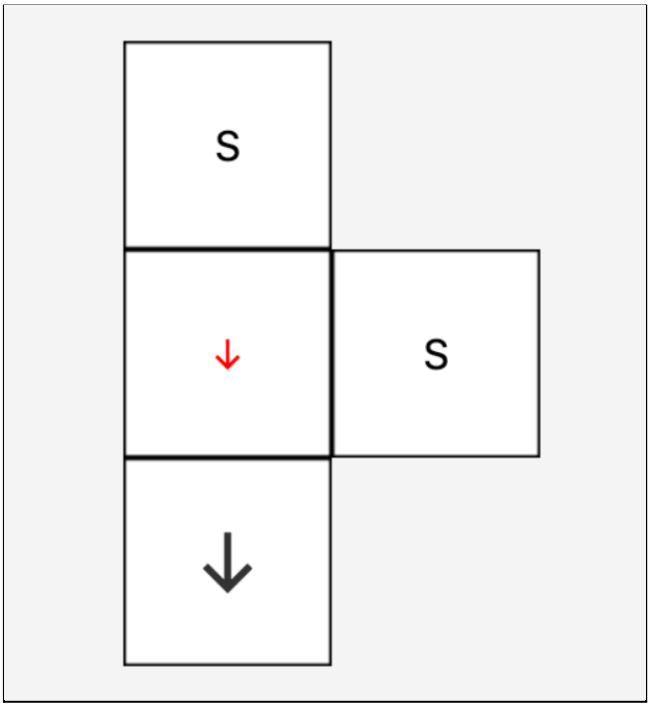
Path Evacuation Creator

Na początku generujemy wszystkie możliwe akcje do wykonania w pokoju. Moze to byc jedna z poniższych akcji:

- lewo (←)
- prawo (→)
- góra (↑)
- dol (↓)
- wyjście (E)
- pozostanie (S)

Każdy pokój ma dodana akcje pozostania, natomiast reszta określana jest dynamicznie na podstawie atrybutów takich jak, czy posiada wyjście, a także kierunek do sąsiadów. Wszystkie możliwe operacje przechowywane są w mapie. Odpowiednie etykietu ustawiane sa bazujac na wyniku otrzymanym z SAT Solvera.

Symulacja zagrożenia odbywa się poprzez kliknięcie na wybrany pokój. Wtedy wygląd aplikacji się zmienia. Zagrożenie jest pokazywane poprzez czerwony kolor. Pokoje które prowadzą do zagrożenia dostają literę “S”, co oznacza że osoba znajdująca się w nim powinna tam zostać i czekać na pomoc służb ratunkowych. Pozostałe etykiety też są odpowiednio uaktualniane na podstawie nowo wyliczonych akcji.



Istnieje możliwość resetu aplikacji do stanu początkowego, poprzez kliknięcie przycisku reset. Wtedy znaki wracają do domyślnych wartości.

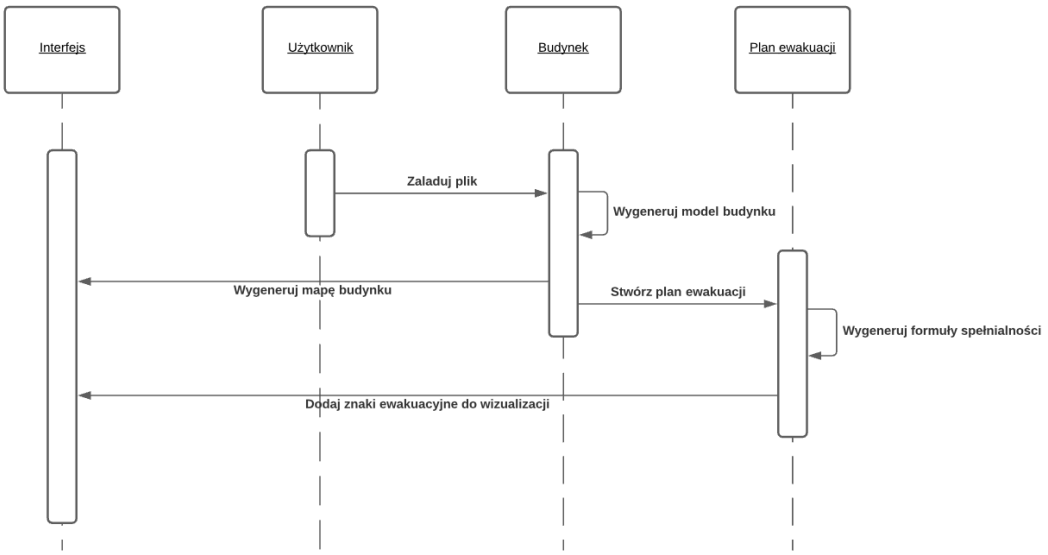


Diagram sekwencji tworzenia wizualizacji ewakuacji.

Import pliku IFC

Odczytanie planu budynku z pliku IFC odbywa się dwuetapowo:

1. Najpierw plik jest ładowany do grafowej bazy danych Neo4J
2. Następnie wykonywana jest seria kwerend do bazy danych, pozwalająca odczytać logiczną strukturę pomieszczeń w budynku

Import pliku IFC do Neo4J

Do wczytania pliku IFC i “przetłumaczenia” go do reprezentacji grafowej służy dedykowany skrypt w Pythonie, będący głęboką modyfikacją skryptu dostępnego w publicznym repozytorium https://github.com/Nobuho/IFC-Neo4j-converter. Logika działania skryptu jest następująca:

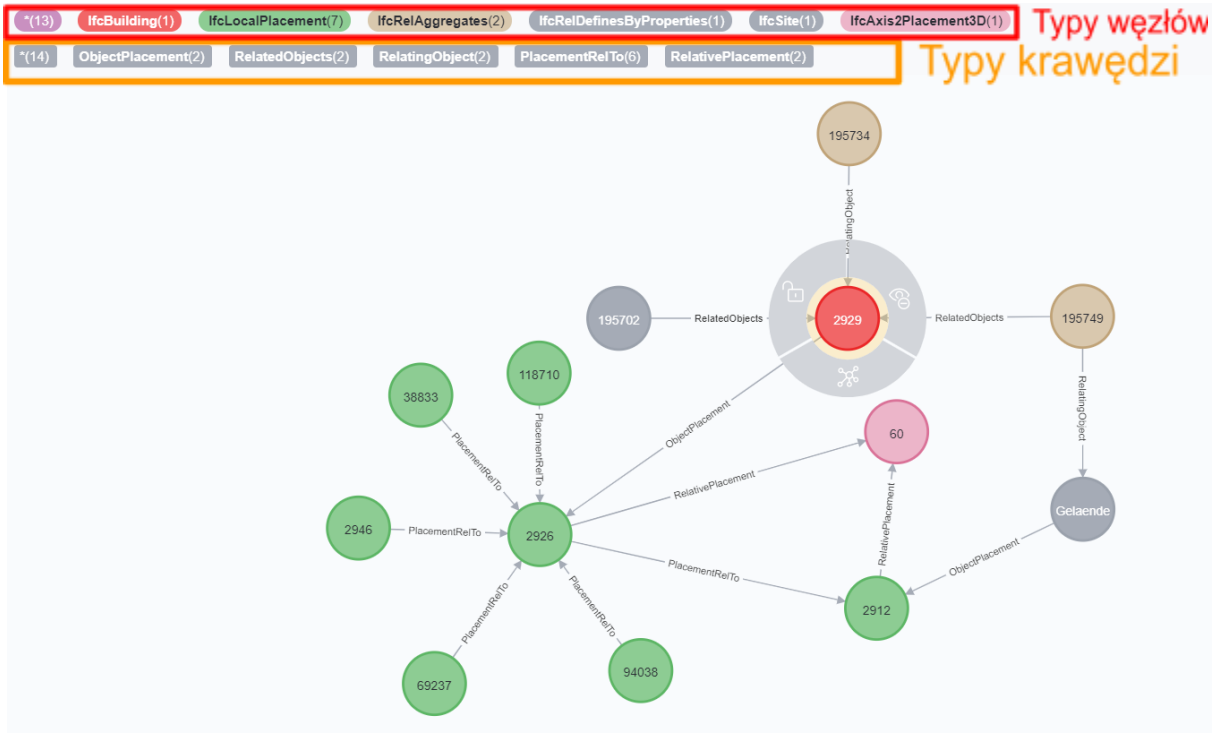
1. Cała zawartość pliku IFC jest wczytywana do pamięci, do struktury grafowej - dla każdego elementu pliku tworzony jest wierzchołek grafu, a powiązania między nimi zapisywane są jako krawędzie.
2. Na podstawie powyższej reprezentacji generowany jest zestaw plików .csv opisujących docelowy graf w formacie zrozumiałym dla Neo4J. Dla każdej klasy wierzchołka (każdego typu elementu pliku, np. IfcDoor, IfcSpace, IfcWall itd.) tworzony jest osobny plik - klasy te będą etykietami (“label”) wierzchołków w bazie danych. Oprócz tego tworzony jest pojedynczy plik opisujący krawędzie grafu.
3. Generowany jest plik wykonywalny (odpowiedni dla systemu operacyjnego - .bat lub .sh) zawierający kroki niezbędne do zaimportowania danych z utworzonych plików .csv do bazy danych (z wykorzystaniem narzędzia neo4j-admin).
4. Wygenerowany w punkcie 3 plik jest uruchamiany, dane są importowane do bazy danych.

Uwagi praktyczne:

- Baza danych, do której nastąpi import nie może zawierać żadnych danych
- Import można wykonać na wyłączonej bazie danych; jeżeli został on wykonany dla uruchomionej bazy, dane staną się dostępne dopiero po restarcie

Wynikowa struktura

Wszystkie encje pliku IFC dziedziczące po *IfcObjectDefinition* lub *IfcPropertyDefinition* są odwzorowane na węzły grafu - ich typ staje się etykietą, a wszystkie własności proste (nie będące odwołaniami do innych encji IFC) - własnościami węzła. Obiekty dziedziczące po *IfcRelationship* oraz własności węzłów będące odwołaniami do encji IFC odwzorowywane są jako krawędzie grafu. Wynikowa struktura:



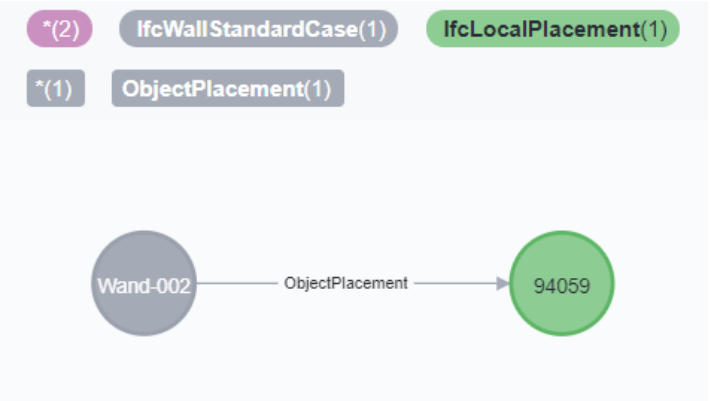
Odczyt struktury budynku ze struktury IFC zapisanej w bazie danych Neo4J

System koordynatów w IFC

Współrzędne wszystkich elementów budynku są w standardzie IFC opisane na jeden z dwóch sposobów:

- jako współrzędne **bezwzględne** (w skali projektu), za pomocą obiektu *IfcAbsolutePlacement*
- jako współrzędne **względne** (względem nadrzędnego elementu, np. pozycja pomieszczenia w ramach piętra), za pomocą obiektu *IfcLocalPlacement*

Element budynku jest powiązany z obiektem opisującym lokalizację za pomocą relacji *ObjectPlacement*:

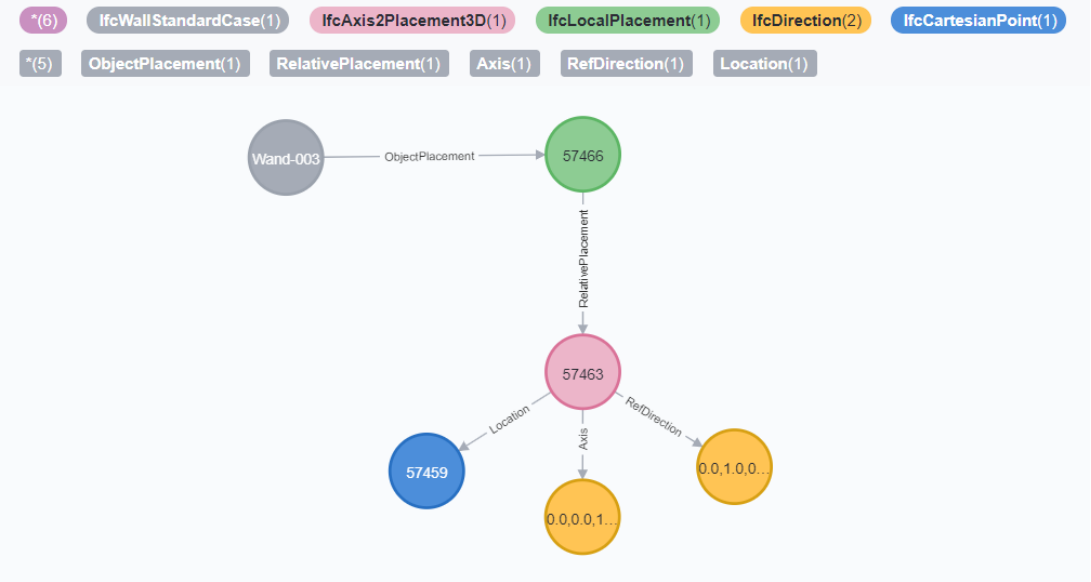


Szczególnie interesujący (i najbardziej popularny) jest drugi z opisanych sposobów, szerzej wyjaśniony niżej.

IfcLocalPlacement

Każdy obiekt *IfcLocalPlacement* jest powiązany z obiektem *IfcAxisPlacement3D* (relacją *RelativePlacement*), powiązanym z kolei z trzema obiektami:

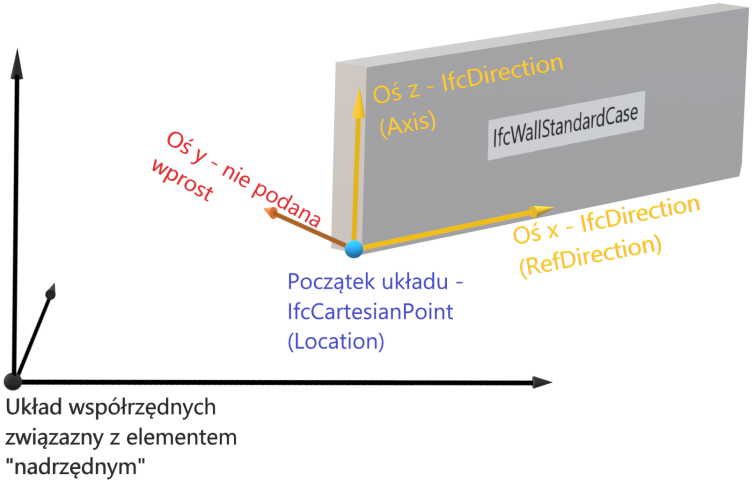
- *IfcCartesianPoint* (relacja *Location*)
- *IfcDirection* (relacja *Axis*)
- *IfcDirection* (relacja *RefDirection*)



Właściwości te mają za zadanie scharakteryzować lokalny układ współrzędnych związany z opisywanym elementem. Poszczególne obiekty opisują:

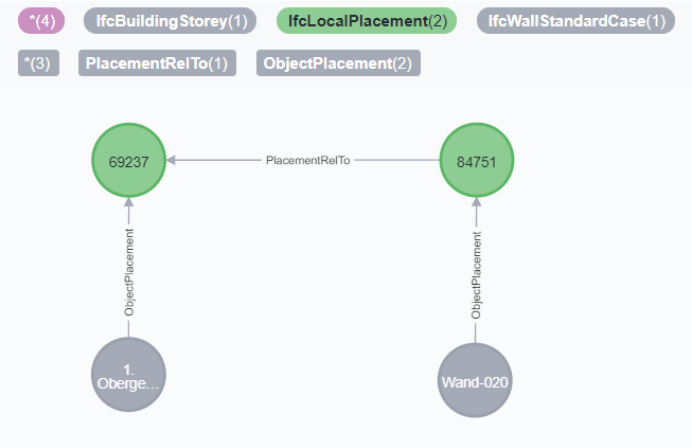
- *IfcCartesianPoint* (relacja *Location*) - początek układu współrzędnych
- *IfcDirection* (relacja *Axis*) - kierunek osi *z*
- *IfcDirection* (relacja *RefDirection*) - kierunek osi *x*

Przykład powyżej pokazuje opis układu współrzędnych związanego ze ścianą (obiekt *IfcWallStandardCase*), którego znaczenie przedstawia rysunek poniżej:

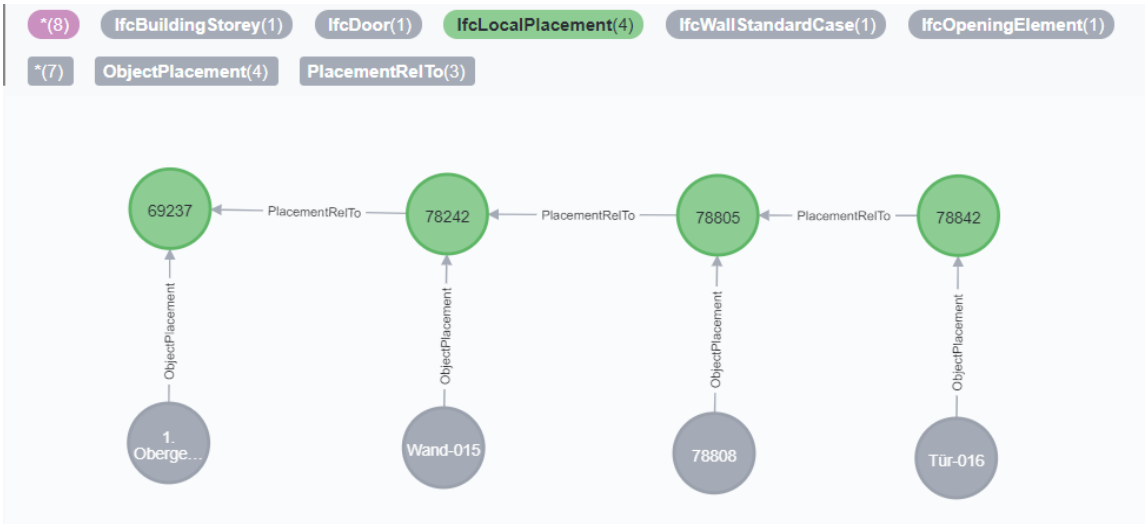


Obiekty IfcLocalPlacement związane z każdym z elementów agregowanych przez ścianę (drzwi, okna, itp.) wykorzystują współrzędne wyrażone w tak opisanym układzie współrzędnych - związanym ze ścianą. Co ważne, obiekt *IfcLocalPlacement* w żaden sposób **nie wskazuje położenia** skojarzonego elementu (ściany). Do opisanego nim układu współrzędnych odnoszą się jednak specyficzne obiekty opisujące kształt elementu (najczęściej *IfcProductDefinitionShape*), pozwalające np. wyznaczyć środek opisywanego elementu (przykłady znajdą się w kwerendach opisanych w dalszej części dokumentacji).

Przedstawiony rysunek zawiera “Układ odniesienia związany z elementem nadrzędnym”. Jest to układ odniesienia opisany przez obiekt *IfcLocalPlacement* związany z elementem agregującym rozpatrywany obiekt - np. w przypadku ściany elementem tym będzie piętro budynku (*IfcBuildingStorey*), a w przypadku drzwi - ściana. Ten “nadrzędny” obiekt *IfcLocalPlacement* jest powiązany z obiektem *IfcLocalPlacement* odpowiadającym ścianie za pomocą relacji *PlacementRelTo*:



Aby poznać współrzędne głębiej zagnieżdżonego obiektu, niezbędne jest odczytanie całego łańcucha tak powiązanych obiektów *IfcLocalPlacement*. Np. wyznaczenie pozycji drzwi (*IfcDoor*) w układzie współrzędnych związanym z piętrem budynku (*IfcBuildingStorey*) wymaga odczytania “po drodze” obiektów *IfcLocalPlacement* związanych z otworem na drzwi (*IfcOpeningElement*) i ścianą (*IfcWallStandardCase*):



Aby odczytać współrzędne globalne elementu, niezbędne jest znalezienie takiego łańcucha rozpoczynającego się od elementu *IfcSite*.

Odczyt współrzędnych

Poniższe omówienie dotyczy przypadku szczególnego, gdy osie Z we wszystkich lokalnych układach współrzędnych są równoległe. Takie założenie jest zazwyczaj spełnione dla interesujących z punktu widzenia systemu elementów - pomieszczeń, drzwi, ścian i pięter. Zastosowane podejście może być łatwo uogólnione (niezbędne byłoby uwzględnienie osi Z w przekształceniach oraz wyznaczaniu osi Y).

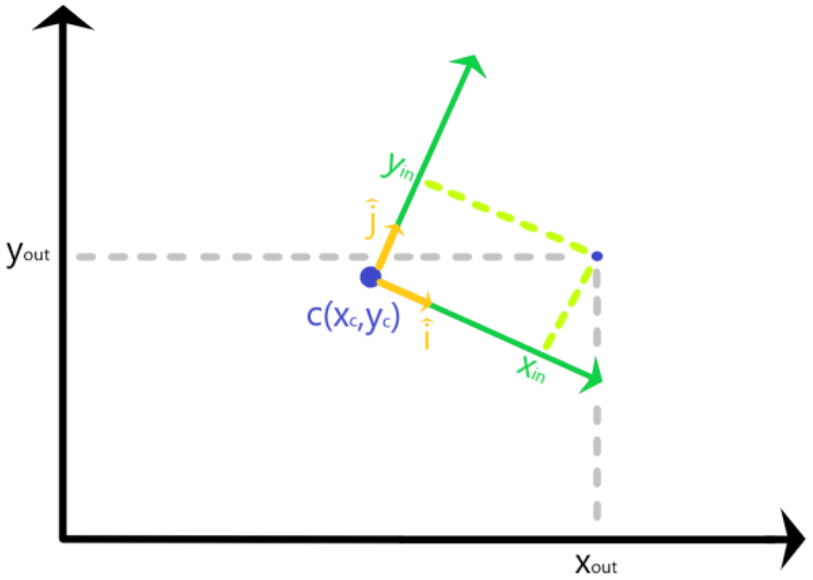
Każdy element *IfcLocalPlacement* możemy traktować jako opisujący **przekształcenie** lokalnych współrzędnych (x_{in}, y_{in}) do współrzędnych (x_{out}, y_{out}) w układzie współrzędnych związanym z elementem nadrzędnym. Dla układu opisanego wielkościami:

- $\hat{i} = [x_i, y_i]$- wektor jednostkowy wyznaczający oś X
- $\hat{j} = [x_j, y_j]$- wektor jednostkowy wyznaczający oś Y
- $\bar{c} = [x_c, y_c]$- wektor wyznaczający przesunięcie początku układu współrzędnych

przekształcenie to będzie miało postać:

$$x_{out} = x_c + x_i * x_{in} + y_i * y_{in}$$

$$y_{out} = y_c + x_j * x_{in} + y_j * y_{in}$$



Aby odczytać współrzędne związane z konkretnymi elementami (np. różnymi pomieszczeniami (*IfcSpace*)) w układzie współrzędnych związanym z którymś z elementów nadrzędnych (np. piętrem budynku (*IfcBuildingStorey*) lub globalnym (*IfcSite*)), musimy:

1. Znaleźć łańcuch obiektów *IfcLocalPlacement* łączący wybrany element nadrzędny z badanym
2. Wyznaczyć przekształcenie lokalnych współrzędnych w układzie odniesienia związanym z badanym elementem do współrzędnych w układzie związanym z wybranym elementem nadrzędnym, będący **złożeniem** przekształceń opisanych

przez obiekty *IfcLocalPlacement* w łańcuch znalezionym w punkcie 1, wykonanym od elementu nadrzędnego do badanego.

Złożenie przekształceń opisanych przez dwa obiekty *IfcLocalPlacement* powiązane relacją *PlacementRelTo* polega na wyrażeniu wektorów $\widehat{i_{in}}$, $\widehat{j_{in}}$ i $\overline{c_{in}}$ we współrzędnych właściwych dla układu w którym opisane są wektory $\widehat{i_{out}}$, $\widehat{j_{out}}$ i $\overline{c_{out}}$ (czyli w układzie *nadrzędnym* względem tego opisanego przez "bardziej zewnętrzny" z pary składanych obiektów *IfcLocalPlacement*). Złożenie ma postać:

$$\widehat{i} = [x_{i_{out}} * x_{i_{in}} + x_{j_{out}} * y_{i_{in}}, y_{i_{out}} * x_{i_{in}} + y_{j_{out}} * y_{i_{in}}]$$

$$\widehat{j} = [x_{i_{out}} * x_{j_{in}} + x_{j_{out}} * y_{j_{in}}, y_{i_{out}} * x_{j_{in}} + y_{j_{out}} * y_{j_{in}}]$$

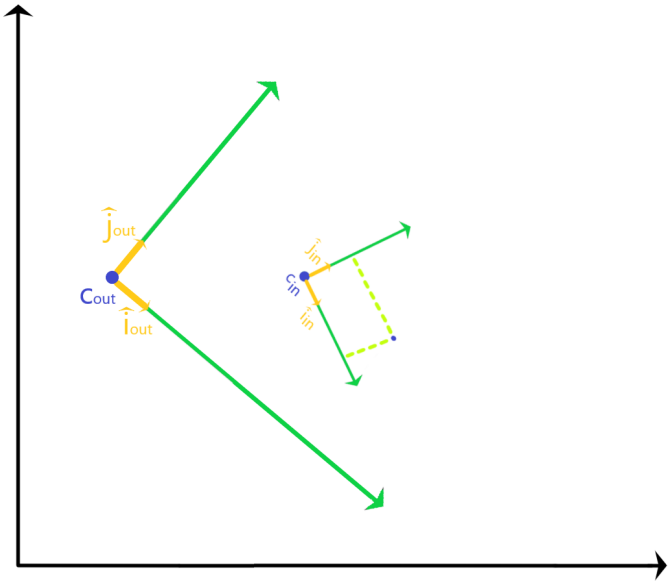
$$\overline{c} = [x_{c_{out}} + x_{i_{out}} + x_{c_{in}} + y_{i_{out}} * y_{c_{in}}, y_{c_{out}} + x_{j_{out}} * x_{c_{in}} + y_{j_{out}} * y_{c_{in}}]$$

Pierwsze dwa równania wynikają z mnożenia macierzy (przekształceń liniowych opisujących osie układów współrzędnych):

$$\begin{bmatrix} x_i & x_j \\ y_i & y_j \end{bmatrix} = \begin{bmatrix} x_{i_{out}} & x_{j_{out}} \\ y_{i_{out}} & y_{j_{out}} \end{bmatrix} \begin{bmatrix} x_{i_{in}} & x_{j_{in}} \\ y_{i_{in}} & y_{j_{in}} \end{bmatrix}$$

podczas gdy trzecie równanie to rozpisane wyznaczenie współrzędnych początku układu w nadrzędnym układzie odniesienia:

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_{c_{in}} \\ y_{c_{in}} \end{bmatrix} + \begin{bmatrix} x_{c_{out}} \\ y_{c_{out}} \end{bmatrix}$$



Oś *y* (wyznaczana przez wektor $\widehat{j}$) nie jest w elemencie *IfcLocalPlacement* podawana wprost - w każdym kroku musimy ją wyznaczyć na podstawie współrzędnych osi *x*.

Najłatwiej jest zrobić to poprzez obrót osi *x* (wektor $\widehat{i}$) o 90° przeciwnie do wskazówek zegara, np. z wykorzystaniem macierzy obrotu:

$$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

Kwerenda języka Cypher, wyznaczająca przekształcenie lokalnych współrzędnych do zadanego układu odniesienia, zbudowana w oparciu o opisane wyżej własności, wygląda następująco:

```
MATCH path = (:IfcBuildingStorey {Name:'1. Obergeschoss'})
-[:ObjectPlacement]->(:IfcLocalPlacement) <-[:PlacementRelTo * ..]-
(:IfcLocalPlacement) <-[:ObjectPlacement]- (obj:IfcDoor)
    WITH obj, nodes(path)[2..-1] AS p
    CALL {
        WITH p
        UNWIND p AS lp_coord
        MATCH (lp_coord) -[:RelativePlacement]-> (a:IfcAxis2Placement3D)
        WITH a
        MATCH (axes_node:IfcDirection) <-[:RefDirection]- (a)
-[:Location]-> (coord_node:IfcCartesianPoint)
        WITH
            split(axes_node.DirectionRatios, ',') AS axes,
            split(coord_node.Coordinates, ',') AS coords
        WITH
            toFloat(coords[0]) AS x,
            toFloat(coords[1]) AS y,
            toFloat(axes[0]) AS x_axis_x_coef,
            toFloat(axes[1]) AS x_axis_y_coef
        WITH x, y, x_axis_x_coef, x_axis_y_coef,
            -x_axis_y_coef AS y_axis_x_coef,
            x_axis_x_coef AS y_axis_y_coef
        WITH collect([x_axis_x_coef, x_axis_y_coef, y_axis_x_coef,
y_axis_y_coef,x,y]) AS coord_trans
        WITH reduce(outer_coefs = [1.0,0.0,0.0,1.0,0.0,0.0], inner_coefs
IN coord_trans | [
            outer_coefs[0] * inner_coefs[0] + outer_coefs[2] *
inner_coefs[1],
            outer_coefs[1] * inner_coefs[0] + outer_coefs[3] *
inner_coefs[1],
            outer_coefs[0] * inner_coefs[2] + outer_coefs[2] *
inner_coefs[3],
            outer_coefs[1] * inner_coefs[2] + outer_coefs[3] *
inner_coefs[3],
            outer_coefs[4] + inner_coefs[4] * outer_coefs[0] +
inner_coefs[5] * outer_coefs[2],
            outer_coefs[5] + inner_coefs[4] * outer_coefs[1] +
inner_coefs[5] * outer_coefs[3]
        ]) AS coord_system
        RETURN coord_system
    }
```

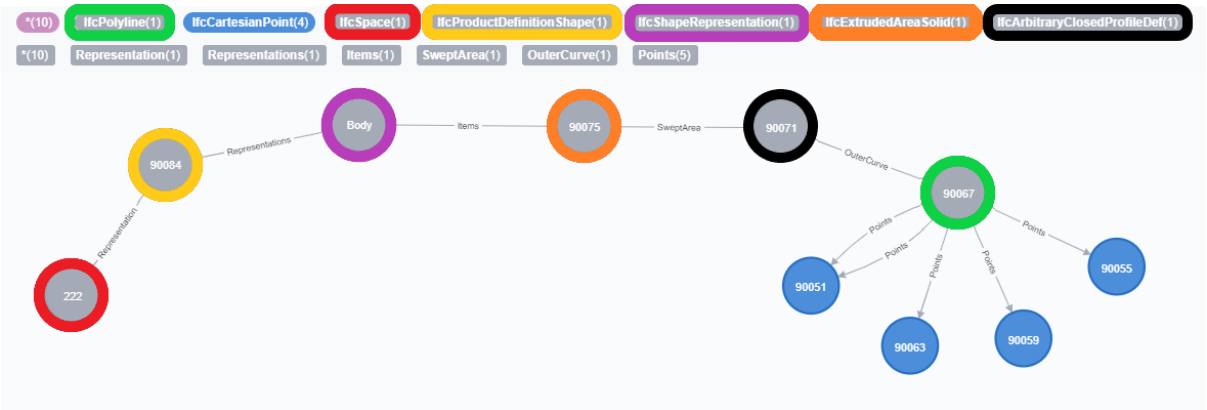
Wyszukanie łańcucha obiektów *IfcLocalPlacement*

Złożenie przekształceń

```
    RETURN obj, coord_system
```

Odczyt koordynatów wierzchołków pomieszczeń

Każde pomieszczenie (*IfcSpace*) ma ściśle określony kształt, zdefiniowany przez obiekt *IfcShapeRepresentation*. W rozpatrywanych przykładach kształt pomieszczenia jest opisany w dwóch wymiarach, za pomocą obiektu łamanej (*IfcPolyline*) wyznaczającej obrys pomieszczenia (trzeci wymiar - wysokość - wynika z kształtu elementu nadrzędnego, czyli piętra). Pełna ścieżka powiązań elementu *IfcSpace* z obiektami opisującymi jej kształt została przedstawiona poniżej:



Jeden z punktów (*IfcCartesianPoint*) jest połączony z obiektem łamanej (*IfcPolyline*) dwa razy, ponieważ jest to łamana zamknięta. Kwerenda znajdująca punkty dla pomieszczenia:

```
MATCH (space) -[:Representation]-> (:IfcProductDefinitionShape)
-[:Representations]-> (:IfcShapeRepresentation) -[:Items]->
(:IfcExtrudedAreaSolid) -[:SweptArea]-> (:IfcArbitraryClosedProfileDef)
-[:OuterCurve]-> (:IfcPolyline) -[:Points]-> (point:IfcCartesianPoint)
RETURN space, collect(DISTINCT point)
```

Współrzędne wszystkich punktów (*IfcCartesianPoint*) są zapisane w lokalnym układzie współrzędnych związanym z pomieszczeniem (*IfcSpace*) - aby odczytać ich wartość w sposób, w który będą one porównywalne z wierzchołkami innych pomieszczeń, musimy przenieść się do układu odniesienia wspólnego dla wielu pomieszczeń - np. związanego z piętrem. W tym celu łączymy powyższą kwerendę z opisaną wcześniej kwerendą znajdującą przekształcenie współrzędnych pomiędzy układami:

```
MATCH path = (:IfcBuildingStorey {Name:'1. Obergeschoss'})
-[:ObjectPlacement]->(:IfcLocalPlacement) <-[:PlacementRelTo * ..]-
(:IfcLocalPlacement) <-[:ObjectPlacement]- (space:IfcSpace)
    WITH space, nodes(path)[2..-1] AS p
    MATCH (space) -[:Representation]-> (:IfcProductDefinitionShape)
-[:Representations]-> (:IfcShapeRepresentation) -[:Items]->
(:IfcExtrudedAreaSolid) -[:SweptArea]-> (:IfcArbitraryClosedProfileDef)
-[:OuterCurve]-> (pl:IfcPolyline) -[:Points]-> (point:IfcCartesianPoint)
    WITH space, p, collect(DISTINCT point) AS points
    CALL {
```

```
        WITH p, points
        UNWIND p AS lp_coord
        MATCH (lp_coord) -[:RelativePlacement]-> (a:IfcAxis2Placement3D)
        WITH a, points
        MATCH (axes_node:IfcDirection) <-[:RefDirection]- (a)
-[:Location]-> (coord_node:IfcCartesianPoint)
        WITH points,
            split(axes_node.DirectionRatios, ',') AS axes,
            split(coord_node.Coordinates, ',') AS coords
        WITH points,
            toFloat(coords[0]) AS x,
            toFloat(coords[1]) AS y,
            toFloat(axes[0]) AS x_axis_x_coef,
            toFloat(axes[1]) AS x_axis_y_coef
        WITH points, x, y, x_axis_x_coef, x_axis_y_coef,
            -x_axis_y_coef AS y_axis_x_coef, // x_axis vector rotated by
90 degrees clockwise
            x_axis_x_coef AS y_axis_y_coef // x_axis vector rotated by
90 degrees clockwise
        WITH points, collect([x_axis_x_coef, x_axis_y_coef,
y_axis_x_coef, y_axis_y_coef,x,y]) AS coord_trans
        WITH points,
            reduce(outer_coefs = [1.0,0.0,0.0,1.0,0.0,0.0], inner_coefs
IN coord_trans | [
                outer_coefs[0] * inner_coefs[0] + outer_coefs[2] *
inner_coefs[1],
                outer_coefs[1] * inner_coefs[0] + outer_coefs[3] *
inner_coefs[1],
                outer_coefs[0] * inner_coefs[2] + outer_coefs[2] *
inner_coefs[3],
                outer_coefs[1] * inner_coefs[2] + outer_coefs[3] *
inner_coefs[3],
                outer_coefs[4] + inner_coefs[4] * outer_coefs[0] +
inner_coefs[5] * outer_coefs[1],
                outer_coefs[5] + inner_coefs[4] * outer_coefs[2] +
inner_coefs[5] * outer_coefs[3]
            ]) AS coord_system
        WITH [p IN points | split(p.Coordinates, ',')] AS points,
coord_system
        WITH [p IN points | [coord IN p | toFloat(coord)]] AS points,
coord_system
        RETURN [p IN points | [coord_system[4] + coord_system[0] * p[0]
+ coord_system[2] * p[1],
                               coord_system[5] + coord_system[1] * p[0] +
coord_system[3] * p[1]]] AS points_global
    }
```

```
    RETURN space.Name, space.GlobalId, points_global
```

Koordynaty drzwi

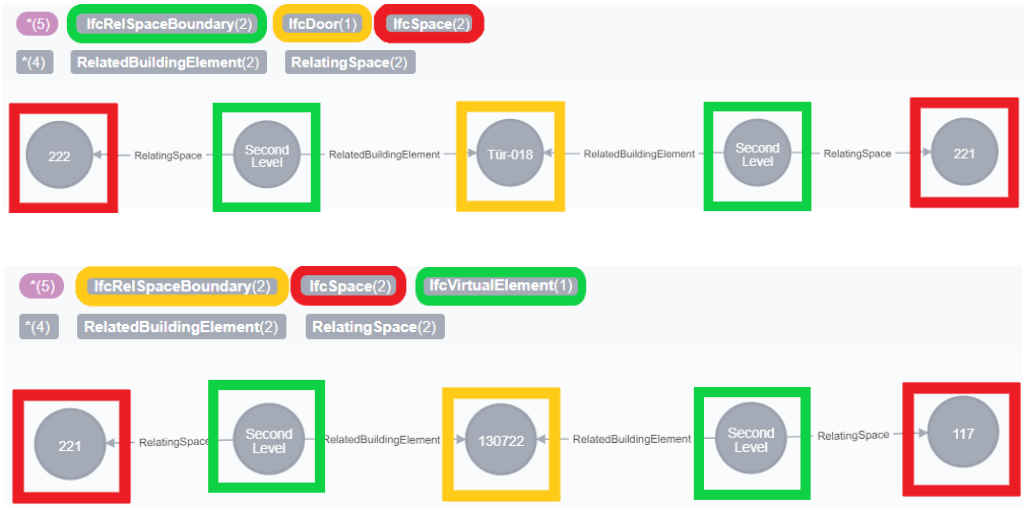
Wyszukanie koordynatów środków drzwi odbywa się analogicznie do wyszukania koordynatów wierzchołków pomieszczeń. Kształt drzwi w interesującym nas zakresie (tzn. przy ograniczeniu się do płaszczyzny XY) to odcinek, który - jak definiuje standard IFC - zawsze leży na osi X lokalnego układu współrzędnych. Stąd, znając lokalny układ współrzędnych drzwi (możliwy do wyznaczenia zaprezentowaną wcześniej kwerendą), możemy łatwo wyznaczyć współrzędne ich środka - wystarczy dodać połowę szerokości drzwi (opisanych przez parametr *OverallWidth* elementu *IfcDoor*) w kierunku lokalnej osi X. Takie postępowanie zostało wykorzystane w poniższej kwerendzie:

```
MATCH path = (:IfcBuildingStorey {Name:'1. Obergeschoss'})
-[:ObjectPlacement]->(:IfcLocalPlacement) <-[:PlacementRelTo * ..]-
(:IfcLocalPlacement) <-[:ObjectPlacement]- (obj:IfcDoor)
    WITH obj, nodes(path)[2..-1] AS p
    CALL {
        WITH p
        UNWIND p AS lp_coord
        MATCH (lp_coord) -[:RelativePlacement]-> (a:IfcAxis2Placement3D)
        WITH a
        MATCH (axes_node:IfcDirection) <-[:RefDirection]- (a)
-[:Location]-> (coord_node:IfcCartesianPoint)
        WITH
            split(axes_node.DirectionRatios, ',') AS axes,
            split(coord_node.Coordinates, ',') AS coords
        WITH
            toFloat(coords[0]) AS x,
            toFloat(coords[1]) AS y,
            toFloat(axes[0]) AS x_axis_x_coef,
            toFloat(axes[1]) AS x_axis_y_coef
        WITH x, y, x_axis_x_coef, x_axis_y_coef,
            -x_axis_y_coef AS y_axis_x_coef, // x_axis vector rotated by
90 degrees clockwise
            x_axis_x_coef AS y_axis_y_coef // x_axis vector rotated by
90 degrees clockwise
        WITH collect([x_axis_x_coef, x_axis_y_coef, y_axis_x_coef,
y_axis_y_coef,x,y]) AS coord_trans
        WITH reduce(outer_coefs = [1.0,0.0,0.0,1.0,0.0,0.0], inner_coefs
IN coord_trans | [
            outer_coefs[0] * inner_coefs[0] + outer_coefs[2] *
inner_coefs[1],
            outer_coefs[1] * inner_coefs[0] + outer_coefs[3] *
inner_coefs[1],
            outer_coefs[0] * inner_coefs[2] + outer_coefs[2] *
```

```
inner_coefs[3],
            outer_coefs[1] * inner_coefs[2] + outer_coefs[3] *
inner_coefs[3],
            outer_coefs[4] + inner_coefs[4] * outer_coefs[0] +
inner_coefs[5] * outer_coefs[2],
            outer_coefs[5] + inner_coefs[4] * outer_coefs[1] +
inner_coefs[5] * outer_coefs[3]
        ]) AS coord_system
        RETURN coord_system
    }
    WITH obj, coord_system, toFloat(obj.OverallWidth) * 0.5 AS w
    RETURN obj.Name, obj.GlobalId, coord_system[4] + coord_system[0] * w
AS x, coord_system[5] + coord_system[1] * w AS y
```

Połączenia pomiędzy pomieszczeniami

Pomieszczenia w budynku są reprezentowane przez elementy *IfcSpace*. Mogą one być połączone na dwa sposoby: przez drzwi lub poprzez “wirtualną granicę” (np. pomieszczenia na różnych piętrach połączone schodami). Połączenia obejmują obiekt pomieszczenia (*IfcSpace*), granicy przestrzeni (*IfcRelSpaceBoundry*) oraz drzwi (*IfcDoor*) lub zastępujący je “element wirtualny”:



Stąd kwerendy wyszukujące wszystkie pary połączonych pomieszczeń mają postać (odpowiednio - dla pomieszczeń połączonych przez drzwi i “granicę wirtualną”):

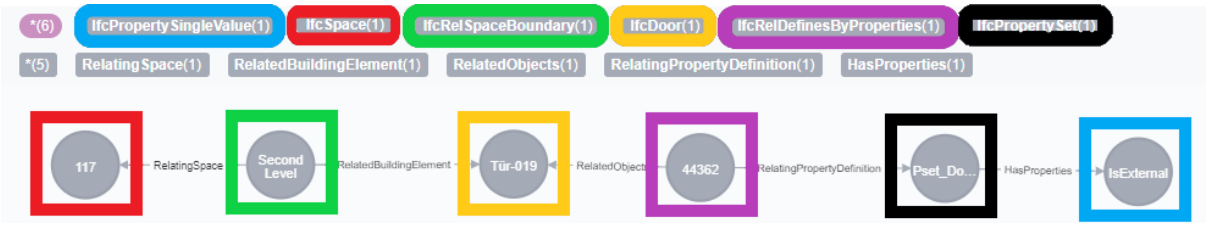
```
MATCH
(s1:IfcSpace)--(:IfcRelSpaceBoundary)--(d:IfcDoor)--(:IfcRelSpaceBoundar
y)--(s2:IfcSpace)
RETURN s1, d, s2
```

```
MATCH (s1:IfcSpace)--(:IfcRelSpaceBoundary
{PhysicalOrVirtualBoundary:'VIRTUAL'})--(:IfcVirtualElement)--(:IfcRelSp
aceBoundary)--(s2:IfcSpace)
```

```
RETURN s1, s2
```

Pomieszczenia z wyjściem

Na chwilę obecną jako pomieszczenia z wyjściem traktowane są tylko te pomieszczenia, które zawierają drzwi prowadzące na zewnątrz budynku. Drzwi takie można rozpoznać po właściwości *IsExternal* powiązanego z drzwiami obiektu *IfcPropertySet* o nazwie *Pset_DoorCommon:*



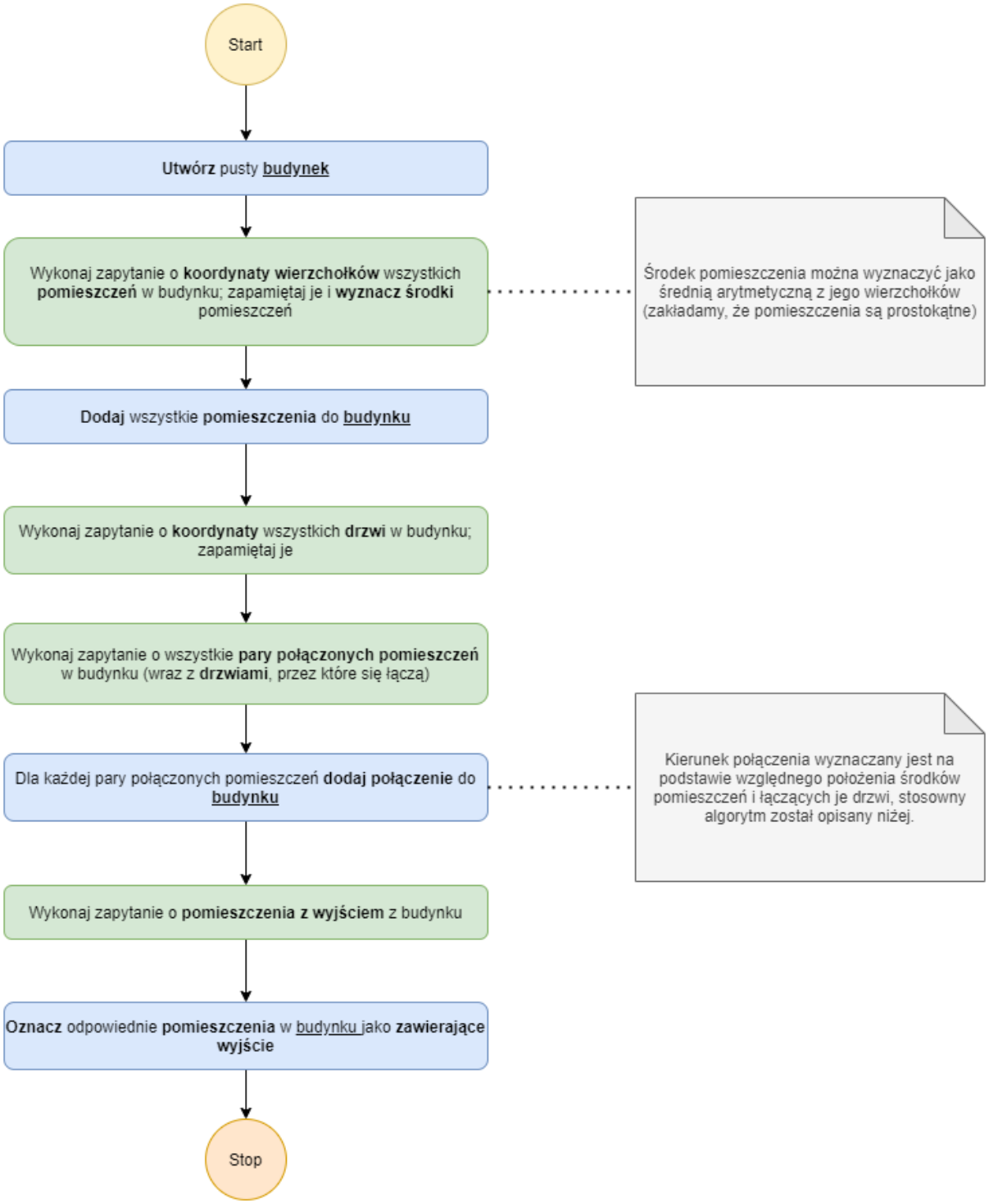
Kwerenda wyszukująca pomieszczenia z wyjściem:

```
MATCH
(s:IfcSpace)--(:IfcRelSpaceBoundary)--(:IfcDoor)--(:IfcRelDefinesByPrope
rties)--(:IfcPropertySet
{Name:'Pset_DoorCommon'})--(s:IfcPropertySingleValue {Name:'IsExternal',
NominalValue:'True'})
RETURN s
```

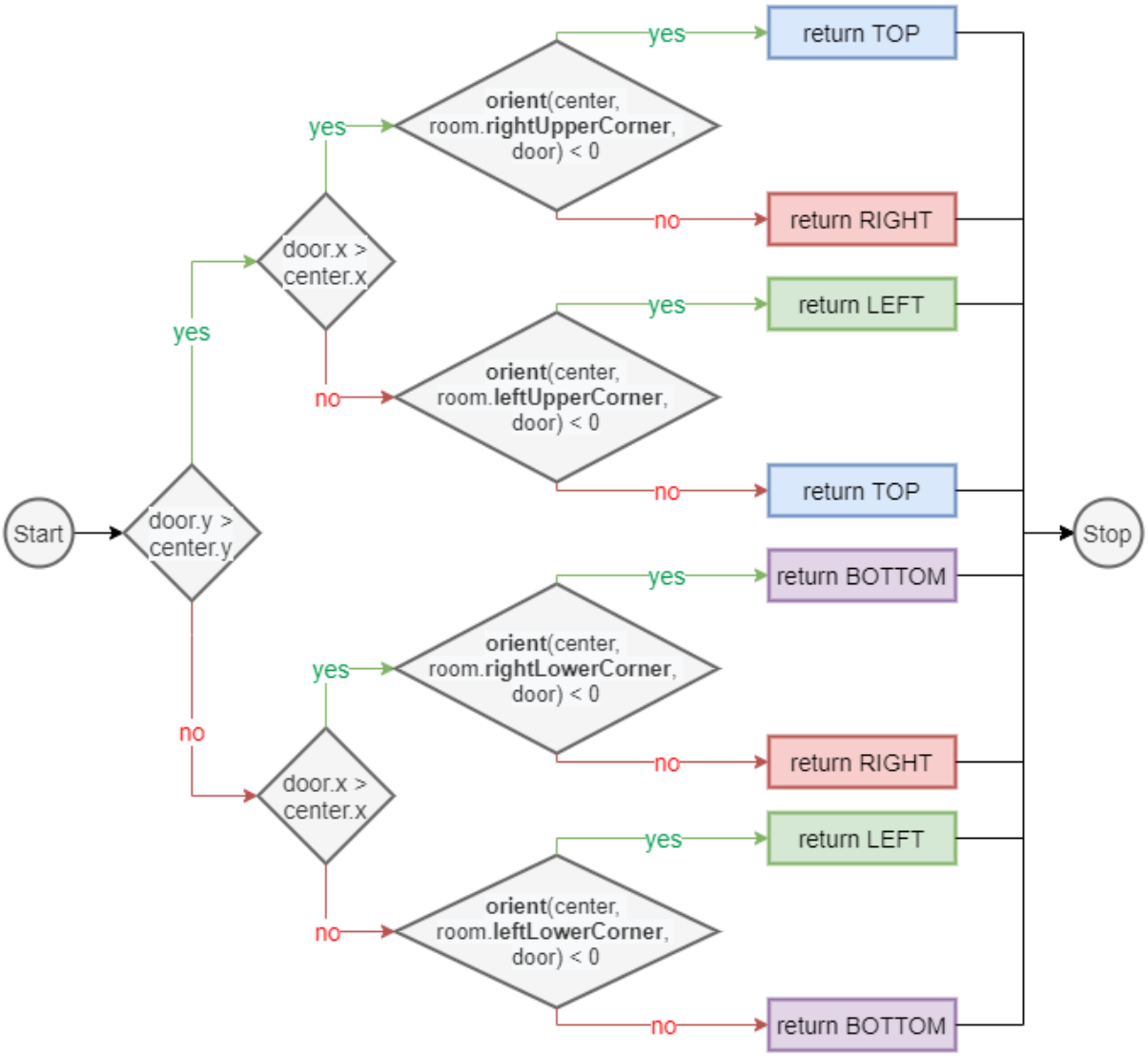
Dalsze prace: W sytuacji skrajnej opuszczenie budynku może być możliwe również innymi drogami (np. okna na parterze). Plik IFC zawiera na tyle bogate informacje o budynku, że można by pokusić się o wykrywanie tego typu niestandardowych dróg ewakuacji (a w wersji minimalistycznej - pozwolić użytkownikowi ręcznie je oznaczyć).

Odczyt budynku z bazy danych

Kroki, które trzeba podjąć, aby w oparciu o powyższe kwerendy odczytać z bazy strukturę budynku, zostały przedstawione na rysunku poniżej:



Algorytm określania kierunku połączenia pomiędzy pomieszczeniami (potrzebny w kroku 6) jest następujący:



Wykorzystana funkcja ***orient(A, B, C)*** określa wzajemne położenie trzech punktów. Zwraca ona:

- -1, jeżeli punkt C leży na lewo od półprostej AC
- 1, jeżeli punkt C leży na prawo od półprostej AB
- 0, jeżeli punkty A, B i C są współliniowe

Funkcja realizuje tę funkcjonalność, obliczając wyznacznik odpowiedniej macierzy (i zwracając jego znak):

$$f(A,\, B,\, C) \;=\; \operatorname{sign}\left(\det\left(\begin{bmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{bmatrix}\right)\right)$$