PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DANIEL CAMOZZATO

A METHOD FOR GROWTH-BASED PROCEDURAL FLOOR PLAN GENERATION

Porto Alegre
2015

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

A METHOD FOR GROWTH-BASED PROCEDURAL FLOOR PLAN GENERATION

Daniel Camozzato

Dissertação apresentada como requisito à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Profa. Dra. Soraia Raupp Musse

Porto Alegre
2015

Dados Internacionais de Catalogação na Publicação (CIP)

P192a Camozzato, Daniel
A method for growth-based procedural floor plan generation / Daniel Camozzato. – Porto Alegre, 2015.
76 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Profª. Drª. Soraia Raupp Musse.

1. Informática. 2. Modelagem Geométrica. 3. Algoritmos. 4. Computação gráfica. I. Musse, Soraia Raupp. II. Título.

CDD 006.60151

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada *"A Method For Growth-Based Procedural Floor Plan Generation"* apresentada por Daniel Camozzato como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 18/06/2015 pela Comissão Examinadora:

[illegible]

Profa. Dra. Soraia Raupp Musse– Orientadora — PPGCC/PUCRS

Isabel HManssour

Profa. Dra. Isabel Harb Manssour– — PPGCC/PUCRS

[illegible]

Prof. Dr. João Luiz Dihl Comba– — UFRGS

Homologada em 10/09/2015, conforme Ata No. 016 pela Comissão Coordenadora.

[illegible]

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320–3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

ACKNOWLEDGMENTS

Over the past two years, I have had the privilege of working with Prof. Dr. Soraia Raupp Musse as my advisor. This work would not have been possible without her guidance, support and patience. Her steadfast enthusiasm and drive are a fountain of inspiration, pushing those around her to do better.

Several others have helped me with this work. I am deeply grateful to Dr. Fernando Marson, who provided insight into procedural generation and whose good ideas have set me on the right path. I am also thankful for the insight and camaraderie of Dr. Leandro Dihl, with whom I shared fruitful discussions regarding this work. I also want to show my appreciation to Dr. Adriana Braun, who contributed expertise and helped me rethink several points of the model.

I would like to thank my fellow students, Amyr Borges Forte Neto, Cliceres Mack Dal Bianco, Rossana Queiroz and Vinícius Cassol for their support, feedback and friendship. They have been great labmates and helped make every day an enjoyable experience, sharing joys and woes in the trenches. I thank my friend Elaine Golendziner for helping me juggle many thoughts and for giving me subtle pushes in the right direction. I thank my best friend Tammie Summers not only for countless hours of proofreading, but also for just listening to me talk. Thank you for providing all the motivation. I must also thank my parents for providing me support, not only during this stretch, but throughout my life. Thank you for giving me the opportunity to thrive.

I would like to thank the Post-Graduate Program on Computer Science in PUCRS for providing me with the resources needed to complete this research, and for endeavoring to make my job easier. In special I would like to thank Régis Escobal da Silva and Diego Cintrão for their kindliness and patience, promptly answering my last minute questions. I would also like to thank the National Counsel of Technological and Scientific Development (CNPq) for funding my research.

Finally, I would like to express my gratitude in writing to all those who have provided instruction and guidance throughout my graduate career, in special professors Dr. Marcio Pinho, Dr. Fernando Gehm Moraes, and Dr. Leandro Möller, who taught me the value of starting with *baby steps*.

Thank you,

Daniel Camozzato

A METHOD FOR GROWTH-BASED PROCEDURAL FLOOR PLAN GENERATION

RESUMO

Neste trabalho apresenta-se um método procedural para criar plantas baixas levando em conta requisitos do usuário e também o limite das paredes externas de uma construção. Primeiro, uma grade é criada no espaço disponível. Então, cada aposento é posicionado de tal forma a ocupar uma célula da grade, e é subsequentemente expandido, ocupando células adjacentes para alcançar seu tamanho final. Essa abordadem baseada em crescimento pode gerar diferentes modelos de interior que atendem aos requisitos do usuário sem depender de passos custosos de otimização. O método proposto é capaz de lidar com uma variedade de formatos externos das paredes da construção, incluindo polígonos não convexos. Possíveis aplicações incluem ferramentas de arquitetura e a geração de conteúdo digital.

Palavras-chave: Algoritmos geométricos, modelagem procedural, geração de planta baixa.

A METHOD FOR GROWTH-BASED PROCEDURAL FLOOR PLAN GENERATION

ABSTRACT

We present a procedural method to create floor plans considering user-provided requisites as well as the constraint of a building's exterior walls. First, a grid is created in the available space. Then, each room is placed to occupy a single cell in the grid, and subsequently expanded, occupying adjacent cells to achieve its final size. This growth-based approach can generate different interior models which follow user requisites without relying on costly optimization steps. The proposed method handles a wide variety of building shapes, including non-convex polygons. Possible applications include architectural tools and digital content generation.

Keywords: Geometric algorithms, procedural modeling, floor plan generation.

LIST OF FIGURES

LIST OF TABLES

SUMMARY

1. INTRODUCTION

Computer graphics applications require models which are commonly created by hand using 3D modeling software. In architecture and engineering, this consists of creating 3D buildings from floor plans to provide a preview of the interior and exterior of unbuilt buildings. In video games, a similar task is performed for the creation of game levels. Such tasks are nontrivial, requiring skill and time.

Several solutions have been proposed in the literature to attenuate the modeling cost. For architectural applications, an automated system that converts 2D architectural floor plans to 3D computer graphics models can be used (see Yin et al. [31]). In video games, procedural algorithms can be used to generate content (see Hendrikx et al. [7]), including virtual environments such as cities (e.g. [8,19,28,29]), buildings (e.g. [5,16,17,32]) and furniture (e.g. [4,14]). Further, research has been conducted on the integration of semantic data with procedural generation techniques, to provide design tools and enable the use of procedurally generated models in simulations [27]. Other methods that use procedural modeling were also covered in a recent survey by Smelik et al. [25].

Several methods have also been presented to provide the interior 3D model of buildings, performing an automated generation of floor plans. One motivation to generate floor plans procedurally for predetermined building shapes is that procedural building generators often create hollow shells, without a 3D model for the interior. To create a complete building an interior must be generated [25]. A second motivation is the planning and optimization of real-world architectural layouts, a problem often tackled from the layout optimization standpoint (e.g., [2,15,24,30]).

Four main categories of procedural floor plan generation methods can be distinguished in recent research work. The first type takes the boundaries of a building and subdivides the internal space to create rooms. Examples for this type of method include the works by Noel and Siobhan [18], Hahn et al. [6], Rinde and Dahl [23], Marson and Musse [11] and Liu et al. [9]. The second type takes the inverse approach, creating the building from the inside out. Rooms are placed according to room connection constraints, without using a building's boundaries as a constraint. Rather, the building's exterior appears as a result of the room layout. This is exemplified in the methods presented by Martin [12] and Merrell et al. [13]. The third type of floor plan generation method distributes room seeds within the building's boundaries, and grows rooms iteratively to their final size, occupying the internal space. Examples of these growth-based methods are presented by Tutenel et al. [26] and Lopes et al. [10]. The final type is exemplified in the work by Peng et al. [20], which tessellates a domain into a quadrilateral mesh and completely covers the domain with tiles which in this case represent rooms.

Each approach has advantages and disadvantages. The subdivision approach is efficient and straightforward, but it does not provide ample control over room shapes and connections in the resulting domain partition. The resulting layouts either ignore room types and connections (e.g. [23]) or are only suitable for regular outline shapes (e.g. [9,11]). The second approach, which consists of generating the building from the inside out, can be used to generate realistic floor plans, but has limited use since it cannot be applied to buildings with predetermined exteriors. The tile placement approach presented by Peng et al. [20] can generate a floor plan for an arbitrary building exterior, covering the available space with tiles which represent desired room shapes, but it does not provide control over how and where tiles will be placed. Finally, growth-based algorithms avoid costly stochastic optimization steps, but need to solve problems which may arise from bad initial room placements. For example, two rooms which are meant to be adjacent may remain separate in the floor plan due to the interference of a third room's growth. Also, a room may fail to grow to an acceptable size if other rooms which surround it impede its growth.

In this work, we present a growth-based procedural floor plan generation method. The objective of this algorithm is to generate floor plans respecting both the external features of a building (walls and openings) and a user-provided set of requisites for the architectural plan (the number and types of rooms, and connections between rooms).

Our first contribution is the use of an irregular grid which is created considering the exterior walls and openings of the building. This results in room placement and growth that snaps to features of the building, creating more realistic floor plans than with an arbitrary regular grid. The second contribution is a method to generate floor plans which follows user requirements for room connections and shapes without the need for a deliberate room connection diagram, such as what is used in other methods (e.g. [11, 13]). This is achieved with an algorithm that makes it more likely for rooms to be placed close together when proximity is desirable, without enforcing adjacency. The third contribution is an alternative to the room expansion algorithm presented by Lopes et al. [10], which does not guarantee that rooms will expand to a minimum requisite size, nor provides control over the room's final shape. Our method guarantees that rooms will acquire a minimum length and width.

The remainder of this document is organized as follows: Chapter 2 presents and discusses the related work which serves as a base for this research. Chapter 3 presents the proposed procedural floor plan generation method. Chapter 4 discusses the results. Finally, Chapter 5 presents our final considerations.

2. RELATED WORK

For the purposes of this study, we focus our review on methods that generate the interior distribution of rooms for buildings, or *floor plans*. Several different techniques are presented and detailed in the following sections, followed by a taxonomy based on their characteristics.

2.1 Subdivision algorithms

A method for generating the interior floor plan for office buildings in real-time is presented by Hahn et al. [6]. Their research focuses on generating rooms and floors based on the user's position, and removing from memory rooms that are no longer visible. This avoids unnecessary use of memory and processing, allowing the procedural generation process to handle large buildings with many different rooms (e.g., skyscrapers).

The process also provides a persistent environment. A discarded room can be restored to its previous configuration by reusing the same random seed that was originally used in its creation, followed by reapplying all changes which may have been made by the user to its objects. All changes made in a given region are stored in a hash map when a region is removed from memory.

The process generates grid-like layouts (see Figure 2.1) through random splitting with axis-aligned planes. Eleven rules serve as a guideline to avoid inconsistencies in the regeneration system and to produce valid interiors (for example, one rule ensures that all rooms can be accessed from the entrance).

Buildings created by this technique are divided into rectangular regions connected by portals. Starting from the building's outline, the process first positions all elements that span across multiple floors, such as elevator shafts and staircases. The process then splits up the building into a number of floors. On each floor a hallway subdivision is applied, creating straight hallway segments or rectangular hallway loops. Finally, the interior spaces are subdivided into rooms, and appropriate objects are placed within.

Rinde and Dahl [23] present a method which performs a subdivision of a building's exterior outline. The input data are the the outer wall of the structure and the positions and sizes of any windows and doors. Other parameters are used to control the result of the algorithm in a variety of ways, ranging from corridor width to which types of regions and rooms the building should contain.

First, a building skeleton is generated from the input data to facilitate the analysis of the building's shape, as well as the construction of interior transition areas, such as corridors. This is achieved using a modified version of the straight skeleton algorithm [3], which produces a skeleton for a polygon. In this case, all exterior walls are pushed inwards at a constant rate and a skeleton edge is created where two walls meet. Doors are created from edges traced perpendicular from the skeleton (see Figure 2.2 (a)).

After the skeleton has been built the corridors within the building are placed. This is the

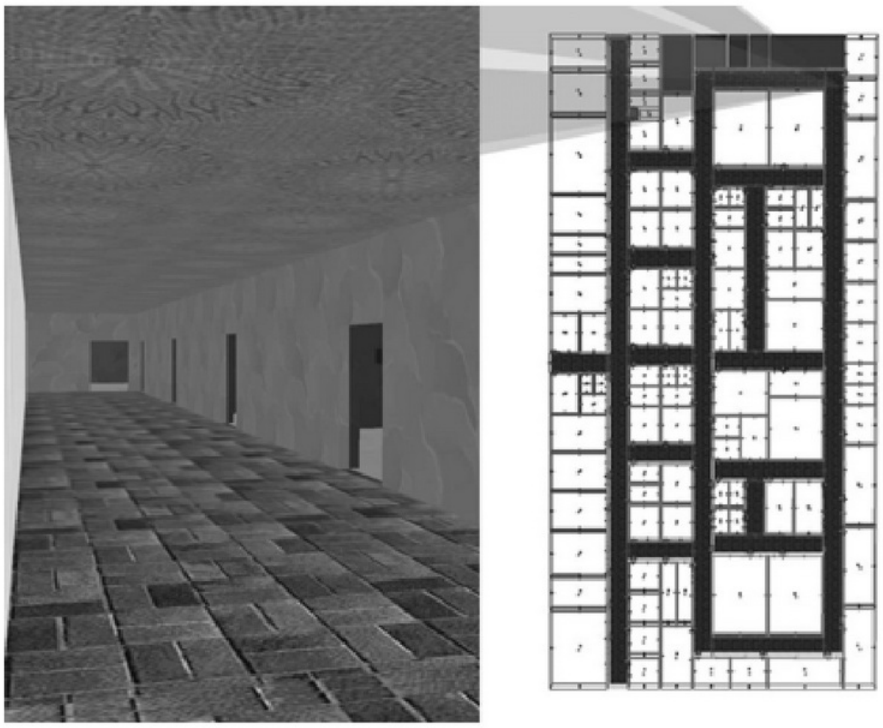

Figure 2.1: A screen shot showing a building's first person view and an overhead view of a completely generated floor. [6]

Transition Area used to gain access to all parts of the interior. Regions making up the maximum continuous areas are created and used for further sub-region division. The sub-regions represent the boundaries of a collection of rooms (e.g. apartments) (see Figure 2.2 (b)). The room walls are built using a hybrid weighted pseudo-Voronoi/S-Space algorithm (see Figure 2.2 (c)), where the cell boundaries of a Voronoi diagram are used to select suitable S-Space edges for walls (see Figure 2.2 (d)). Finally, rooms are built using the resulting room walls. During this process a graph of potential neighboring rooms is also constructed, which is then used to allocate room types and place doors between rooms. Figure 2.2 (e) shows a floor plan generated from exterior walls.

Marson and Musse [11] present a method that subdivides the available footprint using the squarified treemaps algorithm (see Bruls et al. [1]), such that the subdivision process attempts to maintain each room with an aspect ratio of one. The inputs for the algorithm are the dimensions of the building, the desired width and length for each room, and the type for each room, which defines the possible connections between rooms.

The first step in the process applies the squarified treemap algorithm to divide the building's area into three zones: the social, private, and service zones. The second step applies the squarified treemap again to each zone, creating rooms. Once the rooms have been created, the connections between rooms are placed according to a connection tree. The final step is to verify that all rooms can be accessed from the entrance. If a room cannot be accessed (see Figure 2.3), the process creates a corridor (see Figure 2.4).

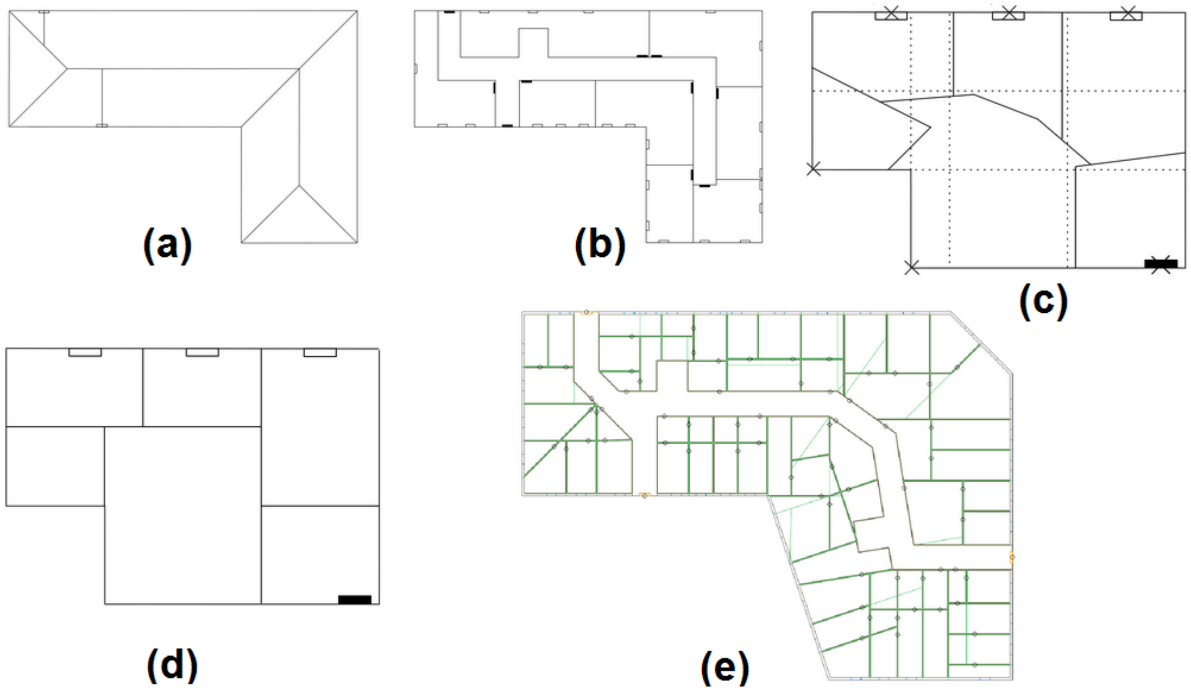

Figure 2.2: Straight skeleton with entrance doors appended (a), division into regions and further into sub-regions to create corridors and apartments (b), a pseudo-Voronoi/S-Space algorithm to select edges for walls (c), a finished apartment (d) and a floor plan generated using this technique (e). [23]

2.2 Tile placement algorithms

Peng et al. [20] present a method for tiling a domain with a set of deformable templates, such that the domain is completely covered and the templates do not overlap. The method breaks the layout algorithm into two steps: a discrete step to layout the approximate template positions and a continuous step to refine the template shapes. In the first stage, the domain and templates are tesselated into quadrilateral meshes (see Figure 2.5 (a), (b) and (c)). A gap-free tiling of the quadrangulated domain with graph-isomorphic copies of the quadrangulated templates is found by an integer programming approach (see Figure 2.5 (d)). Since the shape of a placed tile may deviate from the source template, the tiles are optimized by a continuous quadratic optimization until convergence or until a time limit is reached (see Figure 2.5 (e)).

The approach is suitable for a wide range of layout problems, including the generation of floor plans of large facilities such as offices, malls, and hospitals. During the integer tile placement, additional topological constraints are used to enforce corridor placement and room accessibility. The accessibility criteria is met by using corridor tiles that connect the room tiles to the building entrances or elevators. The aesthetic criteria is met by the admissible transformations of the room templates. The corridor templates are realized by a multi-level branching tree constraint (note the colored borders for corridor templates and red doors in Figure 2.6). Figure 2.6 shows two different office floor plan designs resulting from different predefined locations of the corridor root nodes.

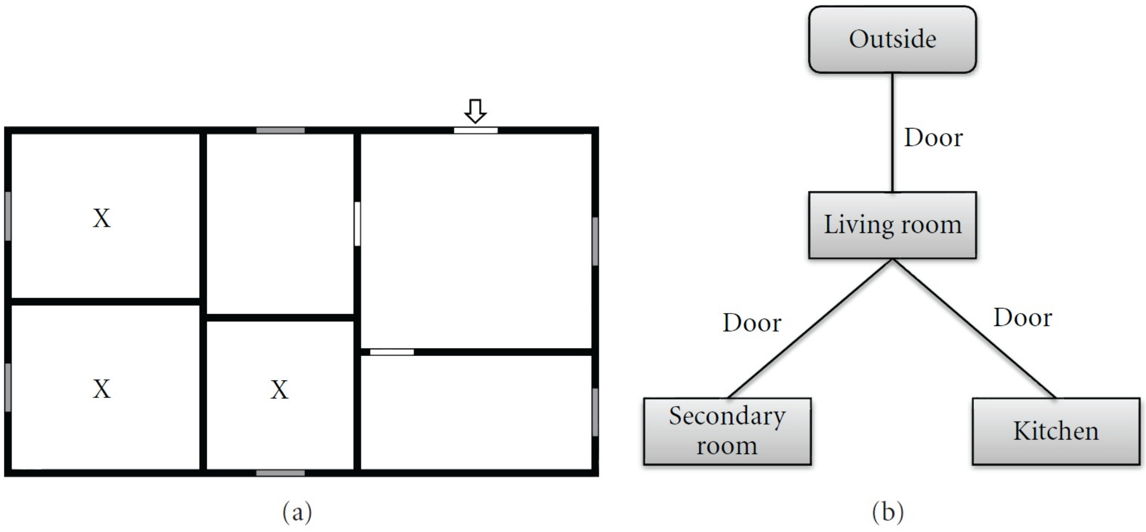

Figure 2.3: Example of floor plan (a) containing 3 nonconnected rooms (labeled with X) and its respective connectivity graph (b). [11]

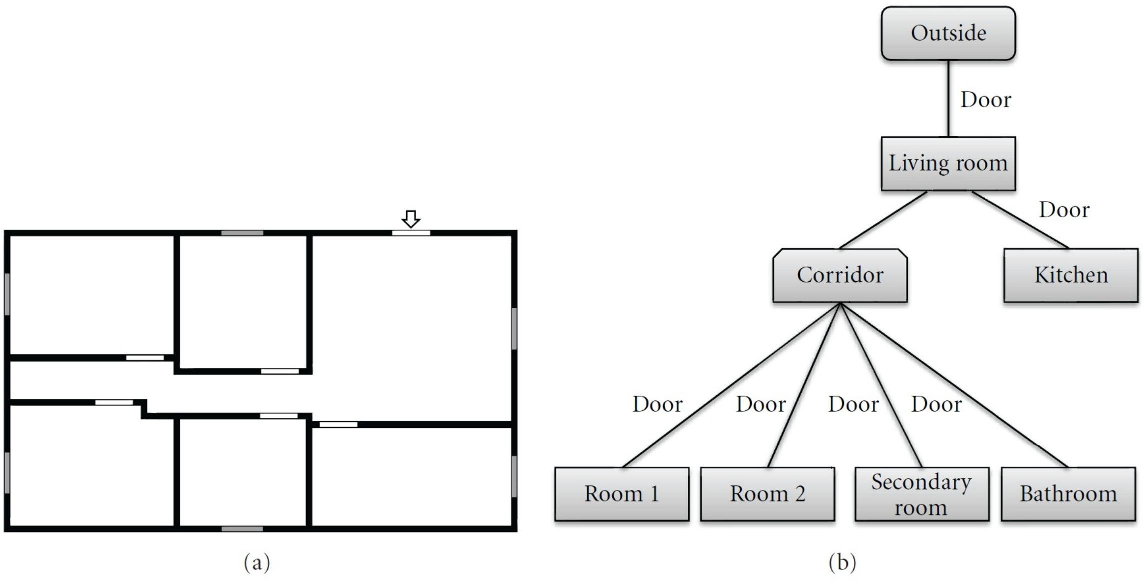

Figure 2.4: The floor plan seen in Figure 2.3 modified to include a corridor (a) and its respective connectivity graph (b). [11]

2.3 Inside out algorithms

Martin [12] presents an algorithm to generate floor plans for residential buildings (see Figure 2.7). The process consists of three main stages.

In the first stage, the basic structure of a house is represented by a graph in which each node corresponds to a room, and each edge corresponds to a connection between rooms. The graph itself is generated by using the front door as a root node, followed by adding all public rooms, and finally adding all private rooms. The specific room types are not initially assigned, but defined immediately after the public rooms have been distributed, according to a set of user-informed attributes.

At the beginning of the second stage, every room has been assigned a type and the connections

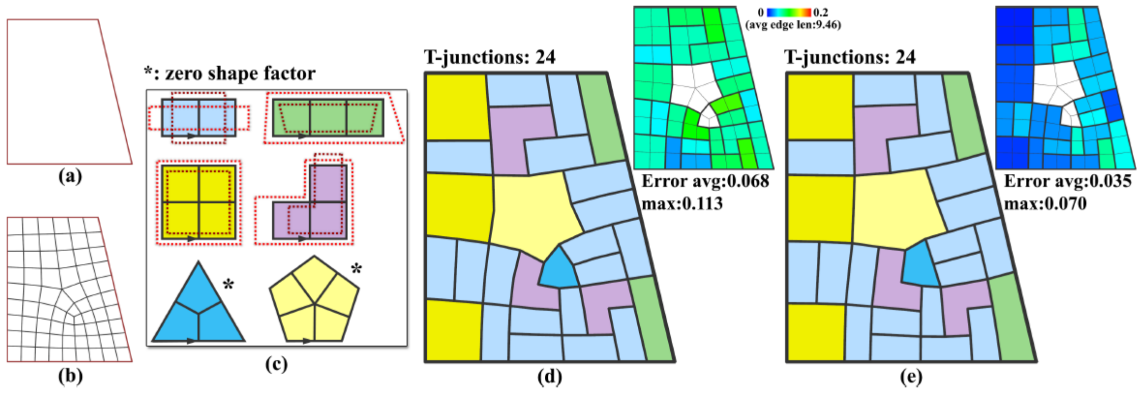

Figure 2.5: The problem domain (a) and its quadrangulation (b). The given tile templates (c). Each template is shown as its base polygon and its admissible transformations. A complete tiling generated by linear integer programming (d). The shape registration errors are visualized in the upper-right corners. The mesh geometry optimized to further reduce the shape registration errors (e). [20]

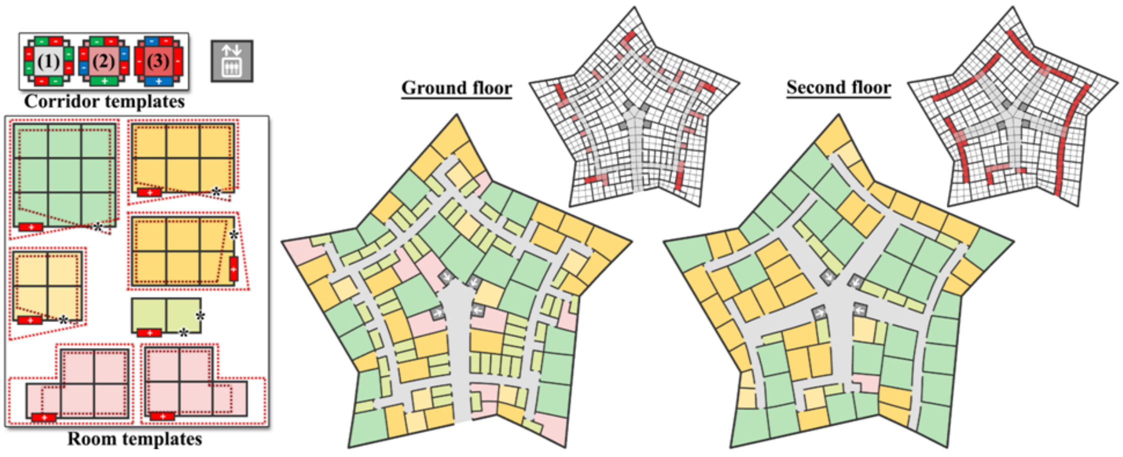

Figure 2.6: Two office floor plan designs. Left: corridor and room templates used during the tiling step. Note the topological constraints for corridor placement (the matching colored edges) and room placement (red rectangles mark door positions). Right: two different floor plans resulting from different predefined placements of corridor root nodes. [20]

between rooms have been determined, but the rooms are not yet located in space. The second stage generates the 2D position of each room from the graph generated in the first stage. To do this, the process treats the graph as a tree with its root at the room that adjoins the front door. From this root, each child node is evenly distributed adjacently, with equal distance from each other and from the root. The process is then repeated for each of the children nodes.

In the third stage, the rooms are expanded to their final size. Each room exerts an outward "pressure" that is proportional to the size the room needs to be. Thus, each room expands to occupy the remaining building space. If a wall is shared by two rooms, the algorithm considers the pressure inside and outside, to determine whether the room will expand.

Merrell et al. [13] present a method to generate realistic residential building floor plans from

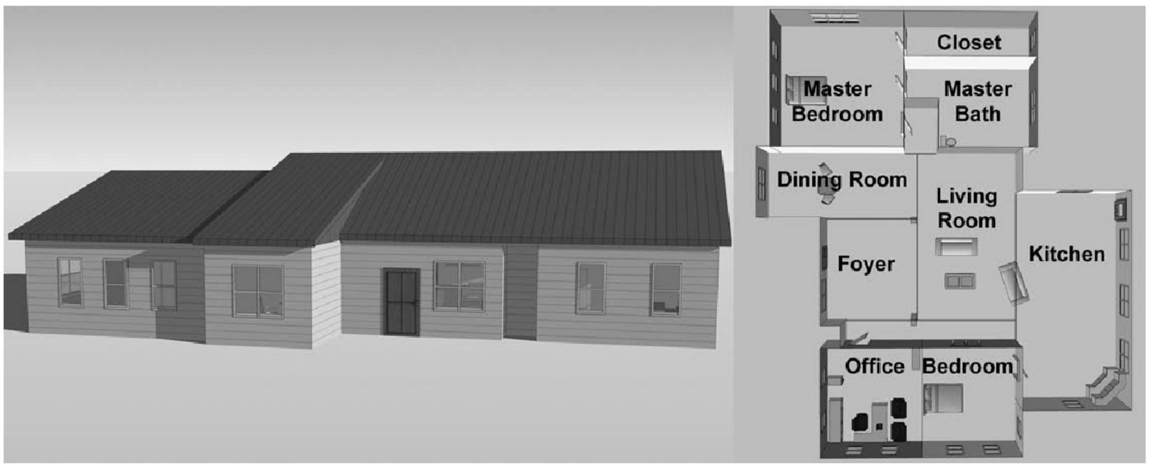

Figure 2.7: Left: Modeled house based on a floor plan generated by Martin's algorithm. Right: Top-down view of the same house's floor plan. [12]

high-level user specifications. They identify that the rules used by architects are too numerous and ill-specified to be modeled by a hand-designed rule-based system. So, instead, they apply machine learning techniques. A Bayesian network is trained with data from real-world buildings to infer aspects of architectural design which are subjective and difficult to codify. Such data includes, for example, which types of room are often adjacent, the room area and aspect ratio, and whether the adjacency is open or adjoined by a door.

In the first stage of the procedural generation process, a sparse set of high-level requirements is defined in a flexible manner. There are requirements such as the number of bedrooms, or the approximate square footage. Then, the Bayesian network is used to expand the requirements into a complete architectural program, describing room adjacencies and the desired area and aspect ratio for each room.

The second stage turns an architectural program into a detailed floor plan for each of the building's floors. This is achieved through stochastic optimization over the space of possible building layouts [22].

At each iteration of the algorithm, a new building layout is generated with local and global reconfigurations that significantly alter the overall layout. A reconfigurated layout is achieved through two types of moves: sliding walls and rooms swaps. The cost function that evaluates the quality of the proposed layout takes into account the accessibility, area, aspect ratio and shape for each room, as well as a term to penalize irregularity in the outline of each floor. Figure 2.8 presents the floor plan optimized for a given architectural program.

2.4 Growth-based algorithms

Tutenel et al. [26] propose an growth-based floor plan generation method. Each different type of room is defined as a class in a semantic library, and for each class several relationships can be defined. The relationships determine which room-to-room adjacencies are permitted, but can also

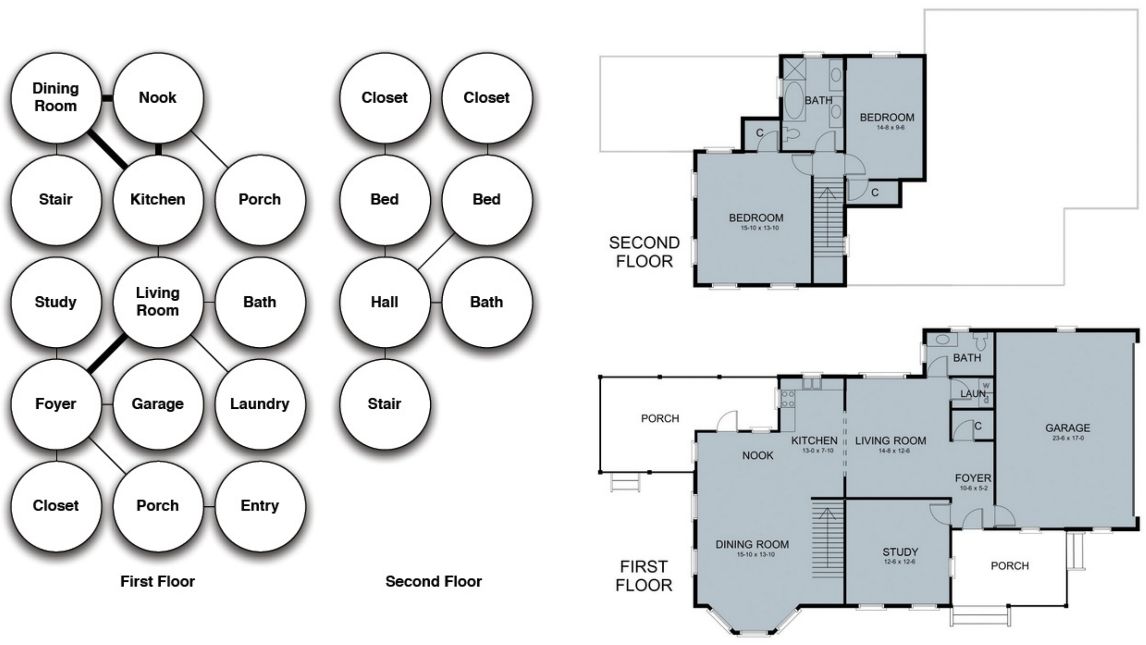

Figure 2.8: Computer-generated building layout. Left: An architectural program generated by a Bayesian network trained on real-world data. Right: a set of floor plans optimized for the architectural program. [13]

define additional constraints, such as requiring that a garage be placed facing the street.

To place the rooms, two steps are performed. First, rectangles called feature areas are placed at locations where all class relationships are met. Then, all rooms expand iteratively until they touch other rooms.

Figure 2.9 presents three floor plan layouts created by feature areas being placed differently inside a fixed-shape building. To place the feature areas, certain rules are followed. For example, in this case, the hall must connect to the street-side exterior wall (seen at the bottom of the image), and the living room must be placed adjacent to the kitchen.

This algorithm also uses the semantic library to determine relationships and constraints which serve as a guide to distribute objects (such as furniture) on the floor plan. All the possible placement positions are determined by considering the object's requirement, such as the clearance space required around an object, or the necessity of being placed against a wall.

Lopes et al. [10] note that previous techniques cannot handle irregular room shapes, for example, L-shaped or U-shaped rooms. They propose a grid-based expansion algorithm which grows rooms within a building layout.

The inputs to the process are the grid cell dimension in meters, a list of rooms to generate, adjacency and connectivity constraints, and for each room, its house zone (public, private or hallway) as well as its preferred area.

To generate a floor plan the following steps are followed. First, the algorithm divides the floor plan space by generating private and public zones.

Second, the room placement phase is performed. For this process, a separate grid of weights

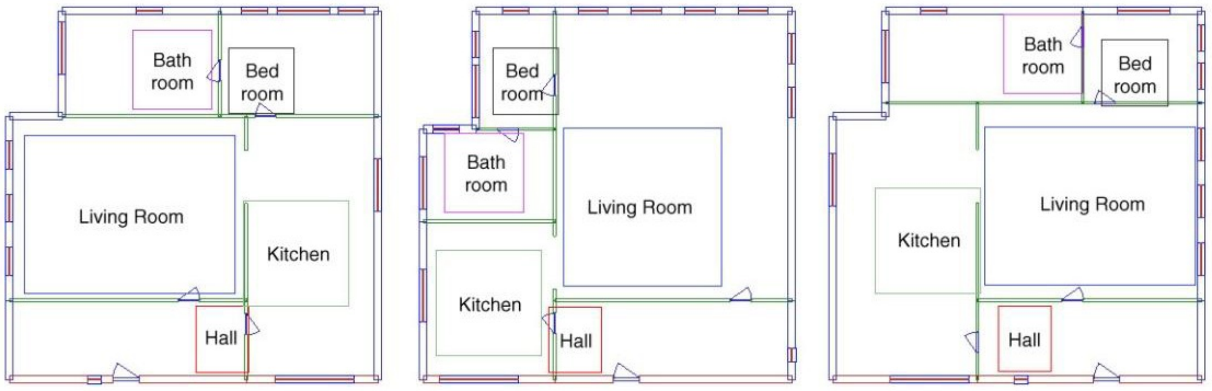

Figure 2.9: Three different floor plan layouts created by modifying the feature area positions. [26]

is created. Initially, each cell outside the building receives a weight of zero, and each cell inside receives a weight of one. The weights in each cell are then modified as needed, according to the restraints. For example, to guarantee that a given room connects to the outside of the building, the cells which do not connect outside receive a weight of zero. Based on these grid weights, one cell is selected to place a room's seed.

The third step expands the rooms. The algorithm picks one room at a time and attempts to grow it one step. Rooms which require a larger area are selected more frequently, expanding faster. When all rooms have been expanded to their maximum rectangular shape, the process attempts to expand the rooms in a L-shape. In a final step, the algorithm scans the grid for empty cells and assigns them to a nearby room, filling the gaps. Figure 2.10 shows an example of the output generated by the expansion process.

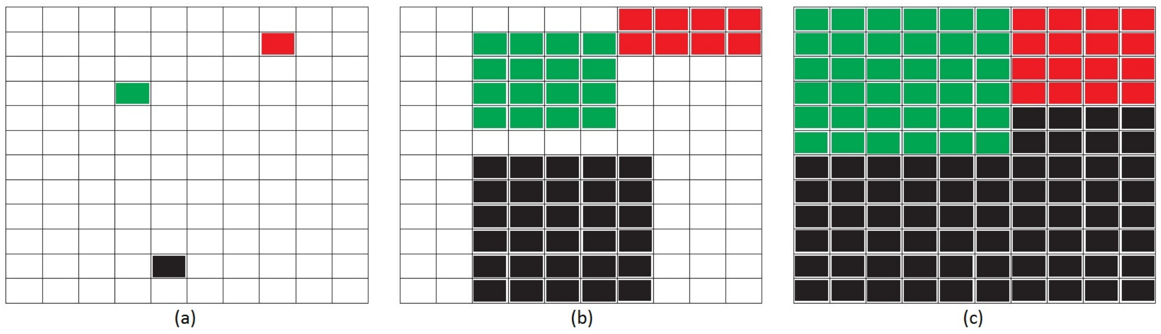

Figure 2.10: Example for the room expansion algorithm: (a) initial room positions, (b) rectangular growth, showing the step where the rooms have reached their maximum size, (c) L-shape growth. [10]

2.5 A taxonomy for procedural floor plan generation methods

The previous sections presented a review of different approaches that can be applied to procedural floor plan generation. We propose a taxonomy to classify these methods according to the following characteristics:

- *Type*: the type of procedural floor plan generation. The types are subdivision, tile placement, inside out and room growth. Subdivision algorithms take the boundaries of a building and subdivide the interior space to create rooms. Tile placement algorithms partition the domain and place tiles representing rooms upon the resulting grid. Inside out algorithms distribute rooms according to room connectivity requisites, without using a building's boundaries as a constraint. Growth-based algorithms distribute room seeds within a building's boundaries, and grow rooms iteratively to their final size, occupying the interior space.

- *Outline constraint*: determines whether a predetermined building exterior serves as a constraint, and if so, what kind of shape the building can take. Some methods do not accept a building exterior as a constraint, some accept rectangles, some accept non-convex axis-aligned polygons and some accept arbitrary polygons.

- *Window constraint*: determines whether the method uses windows as a constraint for room placement.

- *Room requirements*: this concerns whether user-provided requirements for a room's shape and placement in the floor plan are respected by the method.

- *Connection requirement*: determines whether user-provided connection requirements (for example, the need for two rooms to be adjacent) are respected by the method.

Table 2.1 provides an overview of the reviewed methods, classifying each according to the proposed taxonomy.

Table 2.1: Taxonomy for procedural floor plan generation methods

Method	Type	Outline const.	Window const.	Room req.	Connect. req.
Martin 2006	Inside out	No	No	Yes	No
Hahn 2006	Subdivision	Rectangle	No	No	No
Rinde 2008	Subdivision	Arbitrary	Yes	No	No
Tutenel 2009	Room growth	Axis-aligned	No	No	Yes
Marson 2010	Subdivision	Rectangle	No	Yes	No
Lopes 2010	Room growth	Axis-aligned	No	No	Yes
Merrell 2010	Inside out	No	No	Yes	Yes
Peng 2014	Tile plac.	Arbitrary	No	No	No
Our method	Room growth	Arbitrary	Yes	Yes	Yes

2.6 Comparison with state-of-the-art methods

In this section, we present a comparison of our algorithm with what we consider to be the state-of-the-art in each procedural floor plan generation approach (subdivision, growth-based, inside out and tile placement).

2.6.1 Subdivision

The method presented by Rinde and Dahl [23] has in common with our method that it also uses an initial interior partition to determine likely positions for walls. However, the process employs a subdivision approach rather than room growth. Their approach is flexible, generating floor plans for arbitrary shapes. However, the interior divisions are performed without regard for the required room sizes or a user-provided topology. Rather, room types are assigned at the end, by analyzing the connection graph obtained as a result of subdivision. In contrast, our algorithm creates floor plans by following user requirements from the start.

2.6.2 Growth-based

The method presented by Lopes et al. [10] is the most similar to ours, but there are several differences. First, their algorithm grows rooms using a regular grid with square cells. Thus, the room's growth follows the arbitrary length of the grid's cells. Our approach uses an irregular grid created according to the building's features. Thus, rooms grow according to the characteristics of the building, not arbitrarily.

Second, our algorithm uses both exterior walls and openings (windows and doors) as constraints, while the algorithm by Lopes et al. uses only exterior walls. This effectively means that their algorithm must place openings last, as a result of the procedurally generated floor plan, while our algorithm considers predetermined positions for openings and is capable of generating a floor plan for an existing building.

Finally, in Lopes et al., the only control provided for the room expansion process is that the higher a room's priority is set, the more it will grow. For example, a room with priority set to be twice as high as another will grow to be twice as big. Their algorithm does not provide any control over the final shape of the room, nor does it guarantee that a room will achieve a minimum size. In our approach, rooms are expanded at least to the the minimum requisite size, and further expanded to a target user-defined width and length. This prevents the creation of small, non-functional rooms.

2.6.3 Inside out

The method presented by Merrell et al. [13] generates realistic results by applying stochastic optimization to an architectural plan which is automatically generated by a Bayesian network which is trained with real-world buildings. These architectural plans consist of a tree structure, with nodes representing rooms and edges representing connections between rooms, along with other information such as the required room aspect ratios and sizes. However, their method does not use a predetermined building exterior as a constraint. It is not clear how rooms can be fitted inside an arbitrary shape while still keeping intact the room connections defined in the connection tree. Our approach avoids this problem by doing away with the connection tree and instead making it more likely for rooms to be placed close together when necessary, without enforcing adjacency.

A second difference is that our algorithm does not use an optimization process. Our approach does not intend to find the optimal floor plan, but rather a floor plan that respects the user-provided requirements and the building's exterior features at the same time. By avoiding costly optimization steps, our algorithm execution time remains low, while Merrell et al. report upwards from 30 seconds for the floor plan optimization time.

2.6.4 Tile placement

The method by Peng et al. [20] is capable of covering a mesh of quadrilaterals by using a set of tile templates. Their algorithm can be used for any layout problem requiring full coverage without gaps. Concerning floor plan generation, their approach is to use tiles which represent desired room shapes which must occur, and topology rules ensure room connectivity. The result is a large-scale distribution of tiles which resemble a building's floor plan.

The aesthetic component is preserved through the use of templates, which results in the desired shapes being placed within the domain. However, there is no control over where exactly a template will be added, nor how often it will be repeated. Thus, it cannot follow a specific set of user requirements for the number or placement of rooms. On the other hand, the algorithm is similar to ours in that no connection tree is used, and it could perhaps be modified (by adding more constraints) to provide more control over room quantity and placement.

3. MODEL

The proposed method for procedural floor plan generation uses a growth-based approach. First, a grid is created in the available floor plan space (see Figure 3.1(a)). Then, each room is placed to occupy a single cell in the grid (b), and subsequently expanded to occupy adjacent cells until the room achieves its final size (c). Edges of the grid become walls when the adjoining cells are occupied by different rooms.

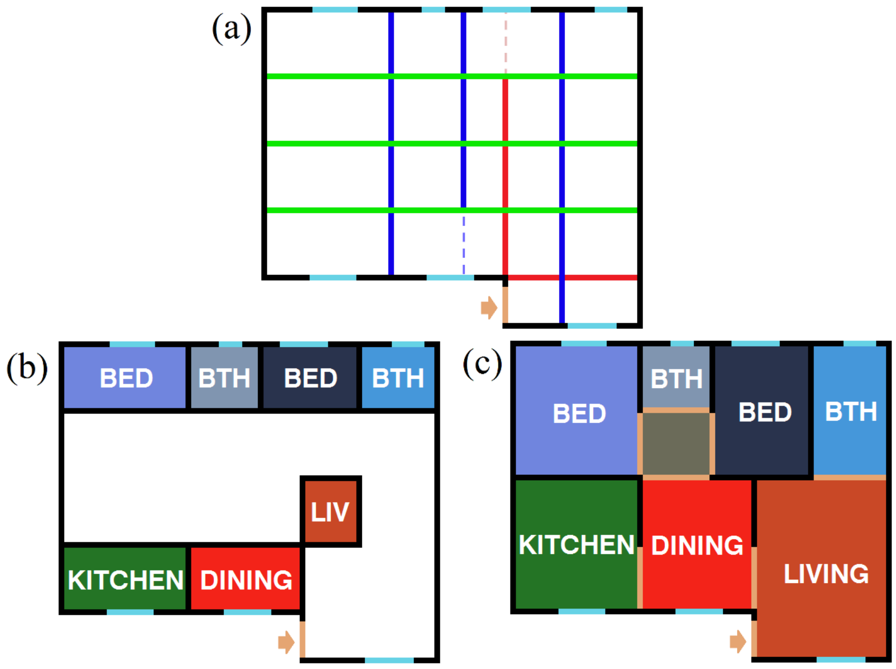

Figure 3.1: An overview of the proposed method. A grid is created in the available floor plan space according to the building's features (a). Then, each room is placed (b) and expanded to its final size (c). Edges of the grid become walls when the adjoining cells are occupied by different rooms. The exterior line segments are walls (in black), windows (in cyan) and a door (in brown, also marked with an arrow). Brown internal walls are wall on which a door may be placed during 3D extrusion.

For ease of reference, we have summarized all the variables used in this chapter in Table 3.1.

3.1 Input

The floor plan generation algorithm requires three inputs: a building's outline (its external walls and openings), user-provided room requisites R_R and connection requisites R_C. The algorithm places and expands one room for each R_R defined by the user, and each connection requisite R_C refers to two different R_R which must be kept adjacent. These requisites form an architectural plan describing the number and type of rooms that must exist in the floor plan, as well as any room adjacencies.

Table 3.1: Variables used in the method.

Variable	Description
K_o	Total score of the expansion o regarding R_R
kb_o	Blocked direction rule's partial score for o regarding R_R
kn_o	Necessity rule's partial score for o regarding R_R
kw_o	Window waste rule's partial score for y regarding R_R
o	An expansion option (either up, down, left or right)
P_o	Probability for an expansion option o to be selected
P_y	Probability for an eligible cell y to be selected
R_C	Connection requisite
R_R	Room requisite
S_y	Total score of the eligible cell y regarding R_R
sd_y	Entrance door rule's partial score for y regarding R_R
se_y	Elastic connection rule's partial score for y regarding R_R
sh_y	Hard connection rule's partial score for y regarding R_R
sk_y	Small window rule's partial score for y regarding R_R
sp_y	Private rule's partial score for y regarding R_R
ss_y	Social rule's partial score for y regarding R_R
sw_y	Window rule's partial score for y regarding R_R
t_D	Distance threshold (for eligible cells)
t_W	Wall length threshold
y	An eligible cell where a room can be placed

Room requisites have the following attributes: type (either social, private or service), minimum/maximum width, length and area, window necessity, and a priority to determine the order in which rooms are placed and expanded. Unlike other algorithms (e.g., Marson and Musse [11] or Merrell et al. [13]), our algorithm does not use a explicit connection tree to predefine the adjacencies of rooms in the floor plan. Rather, during room placement (see Section 3.3), rooms may (or may not) be more likely to be placed close together. Only rooms defined in connection requisites R_C are guaranteed to be adjacent. For example, a connection requisite may be used to keep a kitchen and a dining room adjacent in the floor plan.

Walls and openings are received as a list of typed line segments forming a non-convex polygon. These typed line segments are used by the grid generation process (see Section 3.2) to create the irregular grid used for room placement and expansion.

The objective of the presented algorithm is to create a floor plan within the boundaries of the building's exterior walls while expanding rooms to at least their minimum size and keeping rooms defined in requisites R_C adjacent. If the algorithm is successful, it manages to fulfill all R_R and all R_C and the resulting floor plan is considered valid.

3.2 Grid generation

To create a grid in the floor plan space we use the partition proposed by Peponis et al. [21] which is obtained by extending all exterior lines meeting at reflex angles (see Figure 3.1(a), in red). The extended lines form an irregular axis-aligned grid, created according to the building's features. Internal walls are often guided by features of the external walls rather than protruding randomly within a floor plan. Thus, the edges of this grid are natural positions for internal walls.

Note that in our approach diagonal elements do not extend lines and the grid remains axis-aligned even if diagonal walls are present in the input. Elements such as diagonal walls and openings otherwise exist normally in the cells in which they occur. Further, a line cannot be extended into an opening because the lines in the grid may become walls, and a wall cannot be built into a door or window. If such a line is created, it instead extends up to the last line before the opening (see the dashed lines in Figure 3.1(a)).

Windows are resources in a floor plan and may need to be assigned to different rooms. Thus, to keep windows in different cells of the grid, lines are additionally extended perpendicular to the midpoint of each line segment that separates two openings (see Figure 3.1(a), in blue).

Depending on the exterior features of the building, the above partition rules may not be sufficient to create an acceptable grid for the proposed algorithm. For example, for a rectangular building where all windows lie on one side, the resulting grid is a set of long parallel slices. Thus, we additionally extend a line perpendicular to the midpoint of any wall segment which is longer than a threshold t_W (see Figure 3.1(a), in green). This is done recursively until no wall segment is longer than t_W.

Increasing t_W allows for longer walls to remain without subdivision, which results in a coarser grid. A coarser grid decreases the accuracy of the room expansion process, but favors snapping the internal walls to lines extended from exterior features of the building, which is preferable for internal walls. Conversely, decreasing t_W creates a finer grid which allows greater control for the room expansion process, but decreases the chance of snapping to lines extended from exterior features of the building. Thus, t_W is a variable which must be tuned to balance competing needs (fine control of the expansion vs. snapping to more meaningful grid lines). In our results, we empirically determined $t_W = 1.8m$ to be a good balance between these needs.

Note that the grid created by this step is irregular, presenting edges according to the buildings' features, rather than at arbitrary steps along the X and Y axis.

3.3 Room placement

The room placement step creates a room occupying a single cell for each user-provided room requisite R_R. These single-cell rooms are expanded to their final size in a later step (see Section 3.4).

3.3.1 Determining eligible cells

To place rooms in the grid, we first create a list of grid cells which are considered eligible to receive rooms. The purpose is to limit the options for room placement and ensure that rooms will be placed with sufficient distance from one another to be allowed to grow.

In a first step, we select every cell containing a door or a window. We consider these prime locations for rooms because it is advantageous to place rooms in cells containing features of the building. However, since this step only selects cells containing features of the exterior wall, additional cells must be made eligible to populate the inside of the building.

We use a threshold distance t_D to select the first cells which are at least t_D meters distant from the previously selected cells. We repeat this process iterativelly to cover the entire grid evenly, until all cells are within approximately t_D meters from an eligible cell.

Figure 3.2 presents an example of eligible cells selected for different values of t_D. Lower thresholds result in a larger number of eligible cells, allowing more rooms to be placed. However, each room is also given less space to expand, and prime positions (in pink) lose importance, becoming less likely to be chosen during room placement. We empirically determined $t_D = 2.5m$ to generate good patterns of eligible cells, providing enough eligible cells for rooms to be placed and enough room for rooms to expand.

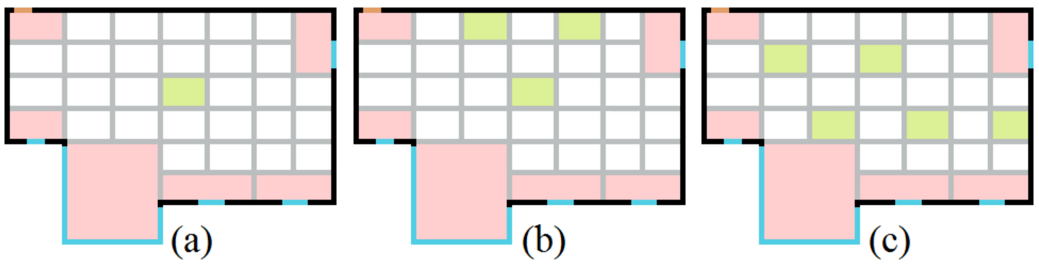

Figure 3.2: Cells containing features such as wall's corners, windows and doors are considered eligible to receive rooms (in pink). Additional eligible cells (in green) are made available by selecting cells t_D meters away from all other eligible cells. This is done iteratively to cover the entire grid. Note how the number of eligible cells increases as t_D is reduced (from a to c, $t_D = 3$, $t_D = 2.5$ and $t_D = 2$ meters, respectively).

Using a different threshold t_D modifies the number and distribution of eligible cells, leading to different room placement and expansion possibilities. A specific t_D values may be more suitable to a given building. However, the results of such adjustments are non-trivial to predict.

3.3.2 Scoring eligible cells

When a room is to be placed, each unoccupied eligible cell y receives a score S according to how well the eligible cell fulfills the needs defined in the room requisite R_R. The score for an eligible cell y is:

$$S_y = sw_y \; ss_y \; sp_y \; se_y \; sh_y \; sd_y \; sk_y, \tag{3.1}$$

where sw_y, ss_y, sp_y, se_y, sh_y, sd_y and sk_y are factors weighed between 0 and 1 referring to scores associated with different rules. Note that S_y also lies within this range. Each score S_y is calculated for a specific room requisite R_R. Therefore, after each room is placed in the grid, all scores are reset and must be calculated again.

Some rules only apply under certain conditions; for example, sp_y is valued by the private rule which only affects room requisites R_R with the private type. Whenever a rule does not apply, its associated score is considered to be 1.

Window rule

The purpose of the window rule is to make it more likely for rooms to be placed in eligible cells that contain a window. Any eligible cell without a window has its score halved. Further, if the room requisite R_R determines that a window is absolutely required, then any eligible cell without a window has its score set to zero. For example, each bedroom explicitly demands a window, so eligible cells without windows have their score set to zero when the bedroom is placed.

The score sw for an eligible cell y is:

$$sw_y = \begin{cases} 1, \text{if a window is present in } y, \\ \\ 0.5, \text{if a window is not required and is not present in } y, \\ \\ 0, \text{otherwise (if a window is required, but is not present in } y). \end{cases} \tag{3.2}$$

Figure 3.3 shows the scores assigned to each eligible cell when a bedroom is to be placed while the window rule is in effect. Scores go from lowest (in red) where there are no windows to highest (in green) where there is a window present. Figure 3.4 shows the scores when a dining room is to be placed, instead. The dining room, unlike the bedroom, does not need a window. Thus, the score for eligible cells without windows is only halved (in yellow), compared to eligible cells containing a window (in green).

Social rule

The purpose of the social rule is to make it more likely for social rooms (e.g., the living room) to be placed near the entrance of the building. This rule only affects room requisites R_R which are of the social type (it does not apply to rooms of the service and private types). The further away the eligible cell is from the entrance, the lower its score will be for the social room.

The score ss for an eligible cell y is:

$$ss_y = 1 - \frac{dist(cell_y, cell_D)}{(max_i(dist(cell_i, cell_D))}, \tag{3.3}$$

where $dist(cell_y, cell_D)$ is the distance between the eligible cell y and the cell D which contains the entrance door and $max_i(dist(cell_i, cell_D))$ is the maximum distance between cell D and any cell i

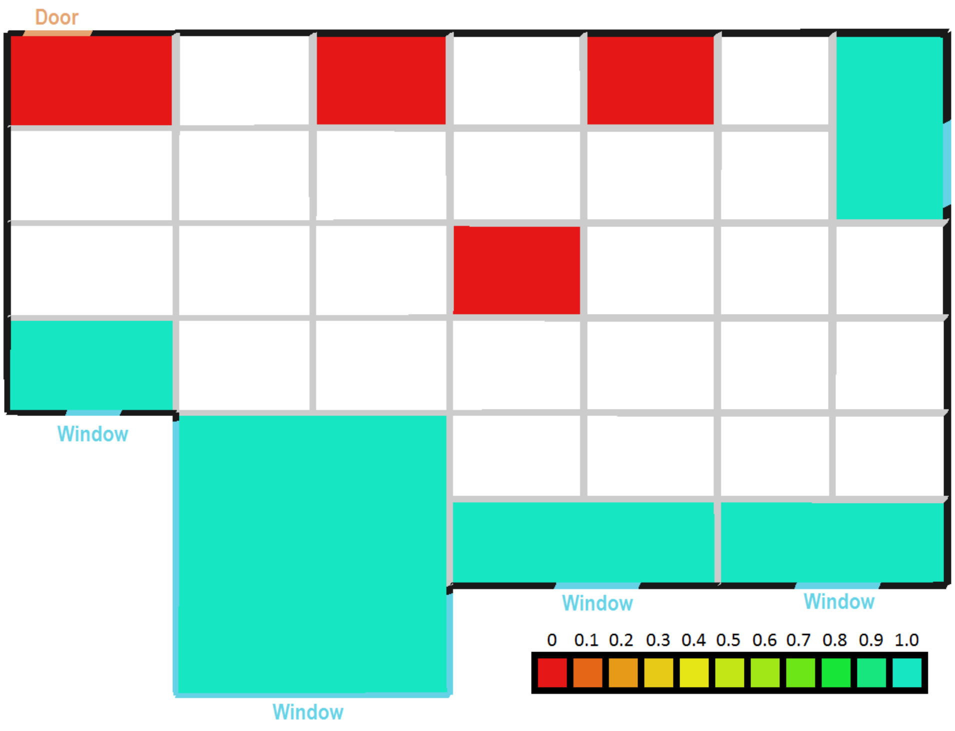

Figure 3.3: The score associated to a set of eligible cells when a bedroom is inserted and the window rule is in effect. A bedroom requires a window, so the window rule sets the score of eligible cells without a window to zero (in red). Windows are cyan line segments on the outer edge of the floor plan.

in the grid.

Figure 3.5 shows the scores assigned to each eligible cell when a living room is to be placed and the social rule is in effect. Scores go from highest (in green) near the entrance door, on the top left of the floor plan, to lowest (in red) the furthest away from the entrance, on the bottom right.

Private rule

The purpose of the private rule is to make it more likely for private rooms (e.g., the bedroom) to be placed away from the entrance of the building. This rule only affects room requisites of the private type (it does not apply to rooms of the service and social types). The further away the eligible cell is from the entrance, the higher will be its score for the private room.

The score sp for an eligible cell y is:

$$sp_y = \frac{dist\left(cell_y, cell_D\right)}{max_i\left(dist\left(cell_i, cell_D\right)\right)}. \tag{3.4}$$

Figure 3.6 shows the scores assigned to each eligible cell when a bedroom is to be placed and the private rule is in effect. Scores go from lowest (in red) near the entrance door, on the top left of the floor plan, to highest (in green) the furthest away from the entrance, on the bottom right.

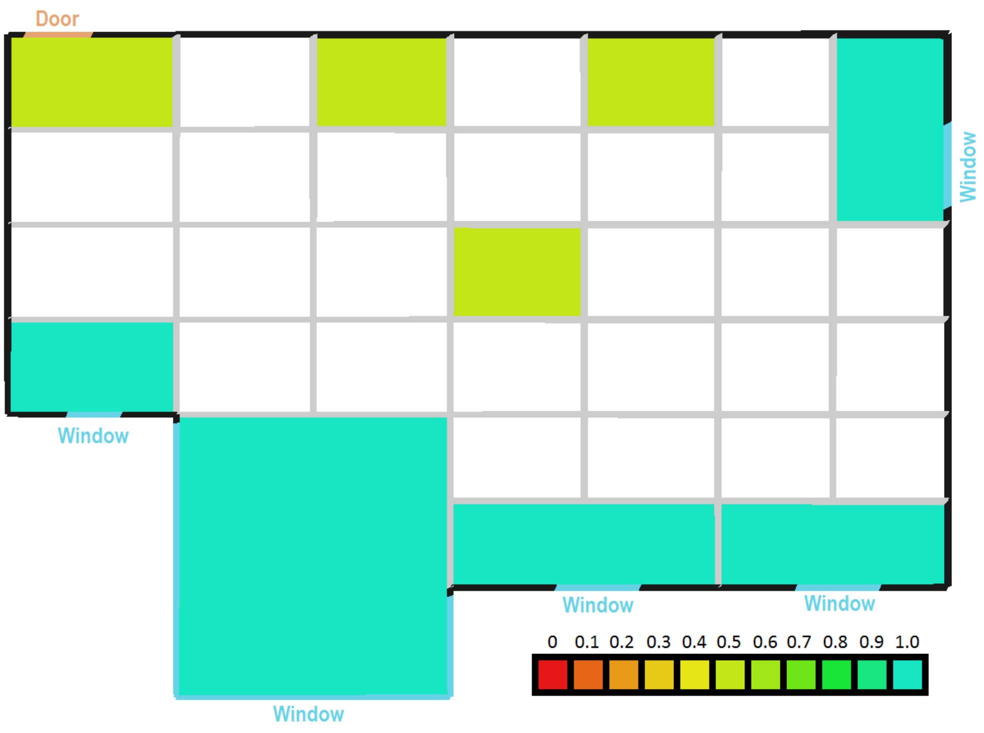

Figure 3.4: The score associated to a set of eligible cells when a dining room is inserted and the window rule is in effect. A dining room does not require a window, so the window rule only halves the score of eligible cells without a window (in yellow). Windows are cyan line segments on the outer edge of the floor plan.

Elastic connection rule

The purpose of the elastic connection rule is to make it more likely for rooms of the same type to be kept close together. When a room is placed, all rooms of the same type placed thereafter will be affected. The eligible cells close to the room that has already been placed receive higher scores, and eligible cells further away receive lower scores.

For example, if a bedroom (a room of the private type) is placed in the floor plan, then if another room of this same type is to be placed, the eligible cells close to the bedroom will receive higher scores. Note that the elastic connection rule applies based on room type, not on whether a connection requisite R_C has been defined for these rooms.

If multiple rooms of the same type have been placed, and a new room of the same type is being placed, then the elastic connection rule will apply multiple times, one time for each room already placed.

The score se for an eligible cell y is:

$$se_y = \prod_{i=1}^{n} 0.5^{dist\left(cell_y, cell_j\right)}, \tag{3.5}$$

where $dist\,(cell_y, cell_j)$ is the distance between the eligible cell y and each cell j containing a previously placed room which shares the same type in its R_R as the room being placed. Note that

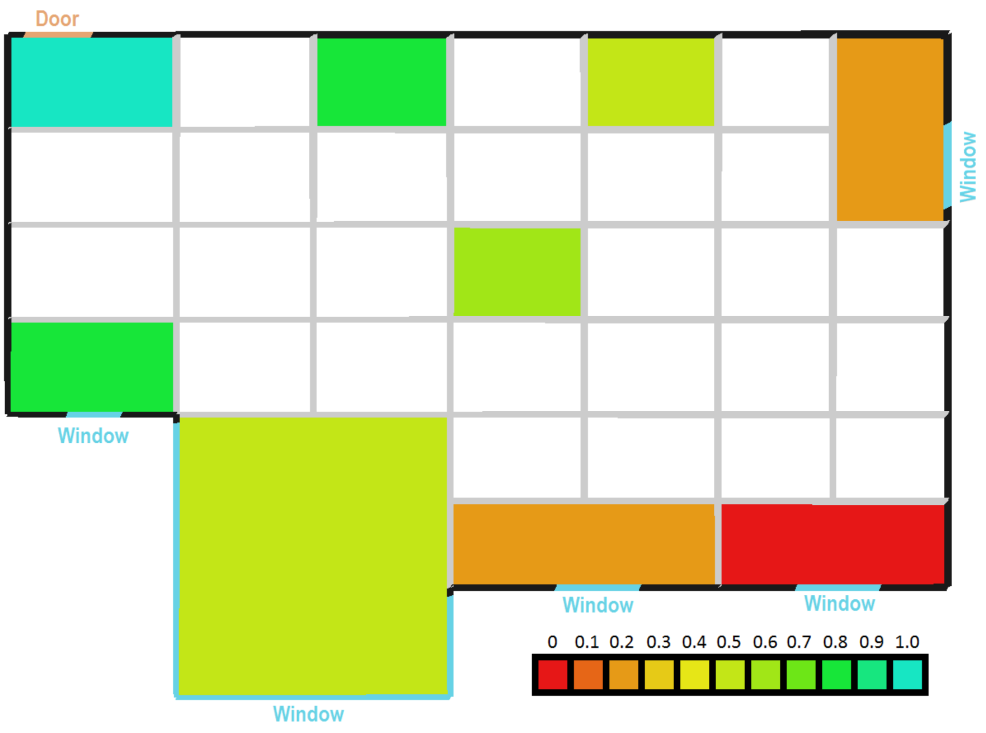

Figure 3.5: The score associated to a set of eligible cells when a living room is to be placed and the social rule is in effect. The living room is a social room, so the social rule decreases the score of eligible cells the further away they are from the building's entrance (the brown line segment located on the top left of the floor plan).

se applies even if a connection requisite R_C has not been defined for these rooms.

Figure 3.7 shows the scores assigned to each eligible cell when a bedroom has already been placed (in magenta) and a second bedroom is to be placed while the elastic connection rule is in effect. Scores go from highest (in green) near the bedroom that has already been placed to lowest (in yellow), further away.

Hard connection rule

The purpose of the hard connection rule is to enforce user-provided connection requisites R_C. For example, when the user determines that a kitchen must be adjacent to a dining room, the hard connection rule applies for these two specific rooms.

The difference between the elastic connection rule and the hard connection rule is that, while the elastic connection rule applies for all rooms of the same type, the hard connection rule only applies for the pair of rooms associated in a connection requisite R_C. Further, while the elastic connection rule creates a field of gradually decreasing scores, the hard connection rule applies a cutoff distance threshold t_H to set the score of all eligible cells beyond t_H meters to zero. This forces adjacency between the rooms in R_C.

When the hard connection rule is in effect, the first room in a connection requisite R_C is placed normally. When the second room in R_C is to be placed, the eligible cells closest to the first room

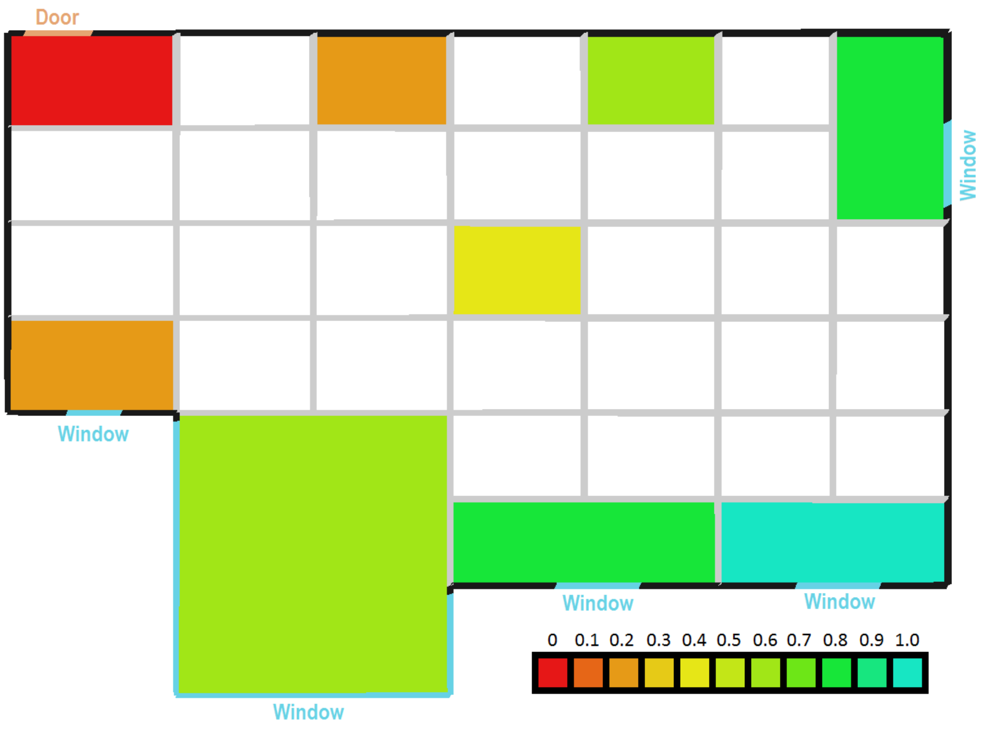

Figure 3.6: The score associated to a set of eligible cells when a bedroom is to be placed and the private rule is in effect. The bedroom is a private room, so the private rule decreases the score of eligible cells the closer they are to the building's entrance (the brown line segment located on the top left of the floor plan).

receive the highest score, and those further away receive lower scores. Effectively, the first room in a connection requisite is not affected by the hard connection rule, and the second room is affected by the position of the first room.

The score sh for an eligible cell y is:

$$sh_y = \begin{cases} 0.5^{dist(cell_y, cell_h)}, \text{if } dist(cell_y, cell_h) < t_H \\ 0, \text{otherwise,} \end{cases} \tag{3.6}$$

where $dist\,(cell_y, cell_h)$ is the distance between the eligible cell y and the cell h containing a previously placed room with which adjacency must be kept. The threshold distance t_H is defined as twice the length of t_W (the wall length threshold of the grid); that is, effectively any eligible cell beyond two times the threshold distance t_W has its score set to zero.

In Figure 3.8, a room requisite exists for a dining room and a kitchen. The dining room has already been placed (in magenta) and we are now placing a kitchen, which must be kept close. The figure shows the score assigned to each eligible cell, according to the distance from the dining room. The highest scores (in green) are closest to the dining room, with some lower scores further away (in yellow). The cutoff threshold distance sets the score of the remaining eligible cells to zero (in red).

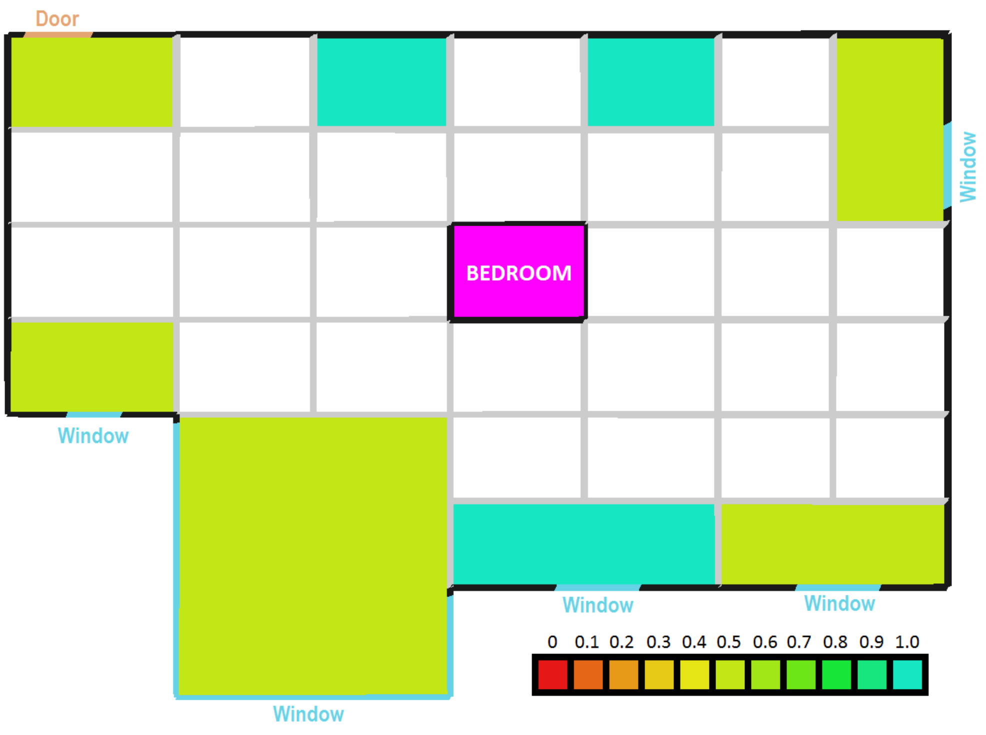

Figure 3.7: The score associated to a set of eligible cells when a bedroom has already been placed in the floor plan and a second bedroom is to be placed while the elastic connection rule is in effect. The elastic connection rule attributes higher score to eligible cells which are closer to the room that has already been placed.

Entrance door rule

The cell which contains the entrance door is a special case because only rooms of the social type can be placed there. Rooms of the private and service type cannot be placed at the entrance because these types block connectivity. For example, a bedroom cannot be placed at the entrance, because a bedroom cannot serve as a path to the rest of the building.

The score sd_y for an eligible cell y is:

$$sd_y = \begin{cases} 0, \text{if a door is present in } y \text{ but } R_R \text{ is not of the social type,} \\ \\ 1, \text{otherwise.} \end{cases} \tag{3.7}$$

Small window rule

The small window rule applies only to eligible cells containing a window. If the cell cannot fulfill any dimension in a room requisite R_R, the eligible cell's score is set to zero. The small window rule effectively prevents rooms which require large windows from being assigned to eligible cells containing small windows.

The score sk_y for an eligible cell y is

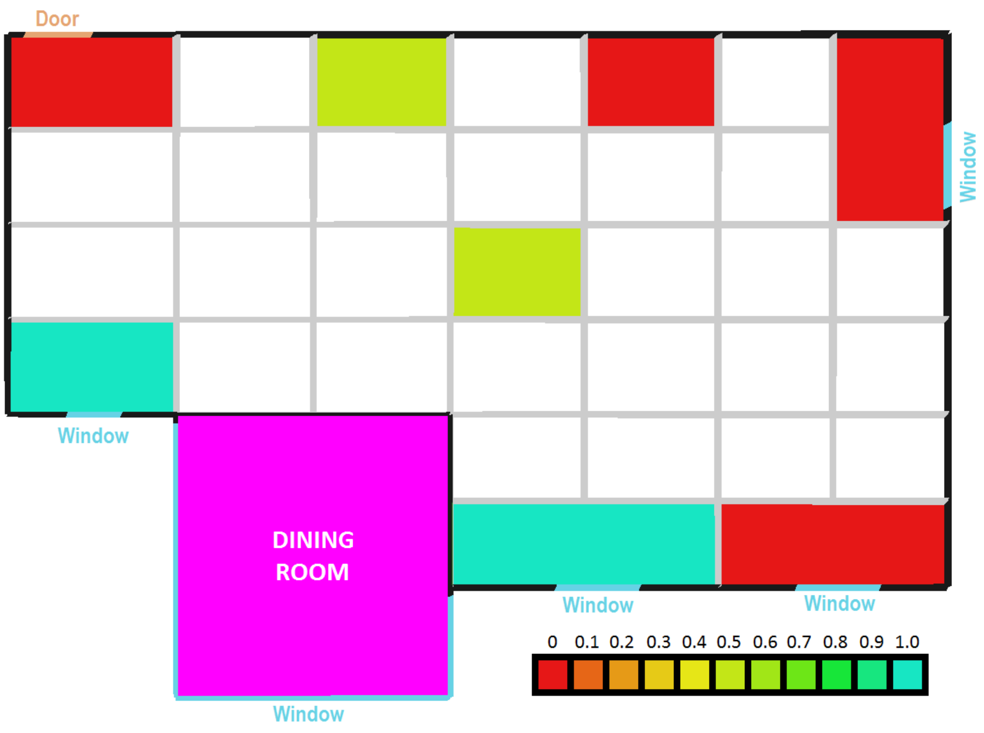

Figure 3.8: The score associated to a set of eligible cells when a room requisite exists between a dining room (already placed, in magenta) and a kitchen (currently being placed), and the hard connection rule is in effect. The eligible cells closest to the dining room have the highest score (in green), with some lower score further away (in yellow). Beyond the threshold t_H, the score is set to zero (in red).

$$sk_y = \begin{cases} 0, \text{if a window is present in } y \text{ but no dimension of } R_R \text{ is fulfilled by } y, \\ \\ 1, \text{otherwise.} \end{cases} \tag{3.8}$$

3.3.3 Selecting an eligible cell

After the score of each eligible cell y has been determined for the room being currently placed, the score array S is normalized, such that:

$$P_y = \frac{S_y}{(\sum_{x=1}^{n} S_x)}, \tag{3.9}$$

where x is every eligible cell for which a score S has been calculated. Thus, each normalized score P_y is the probability for a given eligible cell y to be randomly selected.

An example: if three eligible cells are available, with $S_1 = 0.1$, $S_2 = 0.6$, and $S_3 = 1$, the probabilities are, respectively:

$$P_1 = \frac{S_1}{(S_1+S_2+S_3)} \approx 5.9\%,$$

$$P_2 = \frac{S_2}{(S_1+S_2+S_3)} \approx 35.3\% \text{ and} \tag{3.10}$$

$$P_3 = \frac{S_3}{(S_1+S_2+S_3)} \approx 58.8\%.$$

When an eligible cell is selected, it is occupied by the current room and removed from the list of eligible cells. On the next iteration, it will not receive a score S, nor be considered for selection. The process of scoring and selecting eligible cells is repeated iteratively until every room requisite R_R has been placed. If there are no eligible cells left in the grid and there are still room requisites R_R to be placed, the procedural generation process fails. Either the floor plan is too small for the architectural plan, or the threshold used to generate eligible cells is too high.

3.4 Iterative room expansion

The room expansion process is performed after each room requisite R_R has been placed in the floor plan as a single-cell room (see Figure 3.1(b)). It expands each single-cell room to its final size, handling conflicts between rooms which are competing for space. This step also ensures that all rooms can be accessed from the entrance, keeping cells needed for room connectivity from being occupied.

3.4.1 Scoring expansion options

A room's expansion is performed in iterative steps, and during each step, the room expands in a single direction. Each expansion option o (up, down, left and right) receives a score K according to how beneficial o is toward fulfilling the goals set by the room requisite R_R. The score for an expansion option o is:

$$K_o = kb_o \; kw_o \; kn_o, \tag{3.11}$$

where kb_o, kw_o and kn_o are factors weighed between 0 and 1, referring to scores associated with different rules.

Blocked direction rule

The purpose of the blocked direction rule is to prevent rooms from overlapping or leaving the boundaries of the building's external walls as a result of an expansion step. The score kb_o for an expansion option o is:

$$kb_o = \begin{cases} 0, \text{if choosing } o \text{ causes rooms to overlap or leave building,} \\ \\ 1, \text{otherwise.} \end{cases} \tag{3.12}$$

Window waste rule

Windows are limited resources in a floor plan, given that all rooms benefit from having a window. The purpose of the window waste rule is to make it less likely for an expansion option to be chosen if it causes a room to receive a window, when the room already has one. The score kw_o for an expansion option o is:

$$kw_o = \begin{cases} 1, \text{if } n_w = 0 \text{ or } n_w = 1, \\ \\ \frac{0.2}{(n_w - 1)}, \text{if } n_w > 1, \end{cases} \tag{3.13}$$

where n_w is the number of windows within the room if the expansion option o is chosen.

Necessity rule

The purpose of the necessity rule is to make it less likely that an expansion option will be chosen if the expansion adds to a dimension that has already been expanded to the minimum size required in the room requisite R_R. The score kn_o for an expansion option o is:

$$kn_o = \begin{cases} 1, \text{if } dim < R_{minW} \wedge dim < R_{minL}, \\ 0.5, \text{if } dim > R_{minW} \oplus dim > R_{minL}, \\ 0.1, \text{otherwise,} \end{cases} \tag{3.14}$$

where dim is the dimension increased when choosing the expansion option o, R_{minW} is the minimum required width in R_R and R_{minL} is the minimum required length in R_R.

Thus, the kn score is highest when the expansion option increases a dimension that does not fulfill neither the width nor the length requisite, it is lower when it increases a dimension that already fulfills either the width or the length requisite, and lowest when it increases a dimension that already fulfills both the width and the length requisites.

3.4.2 Selecting an expansion option

To select an expansion option o, the score array K is normalized, such that:

$$P_o = \frac{K_o}{(\sum_{x=1}^{n} K_x)}, \tag{3.15}$$

where x is every expansion option for which a score K has been calculated. Thus, each normalized score P_o is the probability for a given expansion option o to be randomly selected.

At this point, each expansion option o has received a score K_o referring to its fitness toward fulfilling the goals set by the room requisite R_R (see Equation 3.11).

3.4.3 Expansion steps

Rooms are expanded during three distinct steps: the first expansion step, the fixing step and the second expansion step. The purpose of the first expansion step is to grow each single-cell room to its minimum size, as determined in the room requisite R_R. First, the rooms are sorted in a list according to the priority of their respective R_R. Then, each room is selected in order, from highest to lowest priority, and expanded until either its minimum size is achieved, or no expansion option is available. When the list of rooms is fully traversed, the algorithm moves on to the fixing step.

The fixing step attempts to expand rooms which remain undergrown after the first expansion step. To do so, the algorithm attempts to liberate space along the dimensions in which the undergrown room still requires growth to fulfill its R_R. To liberate space in a given direction (either up/down or left/right), each room adjacent to the undergrown room (in the relevant direction) contracts one cell away. After the contraction, the room which contracted grows again to its minimum size (but not toward the direction it contracted from). Finally, the undergrown room occupies the liberated space. An undergrown room may cause multiple adjacent rooms to contract.

Figure 3.9 presents an example of the fixing step liberating space for an undergrown room. In Figure 3.9 (a), the red room on the top left corner is undergrown, but is blocked by the exterior wall (on the up direction) and the blue room (on the down direction). The blue room contracts down, liberating space for the red room (Figure 3.9 (b)). The blue room immediately expands, to once again fulfill its room requisite R_R (Figure 3.9 (c)). The up direction (from which blue contracted) is forbidden during this expansion. Finally, the red room expands to its minimum requisite size (Figure 3.9 (d) and (e).

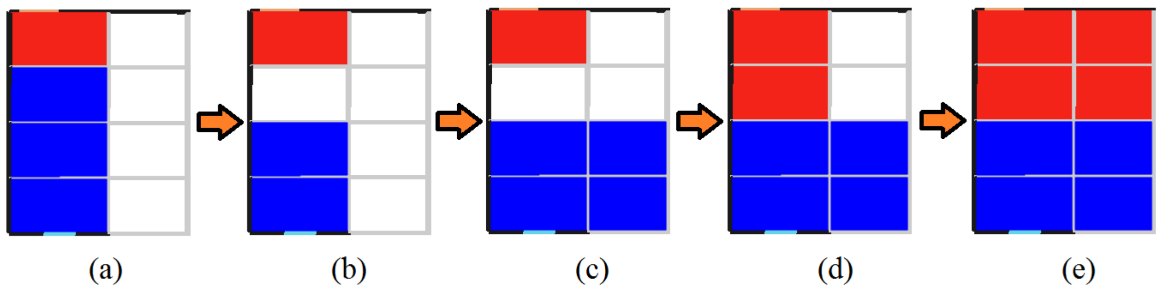

Figure 3.9: The room expansion process' fixing step. The red room is undergrown and blocked along the up/down direction in the top-left corner (a). The blue room liberates space by contracting down (b) and immediately growing (c). The red room then expands to its minimum required size (d and e).

The fixing step does not always succeed in liberating space for an undergrown room. There are two cases in which the fixing step fails. The first case is when the adjacent room is one cell wide and cannot contract away from the undergrown room. The second case is when an adjacent room contracts, but fails to expand afterward. In this case, the space cannot be liberated and the contraction is unmade.

If the fixing step cannot expand an undergrown room, the user requirement R_R cannot be fulfilled and the floor plan is considered invalid.

If all rooms have been expanded successfully during the first expansion step (or fixed during the fixing step), the room expansion process moves to the second expansion step. The purpose of this step is to occupy the remaining space in the floor plan. The rooms are once again sorted by priority, according to their room requisites R_R, but this expansion step is different from the first. This time, rooms take turns to expand a single step, and an additional rule is used: if an expansion option is chosen that causes any room in the floor plan to be blocked from the entrance, the expansion is reverted and the room is removed from the list of rooms to be expanded. Thus, rooms expand as long as they do not disrupt room connectivity. Note that social rooms propagate connectivity. Circulation areas are created in the cells which remain unclaimed by any room at the end of the expansion process.

The final step determines which walls can be used to connect rooms. During 3D extrusion, these walls become either open walls or doors between rooms. Our approach does not use a connection tree, so instead connectivity is determined considering each two adjacent room's types, using rules for which room types can be connected.

3.5 Full example

This section presents a full execution of the procedural generation algorithm.

Room placement step:

- Placing room BED1, rules active: window, private; see Figure 3.10.
- Placing room BED2, rules active: window, private, elastic (BED1); see Figure 3.11.
- Placing room BED3, rules active: window, private, elastic (BED1, BED2); see Figure 3.12.
- Placing room DIN, rules active: window, social; see Figure 3.13.
- Placing room KIT, rules active: window, hard (DIN); see Figure 3.14.
- Placing room LIV, rules active: window, social, elastic (DIN); see Figure 3.15.
- Placing room BATH1, rules active: window, private, elastic (BED1, BED2, BED3), hard (LIV); see Figure 3.16.
- Placing room BATH2, rules active: window, private, elastic (BED1, BED2, BED3, BATH1); see Figure 3.17.
- Finished placing rooms; see Figure 3.18.

First expansion step (see Fig. 3.19):

- 1 - expanding DIN up.
- 2 - expanding DIN right, minimum size achieved.
- 3 - expanding KIT up.
- 4 - expanding KIT left, minimum size achieved.
- 5 - expanding LIV down.
- 6 - expanding LIV right, minimum size achieved.
- 7 - expanding BED1 left, minimum size achieved.
- 8 - BED2 already has minimum size.
- 9 - expanding BATH1 right, minimum size achieved.
- 10 - expanding BED3 up.
- 11 - expanding BED3 up, minimum size achieved.
- 12 - BATH2 already has minimum size.

Fixing step:

- No room to fix.

Second expansion step (see Fig. 3.19):

- 13 - DIN cannot expand, blocked, done expanding.
- 14 - KIT blocks connectivity (down and right), done expanding.
- 15 - LIV cannot expand, blocked, done expanding.
- 16 - expanding BED1 left.
- 17 - BED2 cannot expand, blocked, done expanding.
- 18 - BATH1 cannot expand, blocked, done expanding.
- 19 - expanding BED3 up.
- 20 - expanding BATH2 up.
- 21 - BED1 cannot expand, blocked, done expanding.
- 22 - BED3 cannot expand, blocked, done expanding.

- 23 - BATH2 blocks connectivity (up), done expanding.

In the final step, corridors and doors are created, generating the final floor plan seen in Figure 3.20.

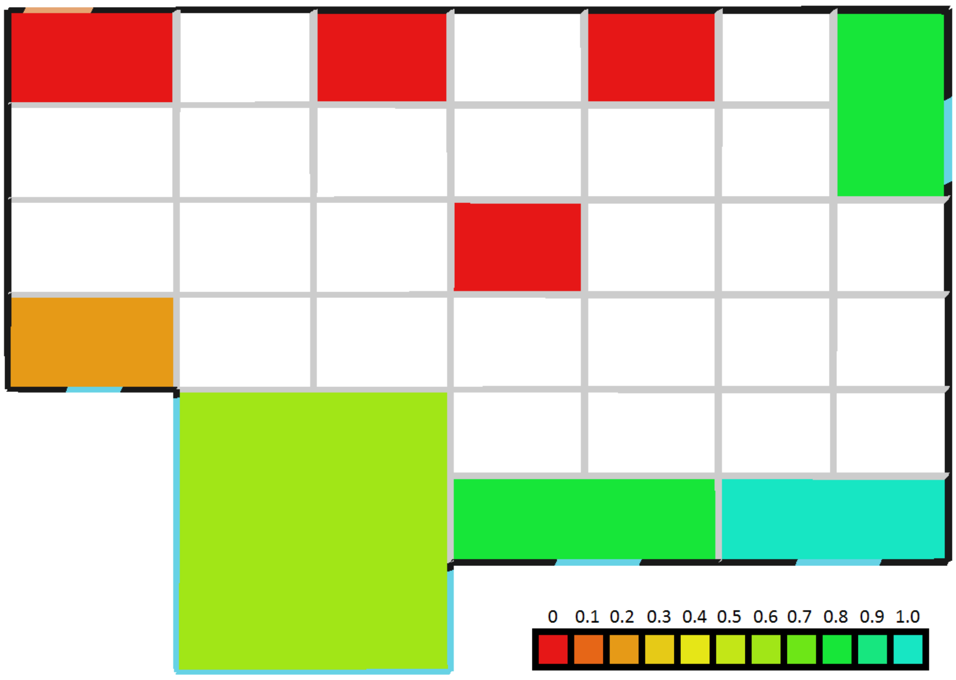

Figure 3.10: Eligible cell scores for the placement of the BED1 seed.

In this chapter, we presented our method for growth-based procedural floor plan generation, which creates an irregular grid considering the features of a building's external walls, places rooms without a connection tree, according to a set of rules and user requirements, and finally expands rooms in the grid to fulfill each room's minimum size while maintaining the connectivity of all rooms to the entrance. We also presented an example of the algorithm's execution. We now present the results achieved with this method.

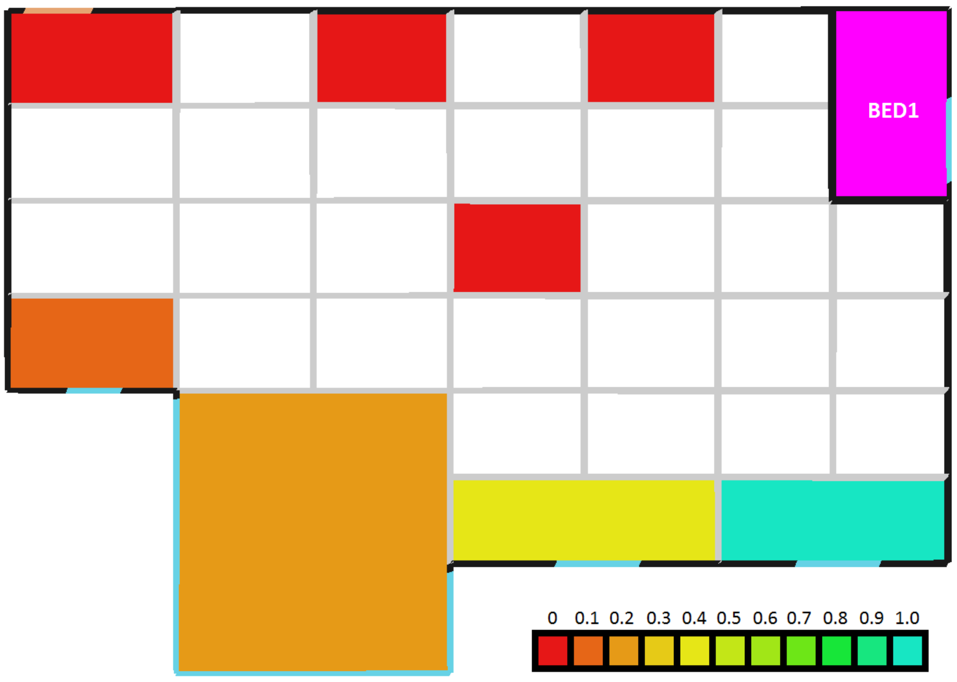

Figure 3.11: Eligible cell scores for the placement of the BED2 seed.

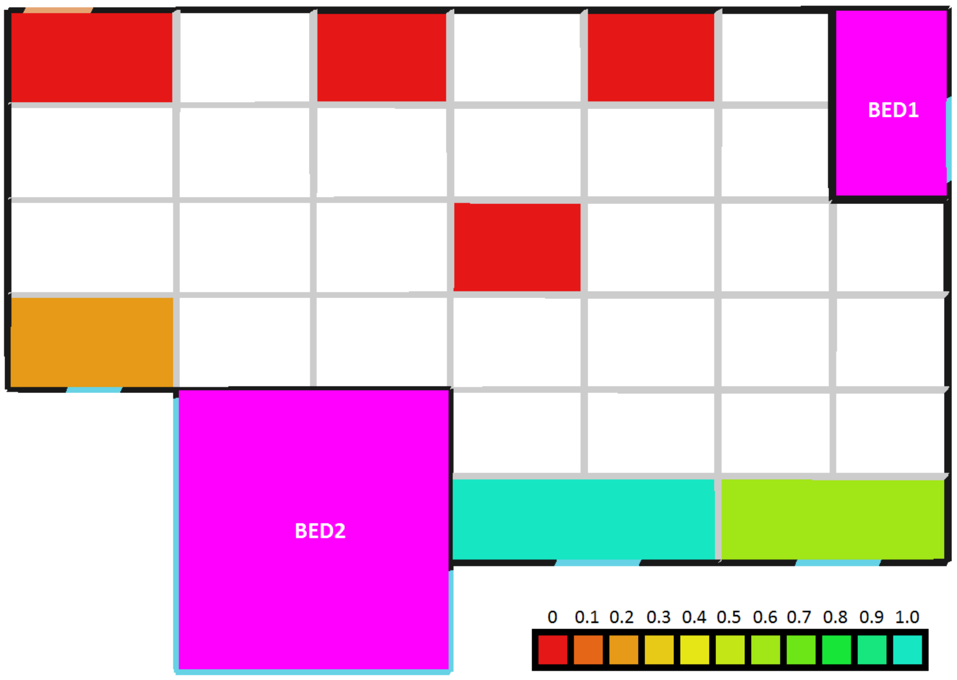

Figure 3.12: Eligible cell scores for the placement of the BED3 seed.

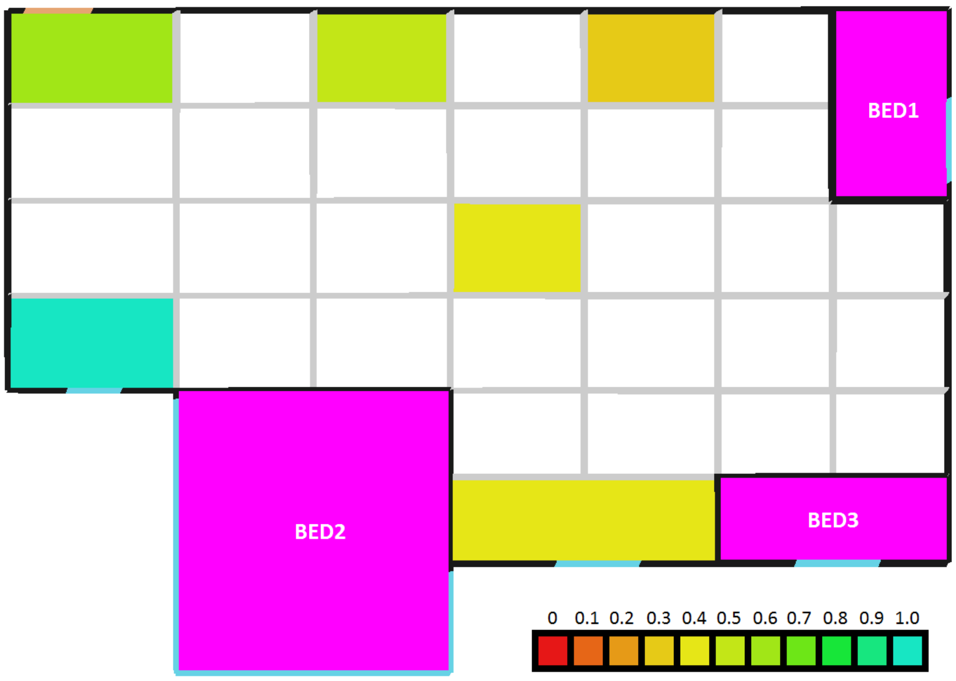

Figure 3.13: Eligible cell scores for the placement of the DIN seed.

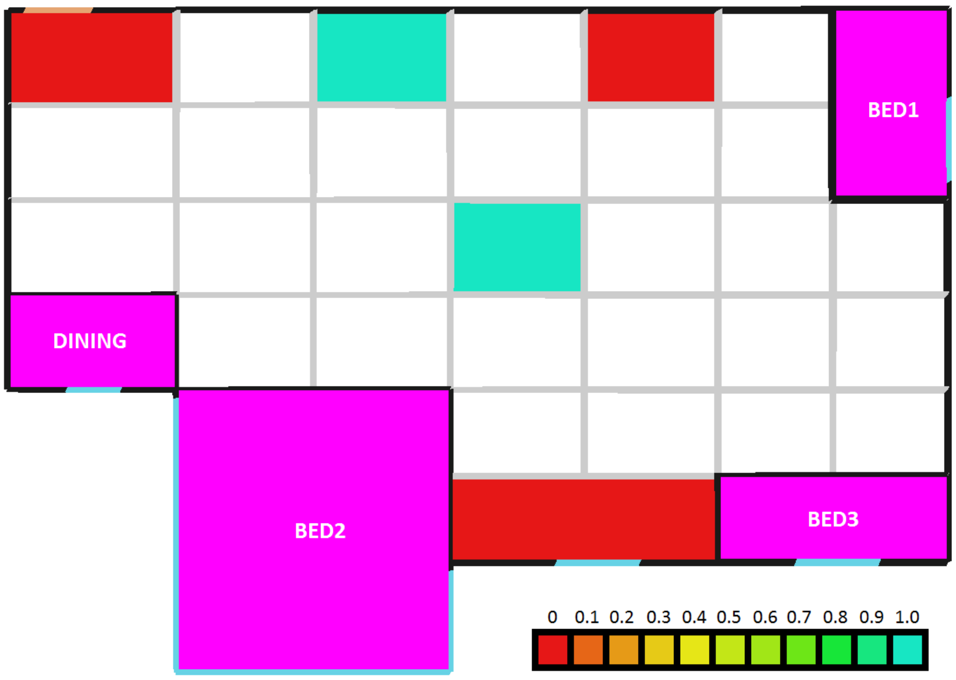

Figure 3.14: Eligible cell scores for the placement of the KIT seed.

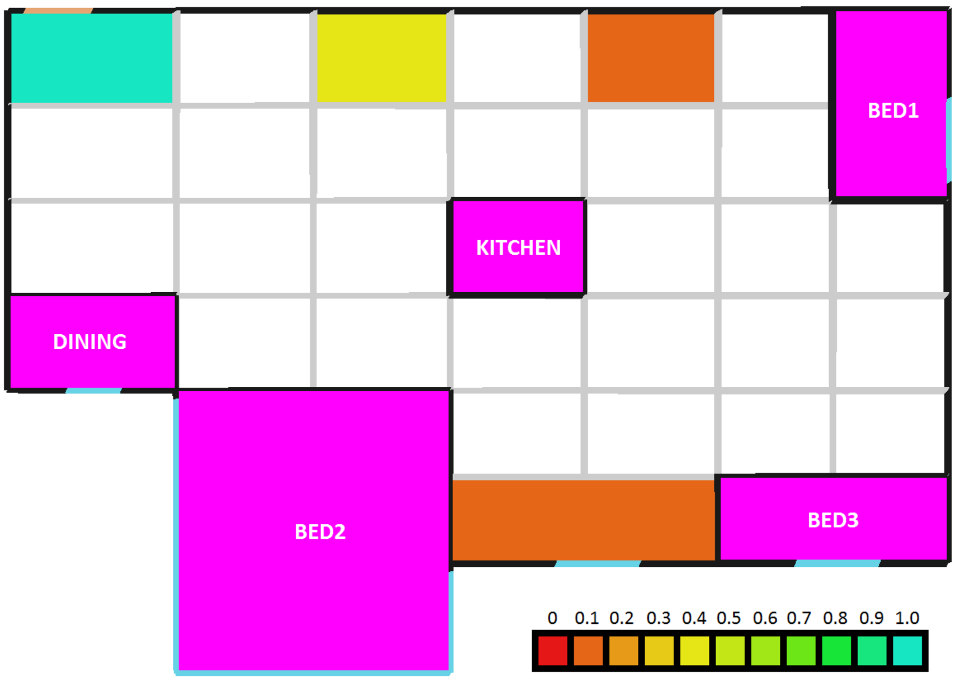

Figure 3.15: Eligible cell scores for the placement of the LIV seed.

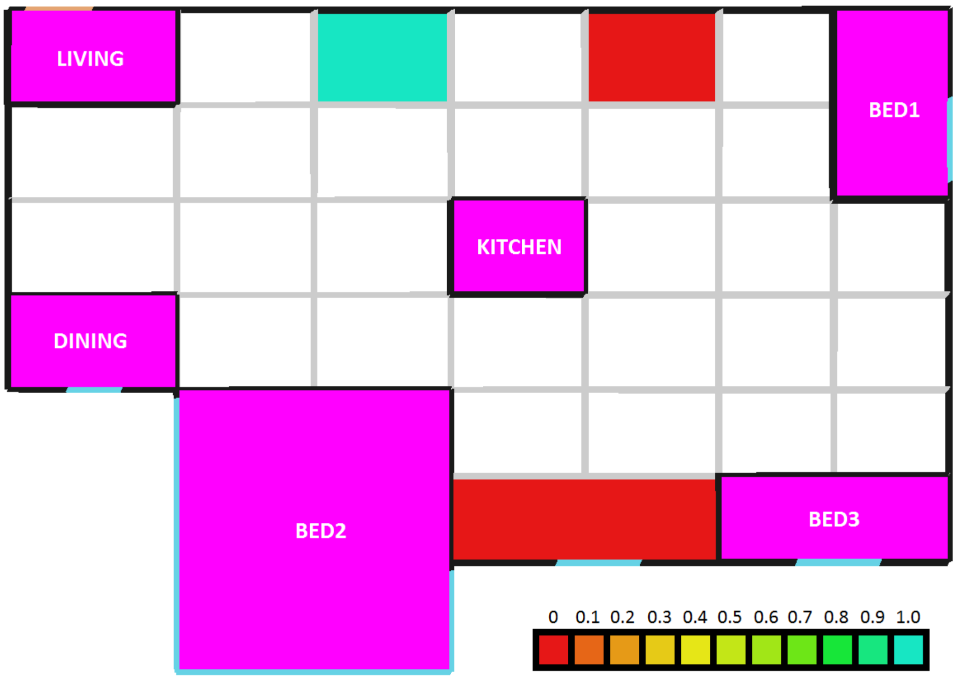

Figure 3.16: Eligible cell scores for the placement of the BATH1 seed.

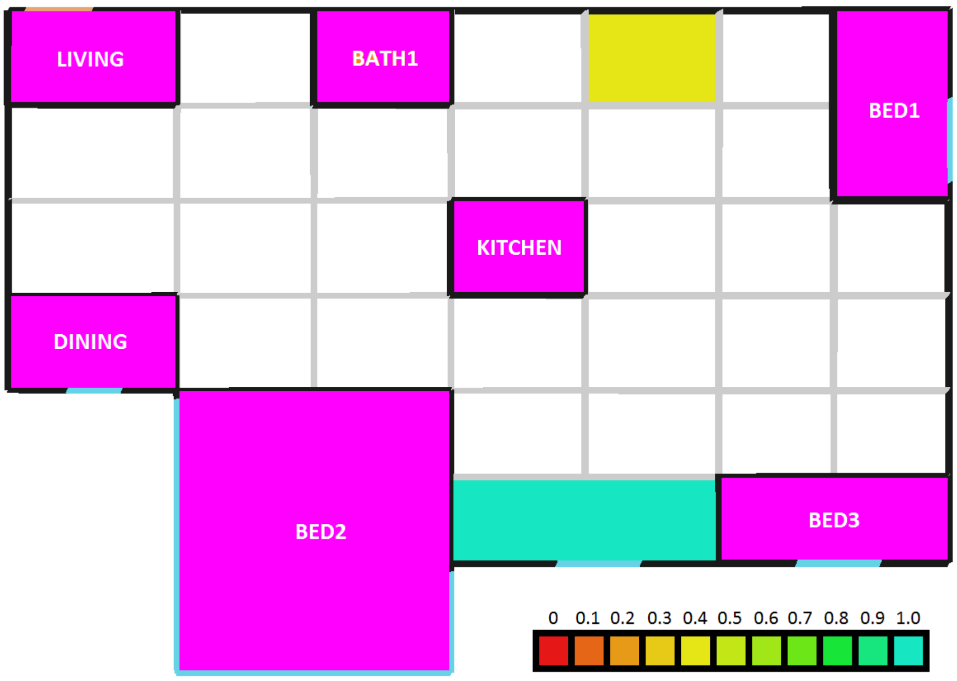

Figure 3.17: Eligible cell scores for the placement of the BATH2 seed.

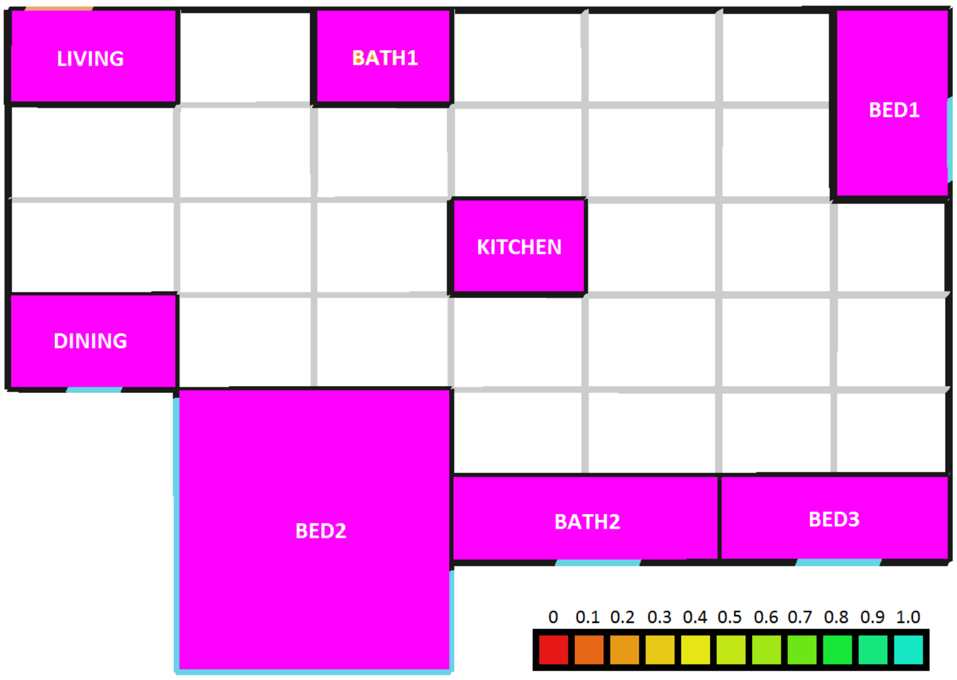

Figure 3.18: Final single-cell room placement.

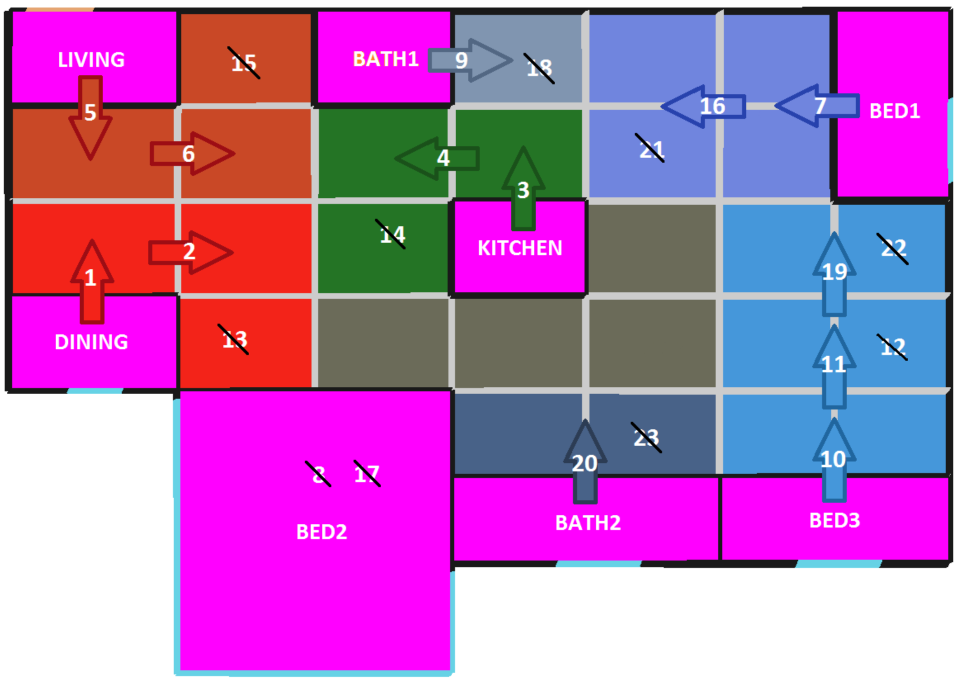

Figure 3.19: Room expansion process. Moves 1, 2, 3, 4, 5, 6, 7, 9, 10 and 11 are performed during the first expansion step. Moves 16, 19 and 20 are performed during the second expansion step. The stricken numbers are steps in which no expansion was performed.

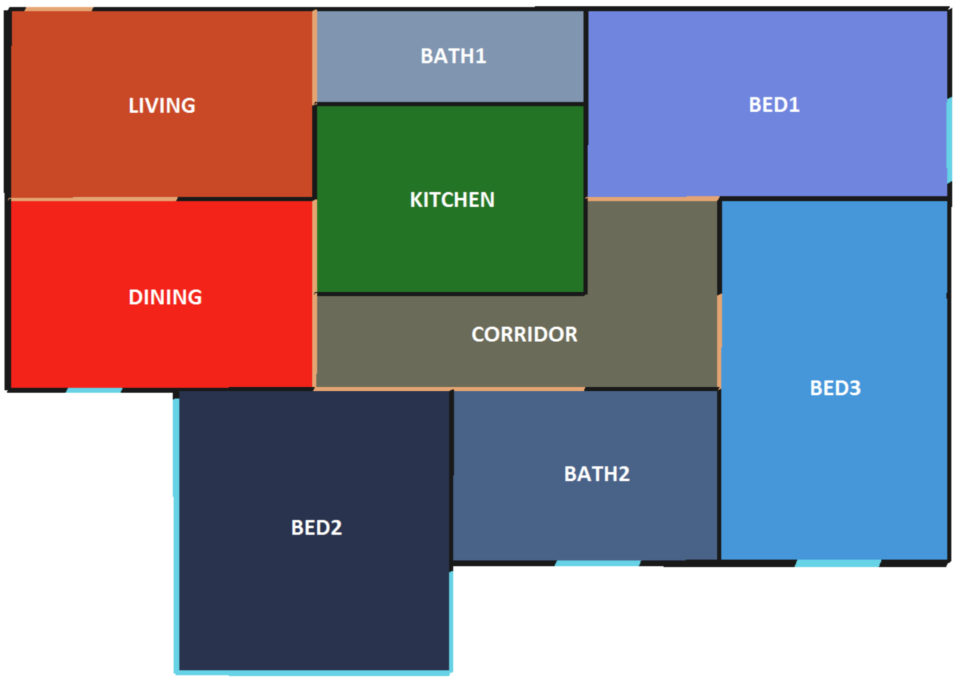

Figure 3.20: The final floor plan, with a corridor (in grey) and walls where internal doors are to be placed (the brown line segments between rooms).

4. RESULTS

In this section we present the results of this research work, as well as a discussion of the results.

4.1 Evaluation

Our first analysis is a qualitative evaluation of the floor plans generated by our method. Figure 4.1 presents a floor plan created by an architect (a) and three procedurally generated floor plans (b-d). To generate these floor plans we used room requisites R_R with similar size to the rooms seen in the original floor plan (a). The only connection requisite R_C defined an adjacency between the dining room and the kitchen. All other connections were left for the algorithm to sort out according to the default room placement rules. All results use default threshold configurations ($t_W = 1.8m$ and $t_D = 2.5m$).

Table 4.1 presents the area assigned to each type of room for the different floor plans seen in Figure 4.1. Note that the area assigned to each type of room varies because the algorithm does not attempt to create the exact same rooms seen in (a), but rather to achieve the minimum dimensions required by the room requisites R_R. However, the proportional area occupied by each remains mostly the same. For example, the bedrooms occupy the most space in all floor plans.

Figure 4.2 presents the connection trees for the floor plans seen in Figure 4.1. We note that private rooms remain in higher levels and social rooms in lower levels of the procedurally generated trees, with mostly the same types of connections. We conclude that in this example the procedural floor plan generator was capable of generating different room layouts with approximately the same topology and area distribution as the floor plan created by an architect (a).

Table 4.1: The area occupied by each type of room in the floor plans seen in Figure 4.1.

	Fig. 4.1 (a)	Fig. 4.1 (b)	Fig. 4.1 (c)	Fig. 4.1 (d)
Type	Real	Procedural	Procedural	Procedural
Kitchen area	$9m^2$	$7.23m^2$	$14.54m^2$	$5.36m^2$
Dining and Living area	$25.36m^2$	$19.97m^2$	$19.97m^2$	$22.7m^2$
Bedroom area	$28.4m^2$	$29.12m^2$	$24.36m^2$	$24.36m^2$
Bathroom area	$6.63m^2$	$8.17m^2$	$7.14m^2$	$10.88m^2$

4.2 Discussion of the algorithm's behavior

Unlike other algorithms (e.g., Marson and Musse [11] or Merrell et al. [13]), our algorithm does not use a explicit connection tree to predefine the adjacencies of rooms in the floor plan. Rather, during room placement (see Section 3.3), rooms may (or may not) be more likely to be placed close together. Only rooms defined in connection requisites R_C are guaranteed to be adjacent. The

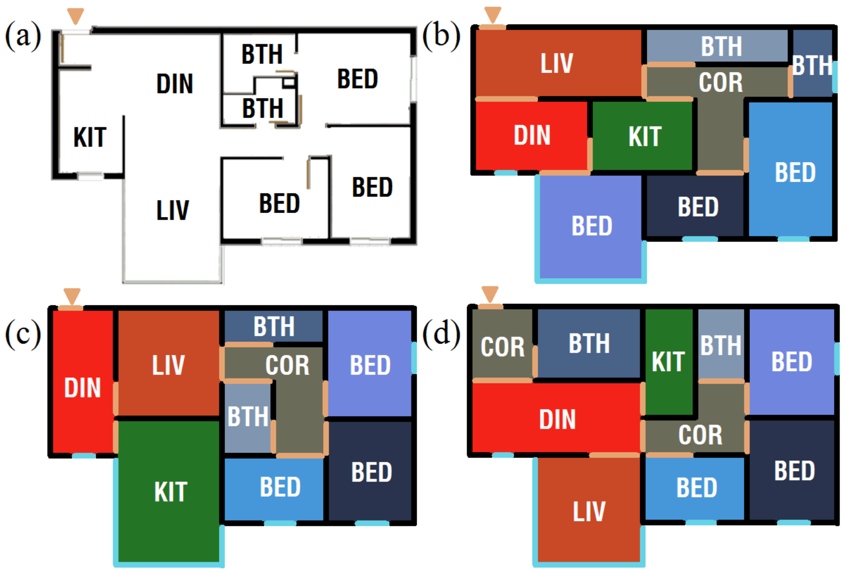

Figure 4.1: A floor plan generated by an architect (a) and three floor plans generated with a similar architectural plan (b-d). The brown line segments between rooms are walls on which an internal door is to be placed.

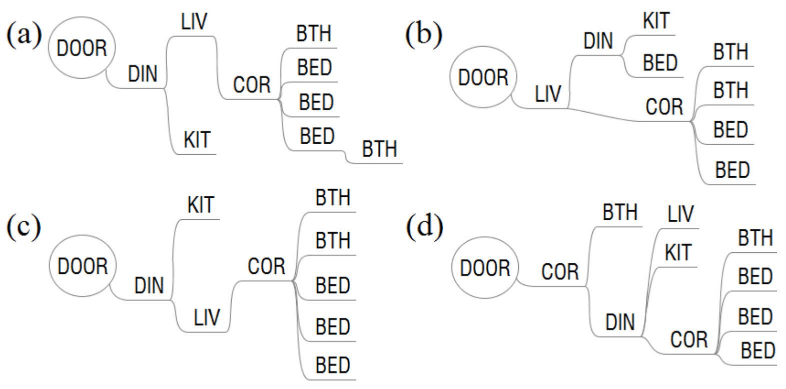

Figure 4.2: The connection trees for the floor plans seen in Figure 4.1.

objective of the presented algorithm is to create a floor plan within the boundaries of the building's exterior walls while expanding rooms to at least their minimum size and keeping rooms defined in requisites R_C adjacent. If the algorithm is successful, it manages to fulfill all R_R and all R_C and the resulting floor plan is considered valid.

In our approach diagonal elements do not extend lines and the grid remains axis-aligned even if diagonal walls are present in the input. Elements such as diagonal walls and openings otherwise exist normally in the cells in which they occur (see Figure 4.3).

The proposed procedural generation algorithm, discussed in Chapter 3, operates in iterative steps. First, it places each room occupying a single cell in the grid, and then grows each room iteratively to its final size. Each step of the algorithm presents several options for how to proceed, but with each step a single option is chosen. For example, when a room grows, it grows in a single direction, and when a room is placed, it is placed in a single cell. Thus, given the initial configuration (consisting of a grid, a set of eligible cells and a set of user requisites), a finite set F of possible floor plans

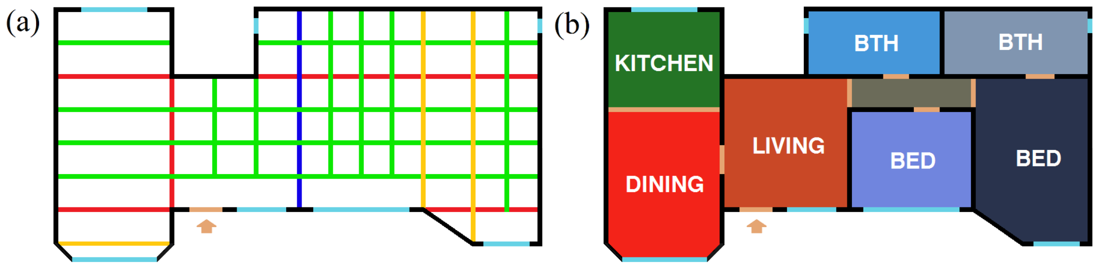

Figure 4.3: An input building with diagonal walls, with its resulting grid (a) and an example floor plan (b).

can be generated. This finite set F can be expanded by exhaustively exploring all permutations for room placement and all subsequent options for room growth. Note that most of the resulting layouts will be unacceptable, presenting rooms with inadequate size or connections considering the user requisites.

The factors of the scoring functions (see Equations 3.1 and 3.11) have two effects. First, the factors prohibit certain types of floor plan from being generated. For example, the window rule explicitly prohibits floor plans with windowless bedrooms in the finite set F. Second, the factors qualify the options available in each step. For example, the social rule assigns higher scores to eligible cells which are closer to the entrance, and this indicates that it is more beneficial to place the social room near the entrance. However, this does not prohibit the social room from being placed away from the entrance.

We allow a low score option to be selected because otherwise the same floor plan would always be generated. Further, selecting a low score option is not immediately optimal, but it can lead to better subsequent options and to a better floor plan overall. This is similar to the idea of avoiding local maximums in stochastic optimization. However, since the scoring process only considers the current state, it is equally possible that choosing a low score option will lead to further low score options and a worse floor plan overall.

Since decisions only consider the current state, it is also possible for the algorithm to choose an option that makes it impossible to create a valid floor plan (a floor plan which fulfills user requisites). This can happen in three situations: first, when there is no eligible cell where a room can be placed; second, when there is no room for a room to grow and achieve it's minimum requisite size; and finally, when the initial distribution or growth of rooms results in a layout where rooms cannot be accessed from the entrance.

4.3 Measurements

To measure how often invalid floor plans occur, we executed the algorithm using several different test cases. Each test case uses a specific building exterior (see Figures 4.4 and 4.5) and a specific number of room requisites R_R and connection requisites R_C. All test cases use default threshold configurations ($t_W = 1.8m$ and $t_D = 2.5m$) which result in a specific grid and number of eligible cells for each building exterior. For each test case the algorithm has been executed a thousand times, and the results can be seen in Tables 4.2 and 4.3, where *Outline* specifies which building exterior has been used, yn is the number of eligible cells available and R_R and R_C are the number of user-provided room and connection requisites.

There are three types of error which result in floor plans which are invalid and must be discarded. E_P is the number of times the room placement step failed to find an eligible cell for a R_R. This often occurs if a room defined in a connection requisite R_C is placed such that no adjacent eligible cell is available for the second room in R_C. The higher the number of R_C, the more likely it is for the error to occur. E_E is the number of times the first expansion step failed to fulfill the minimum size defined in R_R, followed by the failure of the fixing step to expand this undergrown room. Floor plans which provide little space for rooms to grow have higher room expansion error rates. E_C is the number of times one or more rooms were disconnected from the entrance after the first expansion step. To be disconnected, the room must not be reachable through social areas (such as circulation areas or rooms with the social type). A higher number of non-social R_R generates additional blockages, increasing the chance for connectivity errors to occur. Finally, FIX is the number of times the fixing step succeeded, fixing at least one undergrown room and recovering a floor plan which would otherwise be invalid with an E_E occurrence, SUC is the number of times a valid floor plan has been generated and T_t is the total execution time in milliseconds.

Among the results in Tables 4.2 and 4.3, some observations can be made. First, increasing the number of requisites R_R and R_C creates additional constraints which make it more difficult for rooms to be placed and to be expanded. As expected, this results in a higher number of invalid floor plans. Second, a higher number of invalid floor plans result in a lower total execution time. This is because by failing (through an occurrence E_P, E_E or E_C), the algorithm stops and exits early. Third, increasing the number of R_R and R_C does not significantly impact the execution time of the algorithm. This is because if there are more rooms to be expanded, each room has less space to occupy. However, the run time for each building outline varies because the different grid and distribution of eligible cells impacts the number of iterations needed for the algorithm to finish. Fourth, in some cases (for example cases i3, i4 and i5), the number of successes is zero because the number of room requisites R_R is higher than the number of eligible cells yn available. This effectively means that there is no available cell for rooms to be placed and the user requisites cannot be fulfilled. This is different from cases o4 and o5, where the number of R_R is lower than the number yn. In these cases, the building was not large enough to accommodate all the user requisites. Fifth, it takes only a few milliseconds to generate each floor plan.

4.4 Effect of the rules

As mentioned in Section 4.2, the scoring functions (see Equations 3.1 and 3.11) have the effect of prohibiting certain floor plans from occurring in the set F of floor plans which can be generated for a certain input, and also the effect of modifying the probability for each floor plan to be generated. The purpose of the rules, then, is to make valid floor plans more likely to occur.

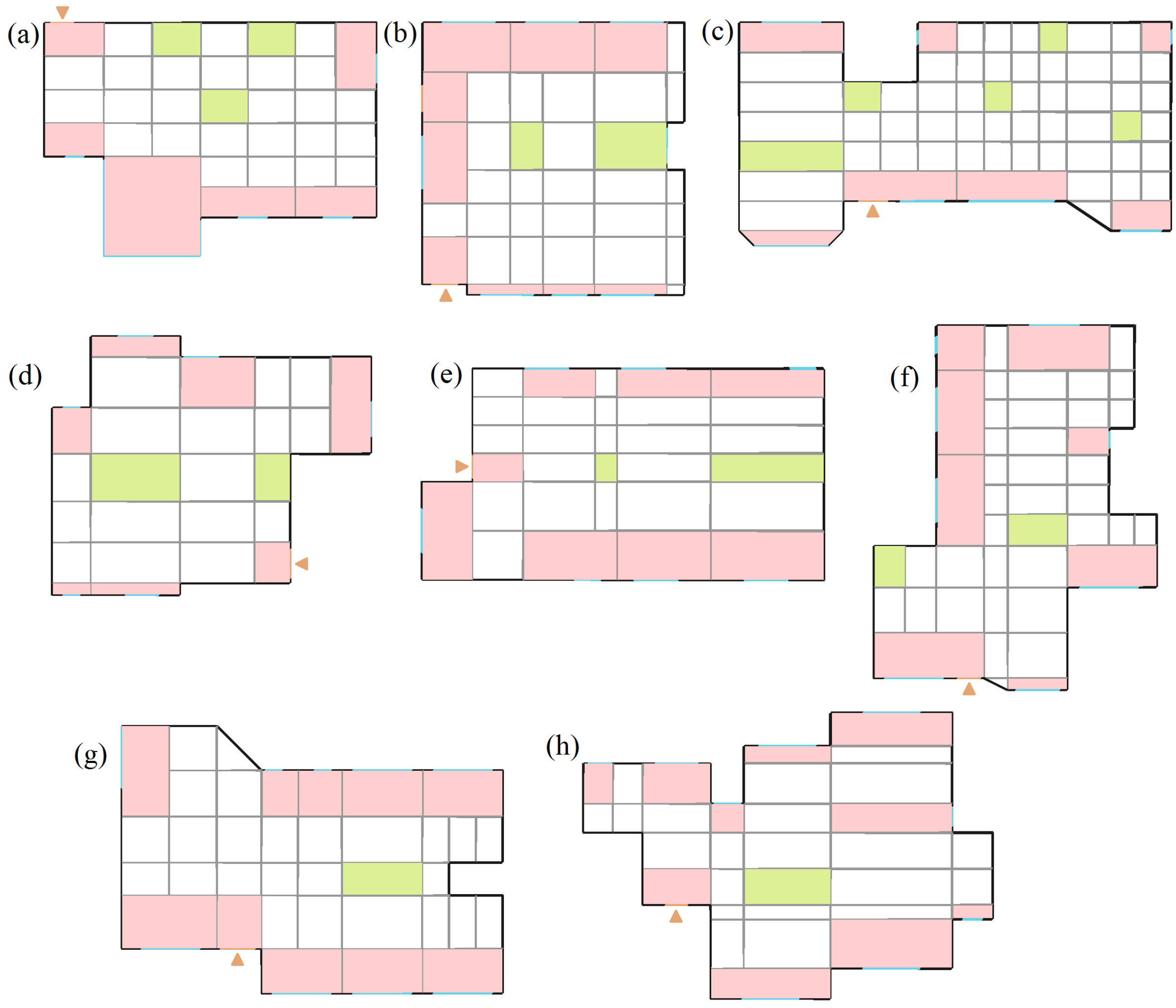

Figure 4.4: Outlines a-h used in the measurements.

Ablating a factor of the eligible cell score function (see Equation 3.1) modifies the probabilities assigned to each eligible cell. Removing sw_y allows requisites R_R of the bedroom type to be placed in any eligible cell. It also removes the modifier that makes it more likely for eligible cells with windows to be selected, and as a result rooms will stop favoring windows and be distributed more evenly in the grid. Removing ss_y will remove the modifier that makes it more likely for eligible cells near the entrance to be selected for social rooms, and as a result it will be more likely for floor plans where social rooms appear far from the entrance to occur. Removing sp_y will remove the modifier that makes it more likely for eligible cells away from the entrance to be selected for private rooms, and as a result it will be more likely for floor plans where private rooms appear near the entrance to occur. Removing se_y removes the modifier that causes eligible cells to be scored according to the placement of rooms of the same type, and as a result, rooms of the same type will no longer be grouped together in the floor plan. Removing sh_y will remove the scoring pattern which implements the connection requisite, so connection requisites will no longer cause rooms to be placed adjacent to one another. Removing sd_y removes the requirement that only rooms of the social type can be placed at the cell containing the entrance door, and as a result, a room of the private or service

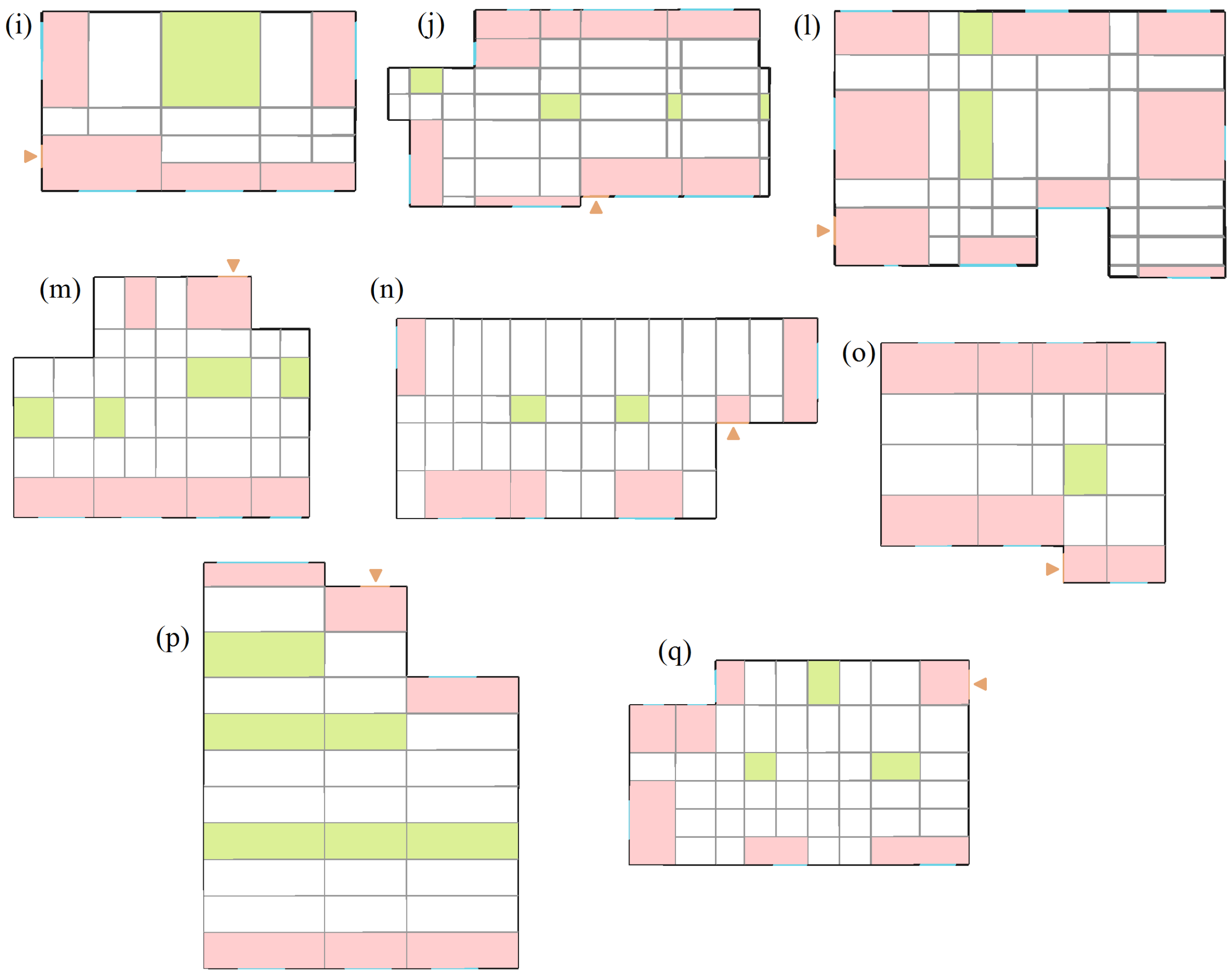

Figure 4.5: Outlines i-q used in the measurements.

type can be placed at the entrance (making the floor plan invalid). Finally, removing sk_y removes the requirement that an eligible cell containing a window must be able to fulfill at least one of the R_R's size requirements (either length or width), and as as result, an R_R requiring a large window may be placed in a cell containing a small window.

Factors can also be ablated in the expansion option score function (see Equation 3.11), modifying the probability for each expansion option to be selected. Removing kb_o removes the prohibition for rooms to expand outside the building's outline, and also the prohibition for rooms to expand beyond the border of neighboring rooms. Removing kw_o removes the modifier which makes it less likely for a room to expand in a direction if it results in adding windows beyond the first to the room. As a result, the expansion process will not discriminate cells containing windows, and this resource will be consumed more quickly. Removing kn_o removes the modifier that makes it more likely for a room to expand in a direction that has not yet fulfilled one of the dimension requisites of R_R. As a result, rooms will randomly expand in any direction, without regard for trying to achieve the room's minimum size.

Table 4.2: Measurements for the outlines seen in Figure 4.4(a-h) after a thousand execution attempts.

Case	Figure	yn	R_R	R_C	E_P	E_E	E_C	FIX	SUC	T_t (ms)
a1	4.4(a)	9	6	0	0	15	49	234	936	3062
a2	4.4(a)	9	6	1	0	32	57	247	911	3081
a3	4.4(a)	9	8	0	0	107	390	334	503	2573
a4	4.4(a)	9	8	1	37	86	403	332	474	2658
a5	4.4(a)	9	8	2	360	66	270	236	304	1926
b1	4.4(b)	11	6	0	0	263	285	268	452	2074
b2	4.4(b)	11	6	1	0	362	209	255	429	2190
b3	4.4(b)	11	8	0	0	625	300	213	75	1896
b4	4.4(b)	11	8	1	8	658	276	192	58	1848
b5	4.4(b)	11	8	2	67	601	283	186	49	1785
c1	4.4(c)	12	6	0	0	36	106	47	858	4843
c2	4.4(c)	12	6	1	5	64	113	46	818	4813
c3	4.4(c)	12	8	0	0	85	266	77	649	4533
c4	4.4(c)	12	8	1	68	175	232	51	525	4113
c5	4.4(c)	12	8	2	435	74	179	39	312	2819
d1	4.4(d)	9	6	0	0	189	220	62	591	714
d2	4.4(d)	9	6	1	25	201	298	57	476	1784
d3	4.4(d)	9	8	0	43	314	597	77	46	520
d4	4.4(d)	9	8	1	78	325	564	68	33	1575
d5	4.4(d)	9	8	2	433	191	354	23	22	378
e1	4.4(e)	10	6	0	0	197	213	143	590	2226
e2	4.4(e)	10	6	1	4	316	167	151	513	2142
e3	4.4(e)	10	8	0	0	356	585	155	59	1783
e4	4.4(e)	10	8	1	56	427	467	145	50	1733
e5	4.4(e)	10	8	2	292	322	357	114	29	1484
f1	4.4(f)	10	6	0	0	327	213	16	460	2489
f2	4.4(f)	10	6	1	7	294	252	15	447	2454
f3	4.4(f)	10	8	0	0	503	433	35	64	1912
f4	4.4(f)	10	8	1	30	408	490	24	72	1906
f5	4.4(f)	10	8	2	111	345	449	19	95	1841
g1	4.4(g)	11	6	0	0	51	67	72	882	2849
g2	4.4(g)	11	6	1	15	68	79	105	838	2797
g3	4.4(g)	11	8	0	0	80	671	152	249	1979
g4	4.4(g)	11	8	1	82	90	601	180	227	1996
g5	4.4(g)	11	8	2	281	48	510	142	161	1639
h1	4.4(h)	11	6	0	0	65	245	283	690	2503
h2	4.4(h)	11	6	1	47	49	223	293	681	2462
h3	4.4(h)	11	8	0	19	248	622	295	111	1977
h4	4.4(h)	11	8	1	76	188	601	285	135	2084
h5	4.4(h)	11	8	2	293	237	357	175	113	1745

Table 4.3: Measurements for the outlines seen in Figure 4.5(i-q) after a thousand execution attempts.

Case	Figure	yn	R_R	R_C	E_P	E_E	E_C	FIX	SUC	T_t (ms)
i1	4.4(i)	6	6	0	67	3	868	34	62	889
i2	4.4(i)	6	6	1	63	5	859	41	73	904
i3	4.4(i)	6	8	0	1000	0	0	0	0	412
i4	4.4(i)	6	8	1	1000	0	0	0	0	429
i5	4.4(i)	6	8	2	1000	0	0	0	0	443
j1	4.4(j)	13	6	0	0	354	52	27	594	916
j2	4.4(j)	13	6	1	0	381	91	54	528	2802
j3	4.4(j)	13	8	0	0	749	161	46	90	2154
j4	4.4(j)	13	8	1	4	763	169	41	64	2184
j5	4.4(j)	13	8	2	24	718	159	63	99	2247
l1	4.4(l)	11	6	0	0	9	85	1	906	3523
l2	4.4(l)	11	6	1	8	13	68	2	911	3361
l3	4.4(l)	11	8	0	0	21	203	3	776	3392
l4	4.4(l)	11	8	1	37	30	201	2	732	3309
l5	4.4(l)	11	8	2	415	18	135	2	432	2304
m1	4.4(m)	10	6	0	0	262	162	221	576	808
m2	4.4(m)	10	6	1	0	292	201	309	507	2476
m3	4.4(m)	10	8	0	0	561	333	299	106	2236
m4	4.4(m)	10	8	1	5	560	344	297	91	2244
m5	4.4(m)	10	8	2	73	521	315	279	91	2177
n1	4.4(n)	8	6	0	0	3	211	136	786	3143
n2	4.4(n)	8	6	1	55	3	364	119	578	2601
n3	4.4(n)	8	8	0	182	10	711	161	97	1747
n4	4.4(n)	8	8	1	296	7	641	128	56	1571
n5	4.4(n)	8	8	2	428	2	536	87	34	1343
o1	4.4(o)	9	6	0	0	250	188	48	562	1599
o2	4.4(o)	9	6	1	0	393	177	35	430	1371
o3	4.4(o)	9	8	0	69	705	141	7	85	1393
o4	4.4(o)	9	8	1	71	800	129	8	0	1340
o5	4.4(o)	9	8	2	386	564	50	5	0	1080
p1	4.4(p)	12	6	0	0	8	34	31	958	3314
p2	4.4(p)	12	6	1	0	16	30	53	954	3472
p3	4.4(p)	12	8	0	0	10	186	75	804	3194
p4	4.4(p)	12	8	1	1	24	113	62	862	3333
p5	4.4(p)	12	8	2	110	28	73	52	789	3099
q1	4.4(q)	10	6	0	0	265	110	125	625	3092
q2	4.4(q)	10	6	1	54	380	67	75	499	2811
q3	4.4(q)	10	8	0	0	637	127	191	236	2668
q4	4.4(q)	10	8	1	28	660	118	168	194	2641
q5	4.4(q)	10	8	2	82	596	123	177	199	2524

5. CONCLUSION

In this document we have presented a method for growth-based procedural floor plan generation which considers a building's exterior features as well as user-provided requisites. The method expands on the idea presented by Lopes et al. [10] in several ways. First, by using an irregular grid for room placement and expansion which allows rooms to snap to the features of the exterior walls. Second, by considering window constraints during the floor plan generation. And finally, by using a different room expansion algorithm which allows user-provided room requisites and finer control over the resulting floor plan.

There is an ample body of research on architectural theory, but such work is often formulated in general terms and open to interpretation, and therefore difficult to apply to the problem of procedural floor plan generation. Merrell et al. [13] avoid this pitfall by using machine learning and subsequently inferring an architectural plan, in the form of a connection tree, which is then transformed into a floor plan with stochastic optimization. Due to the constraint created by the external walls, our algorithm could not similarly use a connection tree. Our solution is the proposed room placement algorithm, which keeps rooms close together without enforcing connectivity.

The procedural generation algorithm guarantees that every valid floor plan fulfills user-provided requisites R_R and R_C, but valid floor plans are not necessarily considered adequate by all users. For example, in Fig. 5.1(a), a corridor runs along the exterior wall of the building. In (b) a U-shape corridor and very small bathroom can be seen. In (c) two circulation areas connect the living room to different sets of bedrooms. In (d), a small kitchen is beside a large dining room. Such evaluations are subjective and vary from person to person.

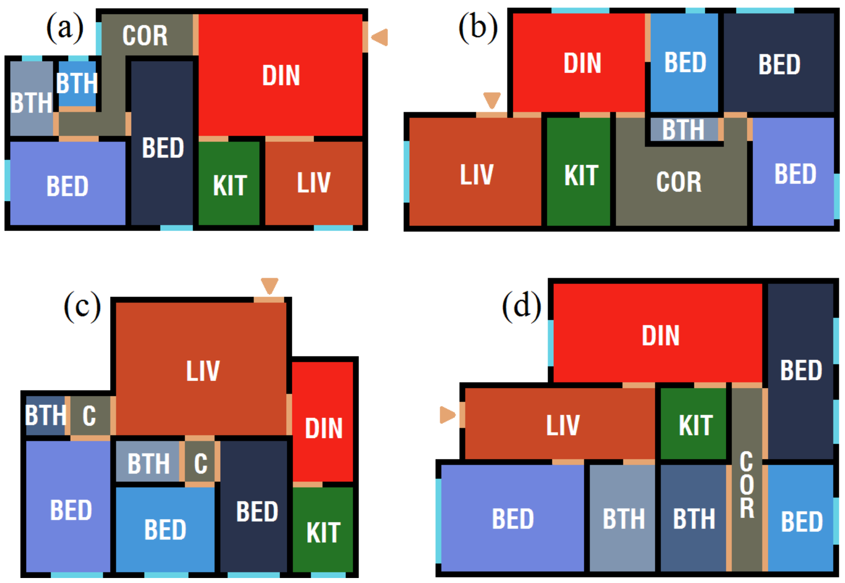

Figure 5.1: Floor plans which are valid, but might be considered unrealistic.

A large part of the experience of an interior space is how it is furnished and decorated. To improve the quality of the floor plans generated by our method, it could be integrated with an algorithm for the automatic placement of furniture (e.g., Merrell et al. [14]). Regarding performance, our algorithm seems appropriate for real-time applications. The low run time (see Tables 4.2 and 4.3) means that the algorithm could be executed during a virtual-world's walk through, generating floor plans for buildings which need to be entered by the user in real-time.

For future work, we intend to expanded our model to handle multiple-floor residences, modifying the concept of an entrance door to the concept of a floor entrance, instead. Second, we intend to expand the model to accept external walls with arbitrary shapes, for example, diagonal and curve walls. This would require a modification of the room expansion algorithm, to allow rooms to grow along different directions, and not only along the UP/DOWN and LEFT/RIGHT dimensions. Another improvement would be a post-processing step, to adjust the exact final position of each wall, enabling finer adjustments than our grid allows. Finally, our algorithm would benefit from an explicit corridor placement step, to enforce room connectivity.

REFERENCES

[1] Bruls, M., Huizing, K., and Van Wijk, J. J. Squarified treemaps. In *Data Visualization 2000*. Springer, 2000, pp. 33–42.

[2] Charman, P. Solving space planning problems using constraint technology. *Nato ASI Constraint Programming: Students' Presentations, TR CS 57*, 93 (1993), 80–96.

[3] Felkel, P., and Obdrzalek, S. Straight skeleton implementation. In *Proceedings of spring conference on computer graphics* (1998), Citeseer.

[4] Germer, T., and Schwarz, M. Procedural arrangement of furniture for real-time walkthroughs. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 2068–2078.

[5] Greuter, S., Parker, J., Stewart, N., and Leach, G. Real-time procedural generation ofpseudo infinite'cities. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2003), ACM, pp. 87–ff.

[6] Hahn, E., Bose, P., and Whitehead, A. Persistent realtime building interior generation. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames* (2006), ACM, pp. 179–186.

[7] Hendrikx, M., Meijer, S., Van Der Velden, J., and Iosup, A. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 9*, 1 (2013), 1.

[8] Kelly, G., and McCabe, H. Citygen: An interactive system for procedural city generation. In *Fifth International Conference on Game Design and Technology* (2007), pp. 8–16.

[9] Liu, H., Yang, Y.-L., AlHalawani, S., and Mitra, N. J. Constraint-aware interior layout exploration for pre-cast concrete-based buildings. *The Visual Computer 29*, 6-8 (2013), 663–673.

[10] Lopes, R., Tutenel, T., Smelik, R. M., de Kraker, K. J., and Bidarra, R. A constrained growth method for procedural floor plan generation. In *Proceedings of GAME-ON 2010, the 11th International Conference on Intelligent Games and Simulation* (2010).

[11] Marson, F., and Musse, S. R. Automatic real-time generation of floor plans based on squarified treemaps algorithm. *International Journal of Computer Games Technology 2010* (2010), 7.

[12] Martin, J. Procedural house generation: A method for dynamically generating floor plans. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (2006), pp. 1–2.

[13] Merrell, P., Schkufza, E., and Koltun, V. Computer-generated residential building layouts. *ACM Transactions on Graphics (TOG) 29*, 6 (2010), 181.

[14] Merrell, P., Schkufza, E., Li, Z., Agrawala, M., and Koltun, V. Interactive furniture layout using interior design guidelines. In *ACM Transactions on Graphics (TOG)* (2011), vol. 30, ACM, p. 87.

[15] Michalek, J., Choudhary, R., and Papalambros, P. Architectural layout design optimization. *Engineering optimization 34*, 5 (2002), 461–484.

[16] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. *Procedural modeling of buildings*, vol. 25. ACM, 2006.

[17] Müller, P., Zeng, G., Wonka, P., and Van Gool, L. Image-based procedural modeling of facades. *ACM Trans. Graph. 26*, 3 (2007), 85.

[18] Noel, J., and North, S. Dynamic building plan generation. *Bachelor thesis, The University of Sheffield* (2003).

[19] Parish, Y. I., and Müller, P. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 301–308.

[20] Peng, C.-H., Yang, Y.-L., and Wonka, P. Computing layouts with deformable templates. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 99.

[21] Peponis, J., Wineman, J., Rashid, M., and Bafna, S. On the description of shape and spatial configuration inside buildings: convex partitions and their local properties. *Environment and Planning B 24* (1997), 761–782.

[22] Press, W. H. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[23] Rinde, L., and Dahl, A. Procedural generation of indoor environments. *Master's Thesis, Chalmers University of Technology* (2008).

[24] Rodrigues, E., Gaspar, A. R., and Gomes, Á. Automated approach for design generation and thermal assessment of alternative floor plans. *Energy and Buildings 81* (2014), 170–181.

[25] Smelik, R. M., Tutenel, T., Bidarra, R., and Benes, B. A survey on procedural modelling for virtual worlds. In *Computer Graphics Forum* (2014), Wiley Online Library.

[26] Tutenel, T., Bidarra, R., Smelik, R. M., and de Kraker, K. J. Rule-based layout solving and its application to procedural interior generation. In *CASA Workshop on 3D Advanced Media In Gaming And Simulation* (2009).

[27] Tutenel, T., Bidarra, R., Smelik, R. M., and Kraker, K. J. D. The role of semantics in games and simulations. *Computers in Entertainment (CIE) 6*, 4 (2008), 57.

[28] Watson, B., Muller, P., Wonka, P., Sexton, C., Veryovka, O., and Fuller, A. Procedural urban modeling in practice. *Computer Graphics and Applications, IEEE 28*, 3 (2008), 18–26.

[29] Weber, B., Müller, P., Wonka, P., and Gross, M. Interactive geometric simulation of 4d cities. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 481–492.

[30] Whitehead, B., and Eldars, M. The planning of single-storey layouts. *Building Science 1*, 2 (1965), 127–139.

[31] Yin, X., Wonka, P., and Razdan, A. Generating 3d building models from architectural drawings: A survey. *IEEE Comput. Graph. Appl. 29*, 1 (Jan. 2009), 20–30.

[32] Zmugg, R., Thaller, W., Krispel, U., Edelsbrunner, J., Havemann, S., and Fellner, D. W. Procedural architecture using deformation-aware split grammars. *The Visual Computer 30*, 9 (2014), 1009–1019.

A. PUBLICATIONS

The following paper has been submitted and published as a result of this work:

- Camozzato, D., Dihl, L., Silveira, I., Marson, F. and Musse, S. Procedural floor plan generation from building sketches. The Visual Computer (2015), vol. 31, pp. 753-763, doi: 10.1007/s00371-015-1102-2

B. CONFIGURATION FILES

The method presented in this document requires that the following files and settings be provided as an input: a building's outline, a list of room requisites R_R and connection requisites R_C, the wall length threshold t_W and the threshold distance t_D. In the following sections each different input is detailed.

B.1 Building's outline

To create a floor plan, the method requires the outline of the building for which a floor plan should be created. This outline is a non-convex polygon, and each line in the polygon is additionally identified as either a wall, a window or a door.

In the prototype implementation three files are used to inform the bulding's outline. The first file contains x and y coodinates for the vertices of the polygon:

```
20         // Number of vertices in the polygon
0 0        // x0 y0
0.7 0      // x1 y1
1.35 0     // And so on
2 0
2 -0.1
2 -3.35
5.25 -3.35
5.25 -2.15
5.25 -2.05
6.5 -2.05
7.5 -2.05
9.35 -2.05
10.35 -2.05
11.15 -2.05
11.15 2.45
11.15 3.45
11.15 4.5
1 4.5
0.2 4.5
0 4.5
```

The second file contains indexes for lines in the polygon which are windows:

```
7         // Number of windows
1         // Index for a line which is a window
4
5
6
9
11
14
```

The last file contains indexes for lines in the polygon which are doors:

```
1         // Number of doors
17        // Index for a line which is a door
```

Any line which is not defined as a window or a door is considered a wall.

B.2 Room and connection requisites

To inform the procedural generator which rooms must exist and how they must be connected in the floor plan, a list of requisites must be provided. A room is placed and expanded for each room requisite R_R. The placement of rooms within the building depends on different rules (see section 3.3), among which is the hard connection rule, which forces adjacency between rooms. The hard connection rule applies only for each pair of rooms defined in a connection requisite. An example list of requisites is presented below:

```
<Requisites>
    <RoomRequisites>
        <Room name="BEDR1" type="Private" priority="10">
            <Width min="2.0" max="5.0" />
            <Depth min="2.0" max="5.0" />
            <Area min ="6.0" max="20.0" />
            <Window required="yes" />
        </Room>
        <Room name="BEDR2" type="Private" priority="7">
            <Width min="2.0" max="4.0" />
            <Depth min="2.0" max="4.0" />
            <Area min ="6.0" max="20.0" />
            <Window required="yes" />
        </Room>
        <Room name="DIN1" type="Social" priority="12">
            <Width min="3.0" max="5.0" />
            <Depth min="4.0" max="6.0" />
            <Area min ="10.0" max="25.0" />
            <Window required="no" />
        </Room>
        <Room name="KITC1" type="Service" priority="5">
            <Width min="1.5" max="3.5" />
            <Depth min="1.5" max="3.5" />
            <Area min ="4.0" max="10.0" />
            <Window required="no" />
        </Room>
        <Room name="LIVN1" type="Social" priority="6">
            <Width min="3.0" max="6.0" />
            <Depth min="3.0" max="6.0" />
            <Area min ="9.0" max="25.0" />
            <Window required="yes" />
        </Room>
        <Room name="BATH1" type="Private" priority="3">
            <Width min="0.8" max="2.0" />
            <Depth min="0.8" max="2.0" />
            <Area min ="2.0" max="4.0" />
            <Window required="no" />
        </Room>
        <Room name="BATH2" type="Private" priority="7">
            <Width min="1.3" max="3.0" />
            <Depth min="1.3" max="3.0" />
            <Area min ="2.5" max="5.0" />
            <Window required="no" />
        </Room>
    </RoomRequisites>
    <ConnectionRequisites>
        <Connection name="BATH2,BEDR1" type="Door">
            <Link a="BEDR1" b="BATH2" />
        </Connection>
        <Connection name="DIN1,KITC1" type="OpenWall">
            <Link a="DIN1" b="KITC1" />
        </Connection>
    </ConnectionRequisite>
</Requisites>
```

B.3 Wall length threshold t_W and distance threshold t_D

There are two variables which must be set to configure the algorithm. The first variable is the wall length threshold t_W, which is used during the grid generation step (see section 3.2). Each wall which is longer than t_W generates a new internal line perpendicular to its midpoint. This process is performed recursively, until no wall is longer than t_W. This effectively limits the length of cells in the grid.

The second variable is the distance threshold t_D, which is used to identify which cells in the grid can become eligible cells. To become an eligible cell, a cell must be at least t_D meters away from any other eligible cell. Cells are selected recursively to populate the interior of the building with eligible cells.