MSc Final Project Thesis
(COMPGV99: Individual Project)

Interiors Automatic Furnishing for Brighton Terrace Houses

— Part of the Brighton Street Cultural Heritage Project

Shanyue Huang

(ID 13051427)

Email: shanyue.huang.13@ucl.ac.uk

Programme: (CGVI) Computer Graphics, Vision & Imaging

Supervisors: Dr Simon Julier & Dr William Steptoe

5 September, 2014

Abstract

This project implemented an automatic furnishing programme. The objective of it is to furnish Brighton Houses is terrace style. The programme supports the first tier furniture populating and arrangement with a various room types and furniture types. The core technique of this project is to using Metropolis-Hastings and Simulated Annealing to optimise the current state to the global best.

Based open projects: Sweet Home 3D

Keywords: Metropolis-Hastings, Simulated Annealing, procedural modelling, automatic furnishing

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Description and Motivation

The group project "Augmented Reality in Cultural Heritage for Brighton Street" is to develop an app on portable device, which provides a window (the screen) to watch the Brighton Street through the time dimension of the history. The user can watch through the walls of the residents' houses, face the interior environments and the stories which are happening in that past time. It will provide different styles and stories in different decades by swiping a time scrolling bar as is shown in Figure 1.1. We wish the tourist could think it compelling, and has a vivid experience on the history of Brighton Street.

Figure 1.1: A diagram describing the idea of the project "Augmented Reality in Cultural Heritage for Brighton Street" (taken from [1])

The thesis behind the project is that telling history through stories and narratives is more engaging. To do this, we need both people and places. The places will be the houses in streets. We don't have actual models of the interiors of houses. This is both for privacy reasons, but also because it was in the past, which had been hard to be reconstructed exactly. As a result, we are developing new approaches to automatically generate plausible content of the building interior. So far we have the information about tentative floor plans, time periods, potential dictionaries of objects. Therefore, the main problem is to focus on unknown furniture arrangement with known floor plans.

There are three main motivations of implementing this project by the thought of procedural modelling. Firstly, it "Saves designers' time without losing perceived realism" [2]. As there are about 40 houses in that street and we expect to show in different time and seasons, it would be a huge work and take long if the developer modelled by hand. Secondly, it generates various scenes with less file size of the resource, as it decomposes models and re-combines them to reach the diversity. Lastly, it is readily extensible. The achievement would be flexible to reuse for other applications. For example, when adding a new style or changing the room size or shape, it only needs to do some trimming on related rules.

1.2 Objectives and Difficulties

1.2.1 External Objectives and Difficulties

This project goal is to fully automated furnish rooms in house floor plan without prior supervised configurations. The interiors were expected to restore the living arrangements in early ages, making users to understand the living, cooking, bedding arrangements at those ages. If developers exactly reconstructed these scene in old ages, it would cost unpredicted time and budgets. Ideal is attractive, but reality has to start from down-to-earth work.

To abstract the problem, it is an optimisation problem which attempts to reach a global best state by a series of randomisation. Although this was the core technique behind this project, there were a bunch of trivial tasks that must be done for building the environment of the problem. As it was a practical problem, the parameters were different if the scope of the input environment was different. So some non-core tasks have to be done firstly.

Firstly, there was no handy floor plan models for the Brighton terrace style houses. We [1] visited three houses with the residents permissions. Figure 1.2 shows the street (Kensington Plance) where the three houses locate. With a general concept on the house floor plan, a ground floor plan was drawn using Sweet Home 3D quickly and manually, and started the core algorithm with targeted domain only on distance between same sized cubes.

Figure 1.2: A map of Kensington Plance, Brighton in 1896

After the algorithm could stably reach the global best for all testings, I started to improve the environment, extending the targeted domain with more considerations and trying to use actual 3D furniture models. Because the programme should not be limited on a fixed known floor plan, the floor plan information should be understood by the programme itself, instead of setting them every time manually.

That pulled the heavy curtain of seeking a solution which could solve the problem and steer by the obstacles Unity set potentially. The progress of the project cycled repeatedly and went forward little by little. Along the target function being improved and complex, the number of cycles increased but the global best value becomes harder to discover. It took a long time to finish a furnishing testing, but the fact is that there is no theoretical values for the built environment; the regularity of all weightings and cooling schedule parameters had to discover from

[1] Dr. Simon Julier (the supervisor), David Pribil (who was in charge of building the back-end system of the AR app), Phil Blume (one of the Brighton local developers in charge of story contents) and me.

times of testings. Simplifying the target would be left too far away from the ideal objective, while sticking on the initial plan would be impossible to complete the project in time. That was the pain from ideal to reality.

1.2.2 Internal Objectives and Limitations

The internal objectives are absorbing the problems appearing in the process of the project, forming a system of procedures on how to solve it next time, converting the first tries to experience.

In particular, although this was my second project on Unity3D, these two projects were in totally different aspects. As the scope of the project was rarely covered in Unity3D projects, some common solutions of the core algorithms were not feasible for Unity3D projects. For example, Unity calculates the angle of two vectors is "**never greater than** 180°" [3], which means you could never simply get the exact angle between two vectors, you only can directly get angle between the lines where vectors are on. And this problem was hardly found, when the furniture was cubes. As the programme went complicated, and very randomised, an unexpected result becomes extremely hard to debug. Finally, when I positioned on it and realised the limitation of this Unity function, there were so many functions depending on this calculation, the wall normals which pointing to the room centre, the angle between the furniture facing direction and wall normals, the rotated degree control... And there is no way to control objects facing directions in vector, Unity3D only supports transform rotation in Euler angles. You have to do conversions between 3D Euler angles to 3D vectors yourself... There were amounts of problems on Unity's gameobject angle handling that could be never imagined before this project.

Angles were not the most painful problem, which could be fixed little by little. The most painful problem was it is really impossible to get the corners of all model oriented bounding box. Finally, I thought I fixed this problem by filtering all vertices in the object combined meshes, but in testings, some models were exactly right, while some models were obviously wrong. None of my models were imported without any scaling, so the vertices position should be in the world coordinates instead of local coordinates. So this problem could only be solved by giving up that model, once it was discovered.

Speaking of gains, this project did make me impressed with the enchantment

of randomisation.

1.3 Organisation of the Thesis

This thesis describes the design and implementation of the automatic furnishing system. Next chapter gives a brief introduction on related background concepts, follows tutorials on background algorithm theories. At the end of Chapter 2, it detailedly analyses two projects with strong relativity to this project, and overviews on other related work. Chapter 3 presents different levels of project frameworks and the reasons of why they were designed as them. Chapter 4 demonstrates the implementation of the system in details according to the system design. Then a series of testing results are presented together with analysis and comments on them in Chapter 5. Finally, Chapter 6 talks about the project evaluation and proposed further work.

Chapter 2

Background

This Chapter gives general introductions on related background concepts, and a tutorial on basic Metropolis-Hastings algorithm and simulated annealing algorithm. Finally, related projects based on these concepts and theories are referred, and two of them that have strong relations to this project are demonstrated and analysed in detail. Finally, open source software is introduced for the project floor plan set up.

2.1 Background Concepts

The concepts are introduced here for their importance to most of the related works presented. Whatever the methods were used, they all involved in the thoughts behind these concepts.

2.1.1 Procedural Modelling

Procedural modelling is to generate a bigger environment with smaller elements of models automatically. [2] It can apply in computer-generated cities, building exteriors, building layouts, building interiors, even trees and clouds, as the example procedurally generated models in Figure 2.1. There are a number of methods for implementing procedural modelling, and we can classify them as probabilistic methods and non-probabilistic methods. The probabilistic methods include trans-dimensional Markov chain Monte Carlo [4], Bayesian Ontology and genetic

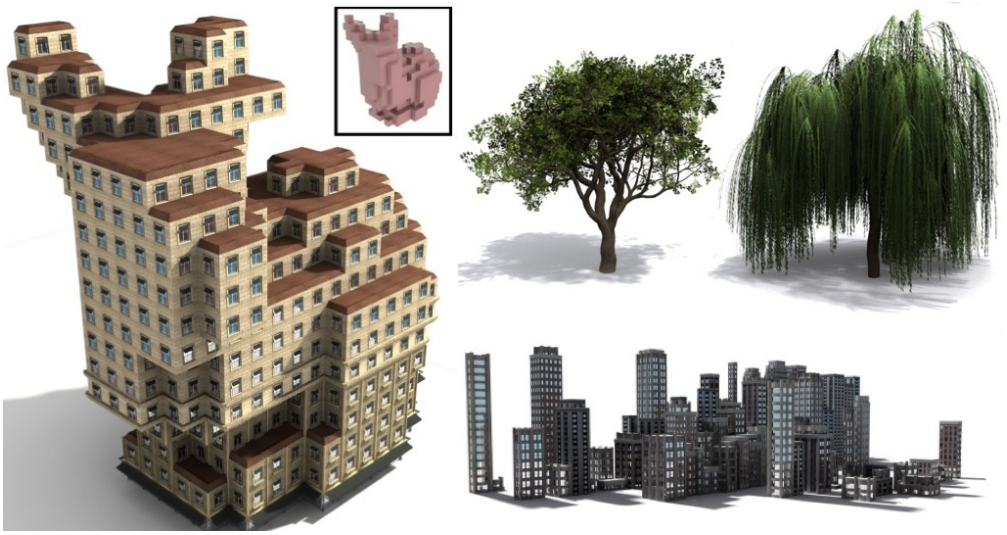

Figure 2.1: The example of procedurally generated models (taken from [4])

programming. The non-probabilistic methods include the rule-based [2], stochastic optimisation [5] [6] and [7], the agent-based [8], and the grammar-based [4]. Most of the application will be introduced in a general scope at the beginning of the Related Work Section; some of them that have strong relations to this project will be given a detailed review in that section.

2.1.2 Ontology

The word "ontology" comes from philosophy on the problem of existence, but here is employed in its computer science field. Ontology can be understood as a "semantic constraints model", whose primitives are classes, attributes, relationships and logical constraints. [9] They are typically specified in languages.

Ontology thoughts are also applied in Artificial Intelligence (AI). In multi-agent system which was employed in the first important related work in Section 2.3.2, it is as an interface specification that specifies a language for speaking to the agent. Ontology is used to build databases, and appears in dialogue systems.

Ontology gives a logical hierarchical relationship between objects in a room, while **Oriented Bounding Box**(see next subsection) helps to determine the spatial relationship and orientation between related objects.

2.1.3 OBB: Oriented Bounding Box

Typically a primitive model from some modelling software is in some default alignment and scaling. In order to apply the transform to fit it in targeted scene, a transformation needs to be specified applying to the model. As the model can be various and complex, in order to simply define its position and especially its orientation, it is necessary to put it into a 6-face box, which is the Oriented Bounding Box (OBB) of the model. Figure 2.2 shows a chair in a red line box, which is its OBB; the green arrows indicates the spatial relations to other adjacent objects.

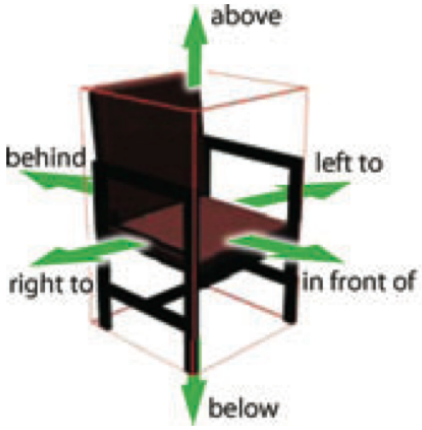

Figure 2.2: The example of a chair in its Oriented Bounding Box (OBB) (taken from [8])

2.2 Related Theories

Automatic furnishing has randomised input and complicated nonlinear constraints to optimise it. This project employed the thoughts of Metropolis-Hastings Algorithm and Simulated Annealing Algorithm to solve the optimisation problem.

2.2.1 Metropolis-Hastings Algorithm for MCMC

Metropolis firstly took advantage of Markov Chain Monte Carlo (MCMC) methodologies to solve the properties of a system with multi-dimensional constraints referring to the Boltzmann distribution [10] in 1953. In 1970, Hastings extended Metropolis' algorithm to sample from more randomly general function [11] and generalised the algorithm as known of "Metropolis-Hastings (MH) Algorithm".

The basic MH algorithm is shown in Algorithm 1. Given a state space χ where holds all x states, and a non-negative function which is the proposal distribution $q(x^*|x_i)$. Initialise x_0 with some random guess. In the MH loop, sample u in the Uniform distribution between 0 and 1, and do a random walk starting with whatever current x_i is at, say after the random walk the state is at x^*.

If

$$\frac{q(x_i|x^*)}{q(x^*|x_i)} > 1, \tag{2.1}$$

when the probability of proposal state x^* larger than current state x_i, which means

$$\frac{p(x^*)}{p(x_i)} > 1, \tag{2.2}$$

the value u sampled from $U_{[0,1]}$ will be always smaller than $\min\left\{1, \frac{p(x^*)q(x_i|x^*)}{p(x_i)q(x^*|x_i)}\right\}$, and this x_i to x^* jump will always be accepted; otherwise, when

$$\frac{p(x^*)}{p(x_i)} < 1, \tag{2.3}$$

the decision of whether jumping to x^* depends on how the proposal distribution function $q(x^*|x_i)$ is defined, and the picked random value u.

If based on the proposal distribution function $q(x^*|x_i)$, s.t.

$$\frac{q(x_i|x^*)}{q(x^*|x_i)} < 1, \tag{2.4}$$

when

$$\frac{p(x^*)}{p(x_i)} < 1, \tag{2.5}$$

this jump will be never accepted. Similarly, if

$$\frac{p(x^*)}{p(x_i)} > 1 \tag{2.6}$$

the decision of whether accepting this jump depends on the chosen $q(x^*|x_i)$ and then another random threshold u. Metropolis-Hastings is a randomised algorithm. The purpose of it is to make sure that the current state x_i does not settle on a local minimum.

Therefore, the proposal distribution function should be chosen carefully and experimentally. A bad proposal distribution function might lead to the convergence of final states. Tierney [12] pointed out that if the target domain is set too narrow, this algorithm will produce a geometrically converged result, whatever the initial state x_0 was at. For example, we just targeted on points' distance maximisation in limited space, the result must converge to uniform distribution filled the whole space, which is predictable. However, in the automatic furnishing case, most of the rooms were not wished to be furnished in a similar distribution. Various and plausible distributions are expected. Otherwise, the amount of calculations and annealing cycling only return a bunch of unitary results that have been exactly known before running the MH algorithm. In practice, each specific testing case is expected to numerically converge to its global best; but with the converged cost or score, we expect the results appear to be different geometrically. Hence, the design of the proposal marking system should be complicated in considerations for the space and the point attributes, as the furniture is not a particle, and also with reasonable ratios to balance the different considerations.

Additionally, $p(\cdot)$ does not have to be the entire Bayesian probability expression, which means the sum of all "probabilities" do not have to be one. It is only required to be proportional to the real probability, since the denominator in the Bayesian probability expression will cancel out in decision operation. This point is meaningful, which could reduce the amount of calculations in every iteration.

Algorithm 1 The basic Metropolis-Hastings algorithm

```
Initialise x_0
for i=0:(N-1) do
    Sample u ~ U_[0,1]
    Sample x* ~ q(x*|x_i)
    if u < min{1, p(x*)q(x_i|x*) / p(x_i)q(x*|x_i)} then
        x_{i+1} = x*
    else
        x_{i+1} = x_i
```

Differently with MH algorithm, Metropolis Algorithm [10] assumes a symmetric random walk for the proposal distribution, which means $q(x_i|x^*) = q(x^*|x_i)$. In other words, the decision for accepting a jump becomes $u < \min\left\{1, \frac{p(x^*)}{p(x_i)}\right\}$. Consequently, the probability of the jump between two points only depends on their independent probabilities; if $p(x^*) > p(x_i)$, the jump will be always accepted.

More details on applying MH algorithm to this project will be shown in the

Implementation Chapter, including how Metropolis-Hasting algorithm helps this project, and the design for proposal distribution functions effecting on different types of furniture.

2.2.2 Simulated Annealing

Annealing is used in steel or other metals processing. When the metal cools slowly from a very high heated temperature, the crystal structure trends to form in a more organised and stable way. Cooling slowly and over several cycles allow the crystals to form uniformly. Simulated annealing in computing technology works in the same theory.

There are three variables that must be passed by the simulated annealing algorithm. The starting temperature and ending temperature are just like the temperature in real world, which refers to excitement level of the molecules in a substance. Figure 2.3 shows the flowchart of Simulated Annealing algorithm. The algorithm controls the number of cycles between the starting temperature and the ending temperature. The more cycles, the more slowly that algorithm is cooled down and so the solution found is better, similarly to the real world annealing. However, the more cycles, the longer the algorithm running time is.

The temperature in the simulated annealing process refers to the amount of randomisation, and the randomisation also affect this algorithm to seek the global minimum or maximum. [13] In consequence, different problems should have their custom randomisation process.

For this project, the role of Metropolis-Hasting algorithm is to try to do an ergodic process under a same temperature, and the role of Simulated Annealing algorithm is to control the speed in cooling down, and the acceptable rate of each temperature state. Practically, the good temperature setting will enable enough diversity of samples in the beginning; and numerically converge to a stable global optimal in the end.

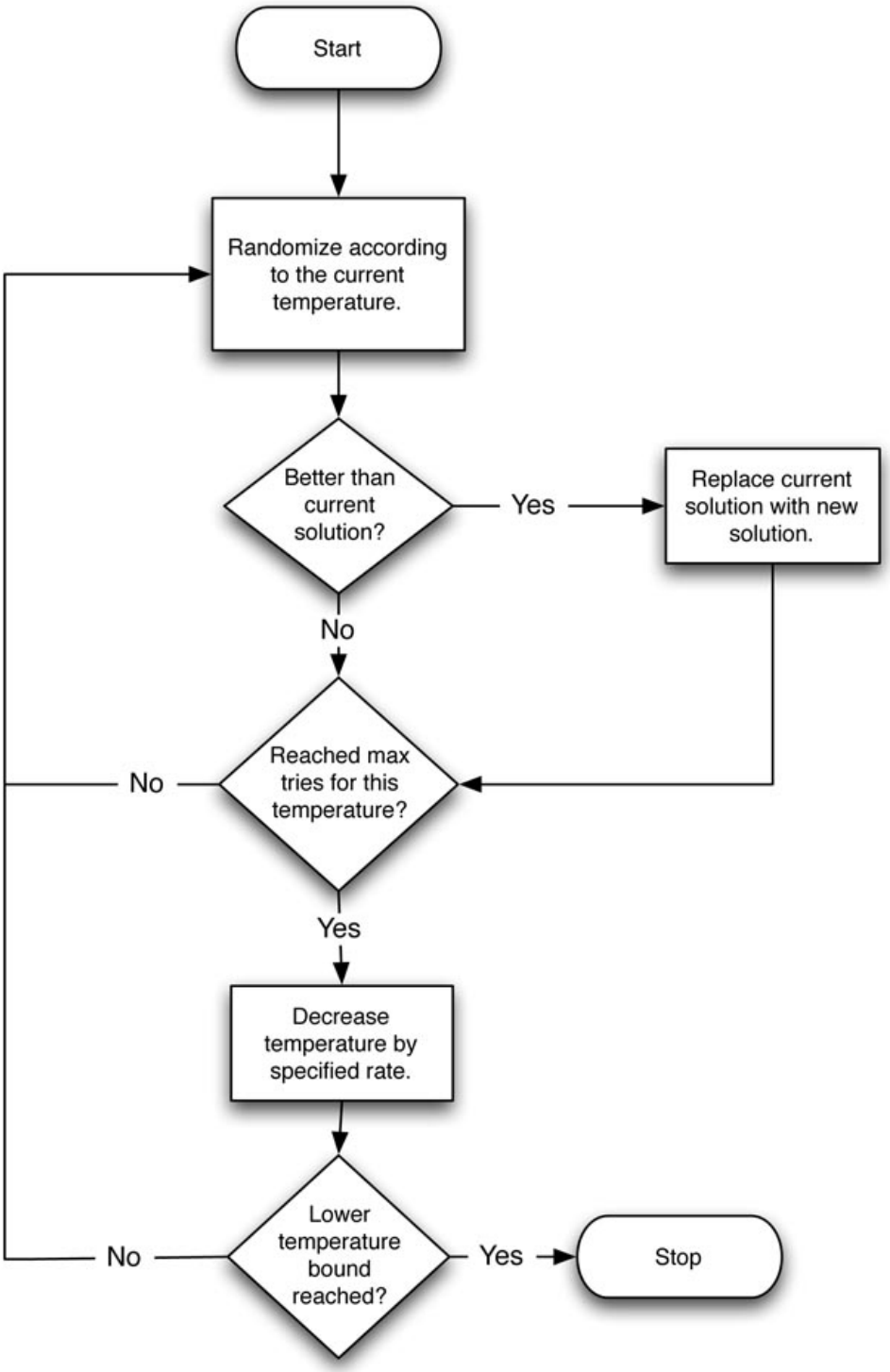

Figure 2.3: Simulated Annealing algorithm flowchart (taken from [13])

2.3 Related Work

2.3.1 Projects Based on Procedural Generation in General Scope

Procedural generation has been applied in many fields:

1. **Biological systems (e.g. L-systems, trees, and clouds)**:

   Talton et. al [4] present powerful means for generating different kinds of trees given a grammar and a high-level specification, such as geometric shapes and analytical objectives. Their method are based on Markov chain Monte Carlo, context-free grammars and geometry synthesis.

2. **Designs of cities**:

   Parish and Muller [14] propose a system to procedurally generate cities with the inputs, such as river boundaries and population density. They employed the extended L-systems which generate geometry and are constrained by both global and local rules, and a texturing system with texture elements.

3. **Designs of exteriors of buildings**:

   The work of Talton et. al [4] is also adaptable for automatically generate building exteriors, given a geometry shape for one building or a cluster of buildings.

4. **Floor plans**:

   A project about procedurally generating a house floor plan with corresponding house exterior is achieved by Merrel et al. [6]. The system uses Bayesian network and is trained on real-world data, then constrained by a set of high-level requirements obtaining the floor plan through stochastic optimisation. Then 3D house exterior is constructed from the floor plan.

5. **Furniture arrangement in buildings**:

   Tutenel et al. [2] propose a rule-based layout solver, solving the layouts of floor plans and room furniture design. Each object is defined in both explicit way (about its neighbour constraints) and implicit way (about its functionality). They also give a definition on "object features", which are 3D geometries with a tag. One object can have multiple tags, and one object

could follow area-based rules (looking for a place next to a object feature or an object) or follow grid-based rules (adaptive grids for objects: e.g. bigger object with bigger grids). Their work is mainly on semantic rule design, which is very sophisticated and easily produces deadlocks.

The scope of this project is the final one, **furniture arrangement in buildings**. The following two projects are very targeted and with great performance in different aspects.

2.3.2 Furniture Arrangement in Real-Time Walkthroughs

Their Features, Considerations and Contributions:

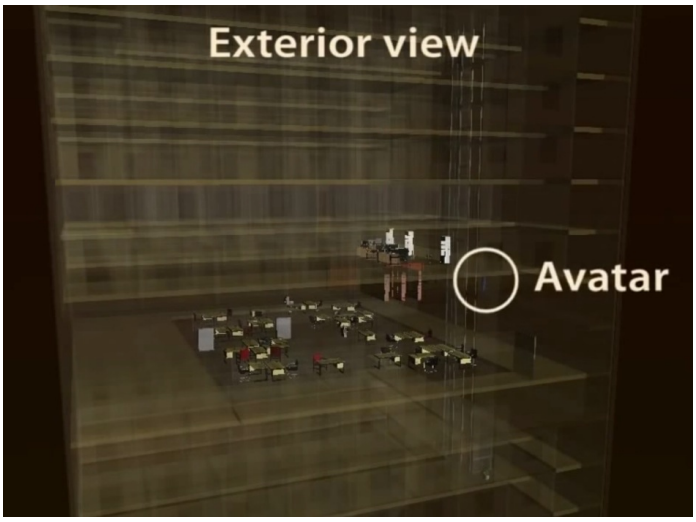

Figure 2.4: Real-time procedurally furnishing example (taken from [8]'s video show [2]; if you click the link in footnote, you can observe as the avatar position changes, the furniture in different rooms appears and disappears.)

Germer and Schwarz [8] implement procedural arrangement of furniture for real-time walkthroughs, which is to only furnish the rooms near where the viewer is at. Figure 2.4 shows an avatar in the lift and about to visit some floor, as a result, only the rooms near the avatar current spot are furnished.

As they furnishing the environment in real-time walkthroughs, the arrangement persistence problem is under their consideration, which means the room should be unaltered if the avatar re-enters the room he has just left, even it should keep the

[2]Germer and Schwarz (2009) [8] video site: `http://youtu.be/xwVUknGeycQ?t=1m31s`

changes made by the avatar. In fact they solved this problem by a deterministic random number generator based on each room's position number and storing the changed displacement as an "offset" of the interior movement. In order to make the rooms more plausible and various, it provides "styles", such as the purpose of the room (e.g. hotel, library, office and restaurant), the style of furniture (e.g. old, plain, modern, abundant), even untidiness with orientation angles and distance from its parent.

This work extended the state of the art in three ways:

1. The arrangement is fast and flexible;
2. A multi-agent system paradigm is built for generating object layouts;
3. It is furnishing in real-time based on the walkthroughs.

Their Methodology:

Germer and Schwarz [8] employ multi-agent system to pick and arrange objects with a bottom-up strategy. Single pieces of furniture are regarded as autonomous agents, which are moving around within a predefined room until they reach the semantic description properly.

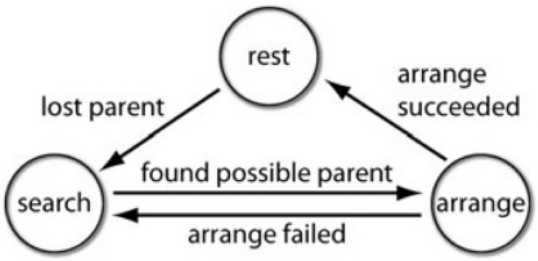

Figure 2.5: The three states of agent behaviour (taken from [8])

As is shown in Figure 2.5, there are three states of their agent behaviour: search, arrange and rest:

1. **Search**

   "Search" is the initial state of all agents. An agent searches all its parents [3] currently in room and appears beside the parent with enough space left, then changes its state into "arrange". If there is no parent found, this agent is deleted in this room.

[3]The parent-children relationship is defined by ontology thought as introduced in last chapter.

2. **Arrange**

   Agents in "arrange" state try to place themselves to the chosen parent on a suitable side. The suitable side is found by trying the spatial relations like the green arrows in Figure 2.2 one by one. In that process, if there is no collision with other agents, it can reach the state of "rest". If it tries all six sides but has still not been allocated, it will change its state back to "search".

3. **Rest**

   Once an agent reaches "rest" state, other agents can become its children. If an agent in "rest" state losing its parent (for instance its parent is removed), it will change its state into "search". However, Germer and Schwarz [8] also said they found most rooms worked better without this feature.

Their Limitations:

In the persistence problem, Germer and Schwarz [8]'s system does not support adding or removing objects at run time interactively with the player. However, adding or removing objects is not required in my project.

Figure 2.6: A failure case of not supporting complex functional dependencies (taken from [8])

They said their approach does not support complex functional dependencies in order to generate efficiently. For example, the TV set should be placed beside the wall and opposite to the sofa in Figure 2.6. Additionally, their system does preserve the space between two independent interiors, but has problems on finding the optimal layout for very dense scenes. In Figure 2.6 accessibility in two area is blocked by the table and the couch.

Figure 2.7 shows the OBB of a round table leading to a rectangular-like gap

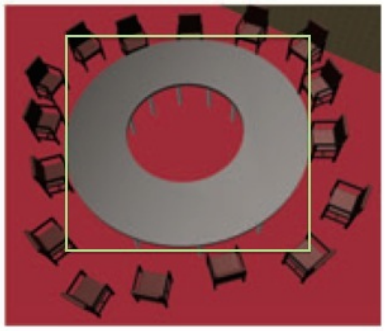

Figure 2.7: OBB of a round table causing failure arrangement around it(taken from [8])

which marked in yellow rectangle between the table and its children chairs. (Because of the rotations of the surrounding chairs, the artifact is not so obvious.)

As their system working in real-time walkthroughs, there is a certain maximum walking speed depending on the system performance. For their testing device, a 3GHz PC, it takes less than 1 second to furnish one room with 20-30 objects, while it takes up to 4s to furnish a big room with 300 objects in grid layout (the layout with similarity between grids) Thus the furnished rooms will not able to be furnished as many as needed if the player walks too fast, they said. [8]

The performance of Germer and Schwarz's project is impressive, but considering the short time limitation of my project, it was not suitable to build the multi-agent system and give semantic descriptions to all objects including a wall in pieces, as it shown in Figure 2.8.

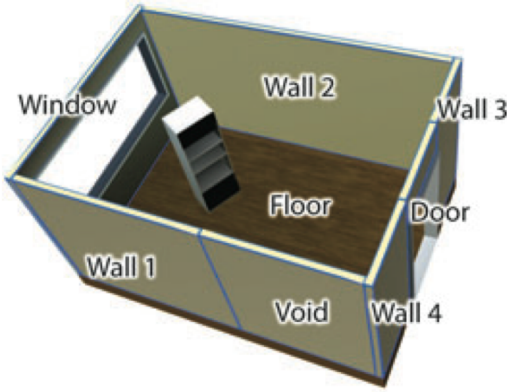

Figure 2.8: "Room components must have suitable descriptions to be found by agents for arrangement..."(taken from [8])

2.3.3 Automatic Optimisation of Furniture Arrangement

Yu et al. (2011) [5] targets on furniture arrangement for complex scenes that are optimised with three key ergonomic factors: visibility, accessibility and pathway between two doors. They say they think their automatic arrangement is with a high degree of realism, and believe their work is the first to take the complex ergonomic factors under consideration and has solved the visibility, accessibility and with pathway in a cubic Bezier curve.

Their Methodology:

As is shown in Figure 2.9, their approach has two steps:

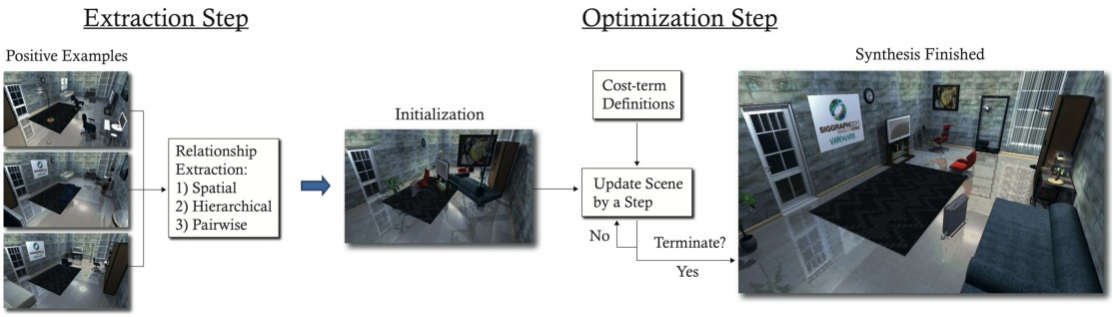

Figure 2.9: Approach overview of Yu et al. (2011)'s work (taken from [5])

1. **Furniture Relationship Extraction**: to extract spatial, hierarchical and pairwise relationships from positive examples;
2. **Furniture Arrangement Optimisation**: to optimise the corresponding cost functions recursively. [5]

Furniture Relationship Extraction:

Figure 2.10 shows the spatial relationships of the prior (a TV set), which are the prior distance (the TV set to its nearest wall) and orientation (its relative orientation to the wall). They are achieved by the clustered means of the positive examples.

Figure 2.11 shows the hierarchical relationships, which are the second-tier priors on the surface of the first-tier objects. The first-tier object is the object that

[5]Yu et al. (2011) [5] video site: `https://www.youtube.com/watch?v=vlDoSv6uDKQ`

Figure 2.10: Extraction step of prior distance and orientation (taken from [5]'s video show [5])

directly on the floor or walls. The objects on the surface of the first-tier object are the second-tier objects. In fact, their approach only supports two tiers of priors for simplicity.

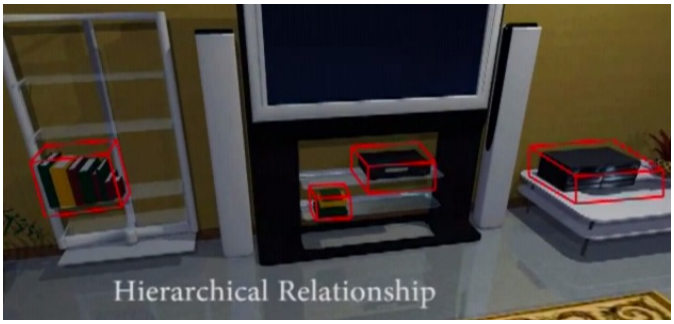

Figure 2.11: Extraction step of hierarchical relationships (taken from [5]'s video show)

Figure 2.12 shows a TV set and its pairwise, a sofa with a certain distance and angle from the positive examples. Other pairwises are like "a desk and a lamp" "a dining table and chairs".

Furniture Arrangement Optimisation:

They employed Metropolis Algorithm, which assumes the proposal distribution $q(x|y) = q(y|x)$ as it mentioned in Related Theories part, and a Boltzmann-like target function: [5]

$$f(\phi) = e^{-\beta C(\phi)} \tag{2.7}$$

Figure 2.12: Extraction step of pairwise relationships (taken from [5])

where $\phi = \{(p_i, \theta_i)|i = 1, 2, ..., n\}$, C is the cost function, which is under four main considerations: similarity to the extracted prior, accessibility (including furniture's and doors of the room), visibility and pairwise constraint. The β is the reciprocal of the simulated annealing temperature [15], according to the target function mathematical model they set. And the temperature is chosen differently for different furnishing problems, as it mentioned in the Simulated Annealing.

The first term, which is about the priors control of the similarity to what is extracted in last step; the prior cost is calculated from

$$C_{pr}^{d}(\phi) = \sum_{i} ||d_i - \bar{d}_i|| \tag{2.8}$$

$$C_{pr}^{\theta}(\phi) = \sum_{i} ||\theta_i - \bar{\theta}_i|| \tag{2.9}$$

where d_i and θ_i are calculated from current position p_i and rotation, which represent as the distance and relative angle to the nearest wall.

The accessibility is defined based on the functionality of that interior; if any object moves into another object's accessible space, the accessibility cost will increase. Figure 2.13 highlights the grids shape of a chair's accessibility. Figure 2.14 highlights the pathway connecting two doors in the cubic Bezier curve. Objects moving into the smooth curve area are penalised.

Figure 2.15 is on visibility, such as a TV set has a range of viewing angles, which will constrain the position range of its pairwise, say a chair.

The mathematical models applied to accessibility and visibility are the same,

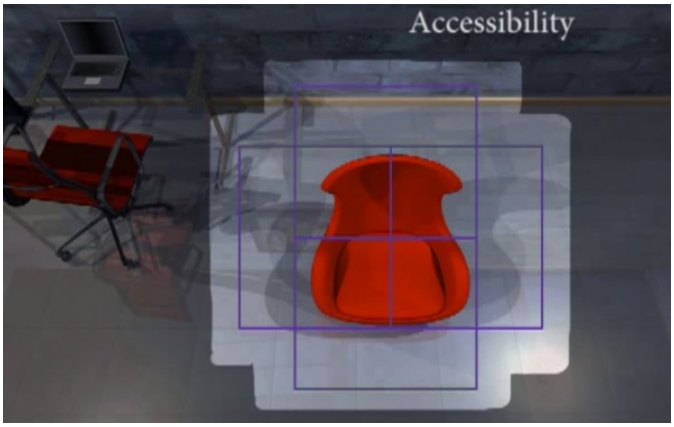

Figure 2.13: Optimisation step of accessibility consideration (taken from [5]'s video show)

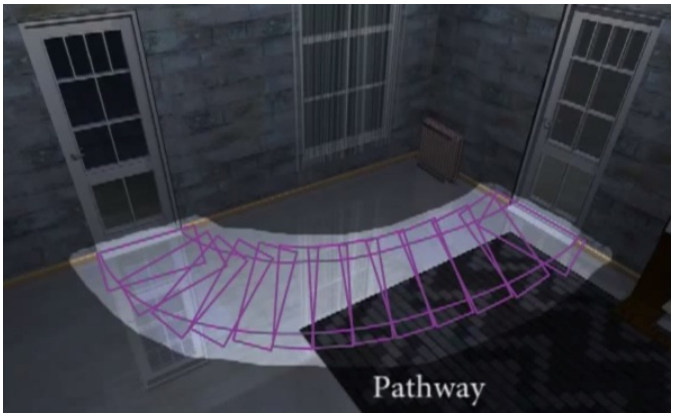

Figure 2.14: Optimisation step of pathway consideration connecting two doors (taken from [5])

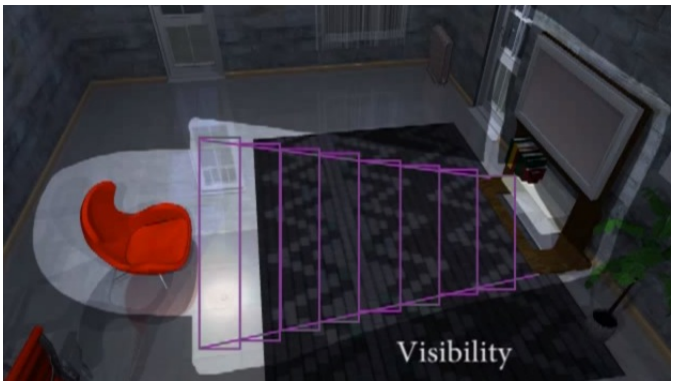

Figure 2.15: Optimisation step of visibility consideration (taken from [5]'s video show)

which are

$$C_a(\phi) = \sum_i \sum_j \sum_k \max\left[0,\ 1 - \frac{||p_i - a_{jk}||}{b_i + ad_{jk}}\right] \tag{2.10}$$

$$C_v(\phi) = \sum_i \sum_j \sum_k \max\left[0,\ 1 - \frac{||p_i - v_{jk}||}{b_i + vd_{jk}}\right] \tag{2.11}$$

where a_{jk} represents the four ($k = 1, 2, 3, 4$) accessible spaces around the object j (see the rectangles around the armchair in Figure 2.13), v_{jk} represents the angles to the facing direction from the centre of the object j at different distance ranges indexed as k, ad_{jk} and vd_{jk} are the diagonal lengths of these rectangles changing by the size of the object.

Lastly, the so-called "pairwise constraints" is to directly apply the d_i and θ_i gained from prior. The final cost function is the weighted sum of these terms above.

Their Limitations:

Yu et al. [5]'s project gives very plausible results for the automatic furnished rooms, but it implements the procedural arrangement with known certain interiors in supervised positions and rotations of the specific rooms.

Differently, I expected a more general settings for all different rooms in the Brighton House, and which furniture should be populated into the room is unknown. Another problem for me is that, the rooms in terrace houses are small compared to the sum of the furniture areas in the room. It is not easy to switch two furniture positions, meanwhile it is difficult to make sure none of them will collide with walls before switching, or how to handle the situation when the bigger furniture collide with walls after switching. However, furnishing a small room without switching, and with limitation on overlap detection (which will be talked about in Implementation Chapter), it is impossible to be guaranteed to go through most of the points. That was one of the challenges of my project.

2.3.4 Sweet Home 3D

As the project going, the data of the floor plan were expected to extract from the software that created it. Sweet Home 3D [16] is a free open source software for 2D floor plan design. It can automatically produce the 3D model of the 2D design, and also supports simple interior design. The 3D model could be exported as ".OBJ" and its textures mapping file ".mtl" with tiled texture images. Sweet Home 3D is compatible with "Windows, Mac OS X 10.4 to 10.9, Linux and Solaris" [16]. Its source code is written in Java, which could be downloaded from `http://sourceforge.net/projects/sweethome3d/files/SweetHome3D-source`.

This project edited the source code version 4.4 [17]. As there was some tricky steps to build its source code, the tutorial for the reader who wants to rebuild Sweet Home 3D is also provided in Appendices A.1.

Chapter 3

Design

There are many optional approaches that could be employed to automatically make decisions on which furniture should be populated into the room, and position them with some randomised process. In order to choose the proper method, the design should be feasible and tolerant of the programme requirements and the problem limitations from the project earlier stage to the end. This Chapter contains the design requirement, and the final programme framework with reasoning about this design.

3.1 Design Requirements

3.1.1 Functionality Requirements

The overall requirement for this project is to produce diverse and plausible furniture layouts in different rooms. To make it detailed, the programme was expected to determine the room type of the room to be furnished, and according to the room type, populate suitable types of furniture into the room at some random initial places with initial rotations, the optimise their positions and rotations to adapt the room environment.

Additionally, it was expected to control the furnishing style by scrolling a time bar, say from 1700s to now. The furniture style and fireplaces should be changes as the time. In fact, the floor plan also has slight changes, for example, since 1980s most of the residents removed the wall between the front room and rear room on

the ground floor, as the number of residents in one house reduced in general. As it is a programme of furnishing rooms in use, there should be with "the breath of life", which means the constraints of big furniture should not be too inhibited, and living products are also furnished. However, as a result, unfortunately the programme did not reach those requirements due to the time limitation.

3.1.2 External Limitations

This project was part of the Brighton VisAge project, which is written in Unity. In order to make it easier to merge with the Brighton VisAge project after the end of the project, the development platform has to be Unity3D.

There were several inconvenient factors for Unity3D and I want to discuss how these factors affect the whole project in the Discussion. At early stage, they have not be realised.

Another limitation was the time. Considering the three months, the chosen approach should not take too long for building a complicated system, but the system should preserve the interfaces for extending the programme with additional functions.

3.2 Programme Framework

3.2.1 Overview in Block Diagram

Figure 3.1 shows the block diagram of the programme design.

3.2.2 Extracted Data Structure

To furnish a room, the position of windows, doors and fireplaces constraints the furniture distribution in common sense. Automatic furnishing the room needs more: room corners coordinates, the wall normals pointing to room centre, windows/doors/fireplaces coordinates and facing directions, and the different effects of windows/doors/fireplaces on the furniture when it near them. Table 3.1 shows the data structure for extracted data from the floor plan. Considering the room shape

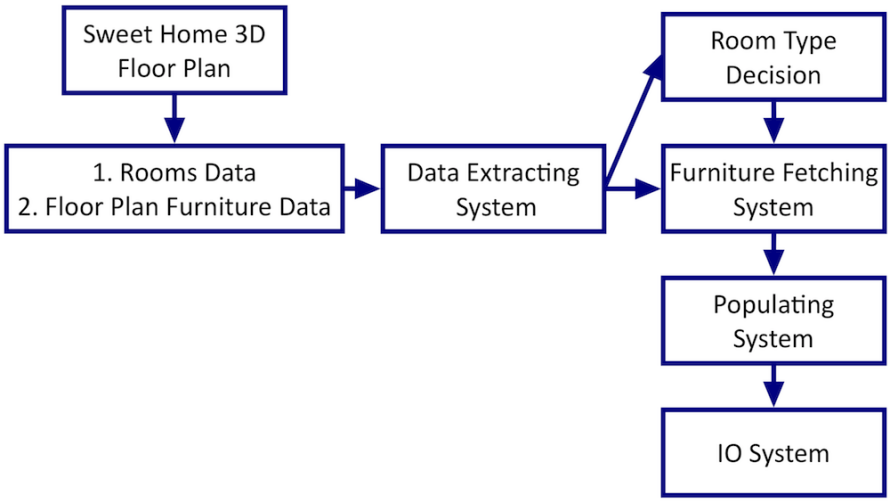

Figure 3.1: Programme design overview in block diagram

is not always in rectangle, the `roomArea` should not be calculated by the room extents. Among them, the "namecode" is encoded as "door=1", "window=2", "fireplace=3", otherwise=0.

Table 3.1: The data structure for extracted data from the floor plan

Variable Name	Format	Saved Content in Line
roomCenter	Vector3	the centre point on the floor
roomExtents	Vector3	the half size of the floor mesh bounding box
roomArea	float	the area calculated by Sweet Home 3D
floorCorners	Vector3[ ]	position of room corners on the floor
walls	Vector3[,3]	(starting point) (ending point) (normal)
floorplanFurniture	Vector3[ ][ ]	(namecode,wallID,0) (centre position) (extents) (width, depth, height)

The reason of this design is just to make it as simple as what was needed and what could be extracted from the floor plan.

3.2.3 Room Types and Furniture Types Decision Trees

Figure 3.2 shows how to decide a room type given a house floor plan. (The % mark means `mod`, which is get the remainder of the division.)

The threshold values were set according to the Brighton House layout. If you go through the tree, you may find a kitchen could be never reached for a whole

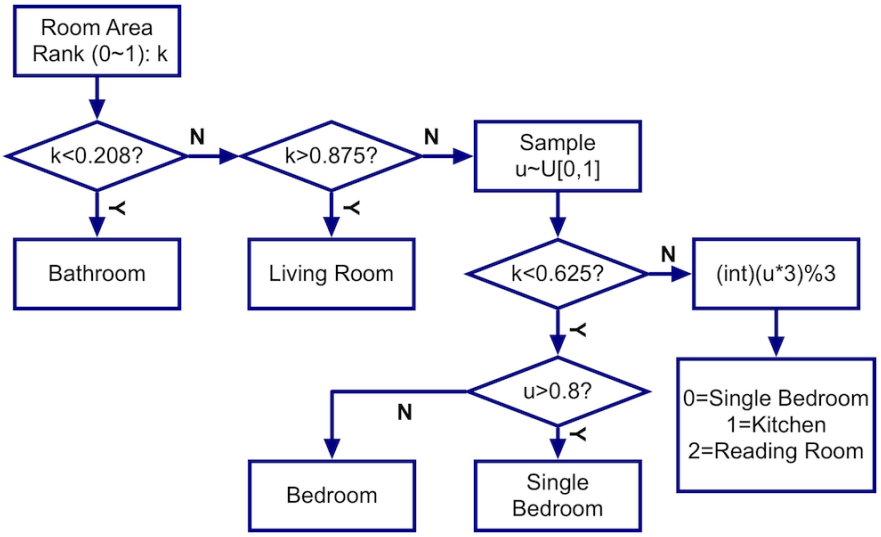

Figure 3.2: Room type decision tree

house. That was because one of the residents we visited preferred to extend the basement room to half garden and make the aisle connecting the original room and garden as kitchen, another two made the largest room in the basement as a living room combined as kitchen. Hence, I made the kitchen furniture "**must**" be populated when the living room is large enough.

The furniture decision flowchart is shown in 3.3. Models needed to be preprocessed with tagging them with their types (such as bed, table, chair etc.), rotating to the programme defined initial angle and much other trivial work. The `tags` in the flowchart is the list holding the allocated furniture tags, while the `furniture` is the list holding the prefab name of the chosen furniture to be populated. Each room type was given a guide on **first line** of key types of furniture according to the room type, **rest lines** of optional types of furniture according to the room type. Among the rest lines, **the first line of them** holds types of furniture could appear independently without making people think it abrupt, **the rest lines** are types of furniture which are commonly considered appearing together, for example sofa and tea-table. In summary, **line1** holds types that must be in; **line2** holds independent functional types of furniture; **the rest lines** hold related types furniture.

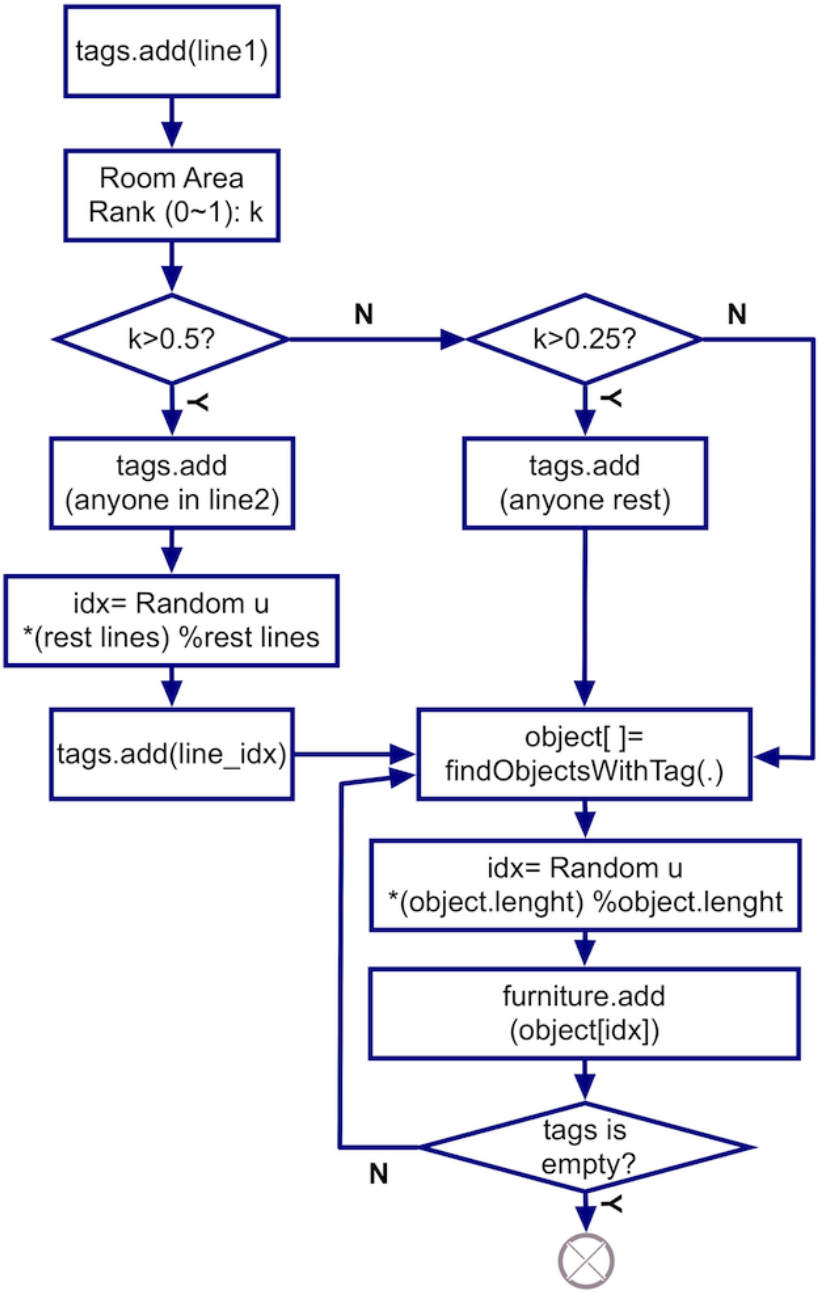

Figure 3.3: Furniture decision flowchart

3.2.4 Populating Optimisation System

With the precision constraints of Unity3D `Mathf`, the finding minimum problem should convert to finding maximum problem. Although Unity could do addition and so on such simple calculation in `double` precision, the calculation in the process could not be `double`, such as get two point distance, (whatever by root computing or Unity's vector dot magnitude). If so, as the cost goes down to zero, the difference between two steps will be not sensitive.

Therefore, this programme is to seek a global maximum. The probability function model is set as

$$p(x^*) = e^{\beta \cdot Score(x^*)}. \tag{3.1}$$

Thus as the *Score* rises, the probability rises. The corresponding Metropolis algorithm jump acceptance expression:

$$u < \min\left\{1, \frac{p(x^*)}{p(x_i)}\right\} \tag{3.2}$$

, where $u \sim U_{[0,1]}$, becomes

$$u < \min\left\{1, \frac{e^{\beta Score(x^*)}}{e^{\beta Score(x_i)}}\right\}. \tag{3.3}$$

Take natural logarithm of both side,

$$\ln(u) < \min\left\{\ln 1, \ln\left(\frac{e^{\beta Score(x^*)}}{e^{\beta Score(x_i)}}\right)\right\} \tag{3.4}$$

$$\ln(u) < \min\left\{0, \ln e^{\beta Score(x^*)} - \ln e^{\beta Score(x_i)}\right\} \tag{3.5}$$

$$\ln(u) < \min\left\{0, \beta(Score(x^*) - Score(x_i))\right\} \tag{3.6}$$

Because u is a random number between 0 and 1, $\ln u$ must smaller than zero. So the acceptance expression could be simplified as:

$$\ln(u) < \beta(Score(x^*) - Score(x_i)). \tag{3.7}$$

where β is inversely proportional to the annealing temperature. As the programme cycles, β will increase. The initial β and specific increment is discovered experimentally.

The score function has 7 terms, one for overall sum of distances between each other, the other six are "near wall score", "near corner score", "unblock door score", "unblock fireplace score", "not hiding window score" and "rotation score". To set the "near wall score" and "near corner score" differently was in order to lead a touching wall furniture goes to the corner instead of leaving a small gap between the wall and the furniture.

Due to the technique reasons, one testing took longer time to cool down if a better global best score was expected. And many related trivial tasks had to be done manually. The project time was very tight. The second tier furniture populating was given up, and the pairwise of the first tier furniture had not fully got worked.

3.2.5 IO System & Mobile Version

The IO system is to write out the furnished room with its furniture positions and rotations, as Unity will not save any changes in the scene view, when the game is running. Because of the long term for furnishing each room, the programme was not suitable to furnish in run time. So the mobile version only supports to read these written out results. The mobile version has first person control; players can view the furnished Brighton House by walking into it.

Next chapter gives a detailed description on the implementation in each layer of design.

Chapter 4

Implementation

In Chapter 3, we have got known of how did the automatic furnishing system work in a high level of logic algorithms. This chapter goes into some concrete pieces of code for important functions. The entire scripts are provided in Appendices B. But foremost it gives an account of the system environment in project development.

4.1 Development Environment

The built development environment with the details are shown in Table 4.1. The house floor plan was drawn using Sweet Home 3D [1]; as the project going, the windows, doors and fireplaces positions and covered areas were needed, so I edited the Sweet Home 3D source code[17] to write out all useful data the Sweet Home 3D can provide, then did some calculations on the two coordinate systems (Sweet Home 3D and Unity3D) transformation. The automatic furnishing part only worked for computers, and then written out the furnishing result for computers and Android devices to view.

There were many 3D models downloaded free on the Internet. The sites which supporting my project are:

1. Archieve3D, `http://archive3d.net/`;
2. Blogscopia, `http://resources.blogscopia.com/category/models/`;
3. TF3DM, `http://tf3dm.com/`;

[1] An open source software, which can be downloaded at `http://www.sweethome3d.com/`.

Table 4.1: The software installed to build the programming and testing environments

Environment	Specifications
Sweet Home 3D	Source code version: 4.4 Built Environment: JAVA 1.6.0, Ant-1.9.3 Running Environment: Mac OS X 10.9.4
Unity3D	Free Version: 4.5.0f6 Scripts are written in C# Andriod Built Setting: Minimum API Level 16 (Android 4.1) Computer Running Environment: MAC OS X 10.9.4
Android SDK (on Mac OS X 10.9.4)	Revision: 21 Build: v21.0.1-543035
Testing Devices	Mac OS X 10.9.4 Android 4.1.2

4. Sweet Home 3D,

   `http://www.sweethome3d.com/freeModels.jsp#ScopiaModels`;

5. Google 3D WareHouse, `https://3dwarehouse.sketchup.com/`;

6. KOLO (for Bathroom only), `http://www.sanitec-kolo.com/brand`;

4.2 Editing the Sweet Home 3D Source Code to Output Useful Floor Plan Data

The floor plan was created by Sweet Home 3D, saved using my edited version to export the featured floor plan data for the automatic furnishing system in Unity3D project. The tutorial on how to build and run Sweet Home 3D from source code is in Appendices A.1, and a brief guide on how to create a Brighton house floor plan is in Appendices A.2. This section mainly demonstrate the implementation of how to export the useful data from Sweet Home 3D source code and how to do the data transformation on two different world coordinates.

The code shown in Listing 4.1 runs when the user saves the working Sweet Home 3D project. It outputs two text files, "`rooms.txt`" and "`floorplanFurniture.txt`" under the `install` directory. The "`rooms.txt`" contains the information for

rooms: each room area, center and corners positions in 3D.

Listing 4.1: The code Inserted to the function `writeHome(Home home, String name)` in `SweetHome3D/src/com/eteks/sweethome3d/io/HomeFileRecorder.java` of Sweet Home 3D source code.

```
/**
 * 3D Room data output:
 * 1. RoomID, room area and room center
 * 2. Room corner coordinates for Unity3D(y-axis rotated 90 degree)
 * 3. Furniture (doors, windows, fireplaces, stairs) position and size
 * */
List<Room> rooms=home.getRooms();
PrintWriter txtFile = new PrintWriter(new FileWriter("rooms.txt"));
Iterator<Room> roomsIterator = rooms.iterator();
int id=0;
while (roomsIterator.hasNext()){
  id++;
  Room r = roomsIterator.next();
  Level roomlevel=r.getLevel();
  float y=roomlevel.getElevation();
  txtFile.println("RoomID " +id+" "+r.getArea());
  txtFile.println("center "+r.getYCenter()+" "+y+" "+r.getXCenter());//x=z,y=y,z=x
  float [][] roomCorners=r.getPoints();
  for(int i=0; i<roomCorners.length; i++){
    txtFile.println(roomCorners[i][1]+" "+y+" "+roomCorners[i][0]);
  }
}
txtFile.close();

txtFile = new PrintWriter(new FileWriter("floorplanFurniture.txt"));
List<HomePieceOfFurniture> furniture=home.getFurniture();
Iterator<HomePieceOfFurniture> furnitureIterator=furniture.iterator();
while(furnitureIterator.hasNext()){
  HomePieceOfFurniture f=furnitureIterator.next();
  Level flevel =f.getLevel();
  float y=(float)(flevel.getElevation()+f.getElevation()+0.5*f.getHeight());

  txtFile.println(flevel.getElevation()+" --------");
  txtFile.println("Furniture: "+f.getName());
  txtFile.println(f.getY()+" "+y+" "+f.getX());
  txtFile.println(f.getWidth()+" "+f.getDepth()+" "+f.getHeight());
}
txtFile.close();
```

Browsing the source code, it does not directly support room center data in 3D; however, we can get the room level for how height the room is elevated as the Y axis data. In fact, most of 3D models uses different world coordinates from Unity 3D. As we known Unity3D's Y-axis is in vertical, to get known the X-Z transformations, the center position was written out as "`getXCenter() getElevation() getYCenter()`" at first. Compared with the room centre position in centred imported floor plan with scale factor 1, they are different indeed.

Work Out the World Coordinates Transformation

In order to calculate the transformation, I determined to output the corners of the same room in these two coordinates systems. Unity3D does not support giving out the corners of a mesh, so I chose a rectangle room to calculate the four corners by its centre position and the mesh bounding box size. Code `gameobject.collider.bounds.center` gives the centre position in world coordinate of a children mesh under a prefab in Unity3D the Hierarchy window, and `gameobject.collider.bounds.extents` gives its half size.

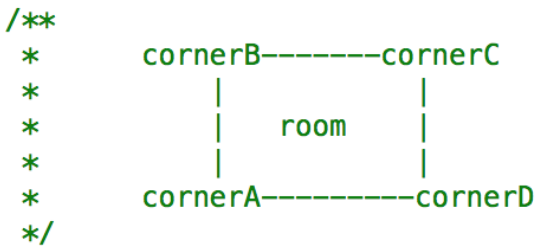

Figure 4.1: Room corner defined in Unity3D

If I define the four corners as the Figure 4.1 shown, the corners positions could be got as below:

```
1 roomCenter=gameObject.collider.bounds.center;
2 roomExtents=gameObject.collider.bounds.extents;
3 cornerA=roomCenter−roomExtents;
4 cornerA.y=roomCenter.y;
5 cornerB=new Vector3(roomCenter.x−roomExtents.x,roomCenter.y,roomCenter.z+
      roomExtents.z);
6 cornerC=roomCenter+roomExtents;
7 cornerC.y=roomCenter.y;
8 cornerD=new Vector3(roomCenter.x+roomExtents.x,roomCenter.y,roomCenter.z−
      roomExtents.z);
```

The output corners for two world coordinate systems are shown in Table 4.2

Table 4.2: Room corners for two world coordinate systems

Unity3D	Sweet Home 3D
cornerA (-452, -152, 8)	(241, -152, 8)
cornerB (-452, -152, 331)	(452, -152, 8)
cornerC (-241, -152, 331)	(452, -152, 331)
cornerD (-241, -152, 8)	(241, -152, 331)

If the imported model is rotated 90° in Y-axis, the corners positions are changed as Table 4.3

Table 4.3: Corner coordinates with 90° rotation in Unity3D Y-axis

Unity3D
cornerA (8, -152, 241)
cornerB (8, -152, 452)
cornerC (331, -152, 452)
cornerD (331, -152, 452)

As the result, the corners coordinates will be the same with Sweet Home 3D if we switch X and Z axes. Therefore, the inserted code in Sweet Home 3D wrote out positions as "`getZCenter() getElevation() getXCenter()`" order.

Floor Plan Furniture Position Calculation

As it mentioned before, Sweet Home 3D does not support to get objects 3D positions directly. There are some fixed furniture in the house floor plan, such as doors, windows, fireplaces and fireplaces. And they had not been allocated to each room, so a floor plan identification system was necessary for calculating out which room had which floor plan furniture and their positions, extents and local size. The differences between extents and local size are that numerically extents are half of size and in space extents take object rotation into account, while local size does not. The details on floor plan identification system will be demonstrated in later section. In Sweet Home 3D, only the centre positions in 3D and their local sizes were output in a list of all floor plan furniture. Y-axis coordinate was calculated by:

$$y = roomElevation + furnitureLocalElevation + \frac{1}{2} \times furnitureHeight$$

Finally, after editing the source code, use terminal to rebuild the project and open the created floor plan to re-save it. Now useful data in two `.txt` files are in the folder of `install` under Sweet Home 3D project directory, when saving the current Sweet Home 3D project.

4.3 Floor Plan Identification System

This section will talk about algorithms of getting the room floor corners and referred floor plan furniture centre positions, extents and local sizes among the scattered meshes of the exported floor plan model.

4.3.1 Identify the Floor Corners of the Room to Be furnished

When the room floor is in rectangle, the corner positions could be calculated by mesh `center` and `extents`. However, there are rooms with more than four walls, even in concave polygon as the front room shown in middle of the Figure A.3, and the aisle is also considered as a room in Sweet Home 3D.

Appendices C.1 lists the output rooms data in last section, and one room data is in the sequence of:

```
"RoomID" <id> <room area>
"center" <x> <y> <z>
<x> <y> <z>
<x> <y> <z>
<x> <y> <z>
...
```

Therefore, the algorithm to find currently furnishing room corners is shown in Listing 4.2

Listing 4.2: Algorithm: Finding currently furnishing room corners

```
1 //read room centres coordinates
2 open the file
3 text=reader.ReadLine();
4 do{
```

```
5     if(text.StartsWith("c")){
6       word=text.Split(' ');
7       //word[0]="center"
8       //word[1:3]=<x,y,z>
9       list.Add(word[1:3]);
10    }
11    text=reader.ReadLine();
12  }while(text != null);
13  close the file
14
15  do world coordinate conversion on current gameobject centre position;
16  find the nearest point index in the list;
17  roomID=nearest point index +1;//because the output RoomID starts from 1
18  get the new room centre, and do world coordinate conversion
19
20  //the gameobject centre which got from Unity3D is the cube mesh centre (under the floor
        surface)
21  //the written out new room centre is the centre position on the floor
22  update the roomCenter variable;
23
24  //read corners coordinates
25  reopen the file
26  do{
27      if(text.StartsWith("R")){
28          rID=text.Split(' ');
29          //RoomID <roomID> <roomArea>
30          if(rID[1]==roomID){
31              roomArea=float.Parse(rID[2]);
32                  text=reader.ReadLine();//center
33                  text=reader.ReadLine();//corners starts
34                  do{
35                      word=text.Split(' ');
36                      corner= word[0:2];
37                      change to Unity world coordinates;
38                  }while(text!=null && !text.StartsWith("R"))
39                  break;
40              }
41          }
42          text=reader.ReadLine();
43      }while(text != null);
44  close the file
```

The information about the walls around the room is also useful. Listing 4.3 shows the code for giving the walls starting point and ending point, then calculating

the wall normals that point to room centre. The result of normalised y-axis vector crossing the wall vector from starting point pointing to the ending point gives the expected room wall normals.

Listing 4.3: Code: Get room walls(starting point, ending point, normal)

```
/**
 * Get walls infomation
 */
int NumOfCorners=floorCorners.Length;
walls=new Vector3[NumOfCorners,3];
for(int i=0;i<NumOfCorners;i++){
   //[][0] wall start point
   walls[i,0]=floorCorners[i];

   //[][1] wall end point
   if(i+1<NumOfCorners){
      walls[i,1]=floorCorners[i+1];
   }else{
      walls[i,1]=floorCorners [0];
   }

   //calculate the wall normal
   //Unity is left hand for cross product:
   //  http://docs.unity3d.com/ScriptReference/Vector3.Cross.html
   //and the output wall order is clockwise
   //in order to point inside the room:
   //it should be Y-axis cross the wallline pointing to the end point
   Vector3 A=walls[i,1]-walls[i,0];//pointing to the wall end point
   Vector3 normal= Vector3.Cross(new Vector3(0,1,0),A);
   normal=normal.normalized;

   walls[i,2]=normal;

}//get all walls lines
```

The entire code for this part is shown in Listing B.1.

4.3.2 Identify the Floor Plan Furniture in the Room to Be furnished

After the corners and walls of the room were identified, the searching floor plan furniture system would work. It read `floorplanFurniture.txt` file, for instance

the text shown in Appendices C.2. Each floor plan furniture data in sequence of:

```
<elevation> "-------------"
"Furniture:" <furniture name>
Vector3 <center>
Vector3 <width, depth, height>
```

The type of floor plan furniture, centre position, extents, local size and pointing direction are needed for each floor plan furniture of the room. The last word of each floor plan furniture name is their type. Centre positions and local sizes are simply read in from the text file and update them to Unity3D world coordinates.

The pointing direction is determined by which wall it belongs to, and the extents depend on its pointing direction. Finding the nearest wall algorithm is based on looking for the shortest distance among all room walls in 2D (the X-Z coordinate). The shortest distance size is not able to determine whether the furniture is in room. For example in Figure 4.2, an opened door of the adjacent room can have a smaller distance to its nearest wall than that distance of a thick fireplace or the opened door for this room.

```
/**
 * After the wallID is determined,
 * it needs to check whether the furniture is in room
 * (e.g. the case TWO DOORS like:
 *   |------|
 *   |      |
 * _\|room  |       _\: another room's opened door
 *   |      |
 *   |/     |       |/: this room's opened door
 *   -------
 *
 *  ^z
 *  |-->x
 * )
 */
```

Figure 4.2: The two doors case for determining whether the furniture is in room.

Therefore, the algorithm to determine whether the furniture is in the room should be to compare the distance from the object to the line of the room centre in its nearest wall direction with the distance between this line and the nearest wall line. If the former distance is smaller than the latter distance, the object is in room.

In other words, assuming a non-rectangle room shown in Figure 4.3, **A** is the room centre, **B** and **C** are the two objects whose nearest wall belongs to the room, and **O** is the original point of the coordinate system. The starting point and

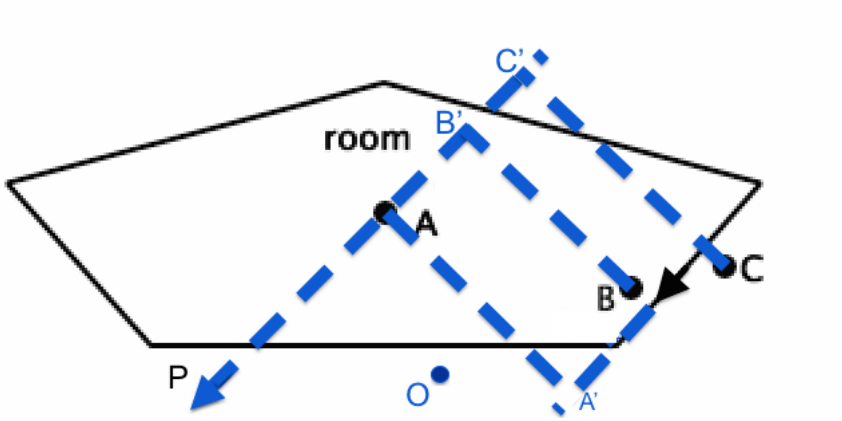

Figure 4.3: Figure for the algorithm to determine whether a furniture is in room.

ending point have been known in last subsection. Assuming the line from Point **A** pointing in the wall direction is known, the problem becomes to compare the length of **BB'** or **CC'** with the length of **AA'**. **AA'** could be calculated by point to two-point-line distance. If we know the other point of line A in wall direction, all the distance could be calculated by a point to two-point-line distance function.

The wall vector is `startingPoint` − `endingPoint`, say $\vec{v}$. Let's call the unknow point as "P":

$$\mathbf{A} + \vec{v} = \overrightarrow{AP} \tag{4.1}$$

According to the vector addition,

$$\overrightarrow{OA} + \overrightarrow{AP} = \overrightarrow{OP} \tag{4.2}$$

And the vector $\overrightarrow{OP}$ equals Point **P** coordinate minus original point coordinate which is `Vector2(0,0)`, then we get the Point P coordinate.

$$\overrightarrow{OP} = \mathbf{P} - \mathbf{O} = \mathbf{P} \tag{4.3}$$

Now it only needs a function to calculate the distance from a point **A** to the line of another two points **BC**.

As we know the magnitude of two vectors cross product equals the area of a

parallelogram with these two vectors for sides. Therefore,

$$\texttt{distance from A to BC} = \frac{\overrightarrow{BC} \times \overrightarrow{BA}}{||\overrightarrow{BC}||} \tag{4.4}$$

This algorithm does not support to find the exact wall when the object is very closed to a wall corner, but in practice, this case will not affect too much on furnishing optimisation, as the left side of the floor plan furniture is equivalent to its right side.

At this stage, the furniture type, centre position, whether in room, the pointing direction (in vector) and the local size in Unity3D coordinates. We can estimate the extents of the bounding box now:

When the angle between furniture tangent (which is its wall vector) and X-axis is smaller than 45° or between 180° to 225°,

$$\texttt{furniture extents(x,y,z)} = \frac{1}{2}(\texttt{width,height,depth}) \tag{4.5}$$

Otherwise the extents are considered as

$$\texttt{furniture extents(x,y,z)} = \frac{1}{2}(\texttt{depth,height,width}) \tag{4.6}$$

The entire code for this part is shown in Listing B.2.

4.4 Furniture Fetching System

This section demonstrates the decision trees for determining a room type and which types of furniture to be populated in the room. Before the decision system, all downloaded 3D models must be preprocessed.

4.4.1 3D Model Preprocessing

Define the Oriented Bounding Box Faces

Figure 4.4 shows my Oriented Bounding Box face ID. To make it easy to remember, it follows the six face number of a die in space, which are 1's opposite is 6; 2's opposite is 5; and 3's opposite is 4. The initial orientation is Face 4 pointing to X-axis, Face 1 pointing to Y-axis, Face 2 pointing in Z-axis, and Face 2 is defined as forward direction. It reveals to the programme, if the furniture is along the wall, its Face 5 should be the face touching the wall. The lower graph in Figure 4.4 gives an example for bed on top of the 3D view.

```
/** Oriented Bounding Box face code:
 * -like a die:
 *
 *     ^ y              1: top area
 *     |                6: bottom area
 *    _|__
 *  /__1_ /|
 * |     |2|            2: facing area
 * |  4  | |--------->z  5: back area
 * |_____|/
 *    /
 *   / x                3: left area
 *  v                   4: right area
 *
 * Bed:
 *           __2__
 * ^z       |    |
 * |       3|    |4
 * +-->x     =====      (1:top;6:bottom)
 *             5
 */
```

Figure 4.4: My Oriented Bounding Box (OBB) face code, and an example of the bed OBB

Procedures

1. Drag the 3D model and its textures and texture mapping files into the project asset, and manually adjust the "meshes scale factor" in "import settings" referring to your floor plan.

   Because the authors of these models were different, it was a heavy work to change them manually and properly. Additionally, because "Unitys physics

system expects 1 meter in the game world to be 1 unit in the imported file" [18], the scale factor of the floor plan was changed to 0.1 finally.

2. Complement the lost materials and textures. Mainly on material shaders, lost alpha value in colours and unmapped textures.

3. Rotate the model until its Face 2 facing positive Z-axis. This step must be done after scaling, otherwise the model will be sheared.

4. When most of the models are rotated, their rotation centres will be found that they are not the actual centres. The simple solution to reset the rotation centre to the model centre is to double click the name of the model in the Hierarchy window, then the scene will focus on and centred this GameObject. Then click menu "GameObject/Create Empty", the created empty GameObject will be set at the model's real centre. Finally, drag the name of the model to the empty Gameobject in the Hierarchy window.

5. Save that model as a prefab. [19], which is the GameObject that you will directly handle in Unity3D project;

6. Delete the model in Hierarchy, and drag the prefab to scene;

7. Finally, set the prefab transform position to $(0, 0, 0)$.

4.4.2 Room Type Decision and Furniture Type Allocation

The implementation of the room type decision and furniture type allocation guide is in simple logic, which is exactly as same as what discussed in Design Chapter. This part of code is shown in Listing B.3. The thresholds of those generated random numbers were learnt from the actual floor plan, and were expected to be able to extend for other houses.

Because of the compatibility of importing 3D models to Unity3D, the programme cut down a lot of models, as the centre positions and extents of their bounding boxes were never right. The was no way to tell them out from some regularities, such as appearance, model format, scaling range, or complexity. It took me long time to filter one out and delete it, when the testings randomly pick a "incompatible" (let's call it as "incompatible") models. Therefore, many types of furniture were cut down to one actual model in the end. Nevertheless, the programme supports to do random selection from whatever the number of optional

models was. (Please at lease one model in that tag, otherwise, the tag would be not defined.)

4.5 Furniture Populating Optimisation System

This was the core system of this project, which was the most interesting and painful part. The interesting point is to explore some different function giving marks to each furniture, trying to thinking at the furniture standpoint to evaluate why this function designed artful or silly. The painful is the creative ideas always died from Unity's computing limitations and unknown bugs on uncertain models.

Due to the bad angle computing support of Unity3D, which has been mentioned in the Introduction chapter, all the functions that took angles in had to be cancelled or updated in another thinking.

Another detour might be the Unity physics engine for "rigidbody" gameobjects. At the beginning of the project, I did not know that "rigidbody" could stop two gameobjects when they started to collide each other. There were two score functions; one for sum of distances, the other was on overlaps. The problem on overlap score was that it could not distinguish whether the jump was better or not when they had been overlapped. The overlapped area was too complicated to calculate, especially when the rotations added in. So when I found one of the Unity3D components, "rigidbody" could avoid overlaps, I was stuck on it. The side-effect of the "rigidbody" is that you could not control it, once a move makes the furniture dragged to the floor plan, which due to many causes, such as the moving speed is too fast for unity physics engine to detect the collision entering, but when it detected, that was too late for the furniture leaving the floor plan or re-enter the room. There are many developers asking solution for it, for example Asle [20] asked about stop shaking when digging into another one accidental. But with hard tries on it, it seems nothing real works. You only can expect after strongly shaking, it could escape from the place caught it, before the annealing temperature cooling down. This property made me gave up my function of switch any two furniture with their positions reluctantly, but the probability of causing the bigger furniture collides with the wall was too big. As the feature of the problem, a Brighton House, it was not a gallery with several small objects in a big space; if the furniture could not have a chance to pass the global best state by other furniture blocked it, no matter how many iterations it did, the global best

of it passed looked not maximise each term of the score function.

In my programme there were seven terms for giving a score to the so-called "proposal jump": Using the physics engine, the furniture have to actually move to that state first, and then do Metropolis-Hastings for staying on this step or roll back to last step, because the step may not be the same as the step you proposed to walk into. Some walks may be stopped and bounced to another direction, but the data used for scoring were fetched from the actual scene, instead of the generated random walk.

The seven terms of score function are on the sum of furniture mutual distances, sum of the distances of each furniture's nearest wall and with a judging process for their rotations (these two considerations combined together due do the mentioned Unity angle difficulties), sum of their nearest corners, windows shielded score, fireplaces shielded score, and door path score. The last three terms on doors, windows, and fireplaces were in same expression, but with different experimental parameters.

4.5.1 Distance Score

There were two versions of the sum of distance term. The previous version ignoring the size of the objects, simply added the point-to-point difference up:

$$DistanceScore_1 = distanceFactor \cdot \sum_{i=0}^{N} \sum_{j=i+1}^{N} ||P_i - P_j|| \tag{4.7}$$

This function is simple and effective, however, if the objects are different, the biggest object must be centred so that other smaller objects could go further away due to the limited room space. So I also tried a function like that:

$$DistanceScore_2 = Distance3D + \min\{D_{ij}\} \tag{4.8}$$

where $Distance3D$ is

$$Distance3D = \frac{||P_i - RoomCentre|| + ||Extents(P_i)||}{2i + 1} \tag{4.9}$$

and $Extents(P_i)$ is

$$Extents(P_i) = \frac{1}{2} \cdot \texttt{Vector3}\,[width(P_i), depth(P_i), height(P_i)] \tag{4.10}$$

Equation 4.9 gives a self-defined non-particle estimated 3D distance, which is the distance from the room centre to the object centre plus the project extents magnitude[2]. In space, the distance is shown in Figure 4.5. This distance is regarded as the distance from the room centre the object furthest point, which attempts to balance all size of objects. The denominator of the $Distance3D$ attempts to give higher score to the object with a larger 2D area, because the furniture in the queue of to be populated has been sorted by their 2D areas from biggest to the smallest. The index i times 2 is in order to amplify the difference between two adjacent (in index) furniture, and plus 1 is just to make sure the first furniture whose index is 0 would not lead to a infinite score.

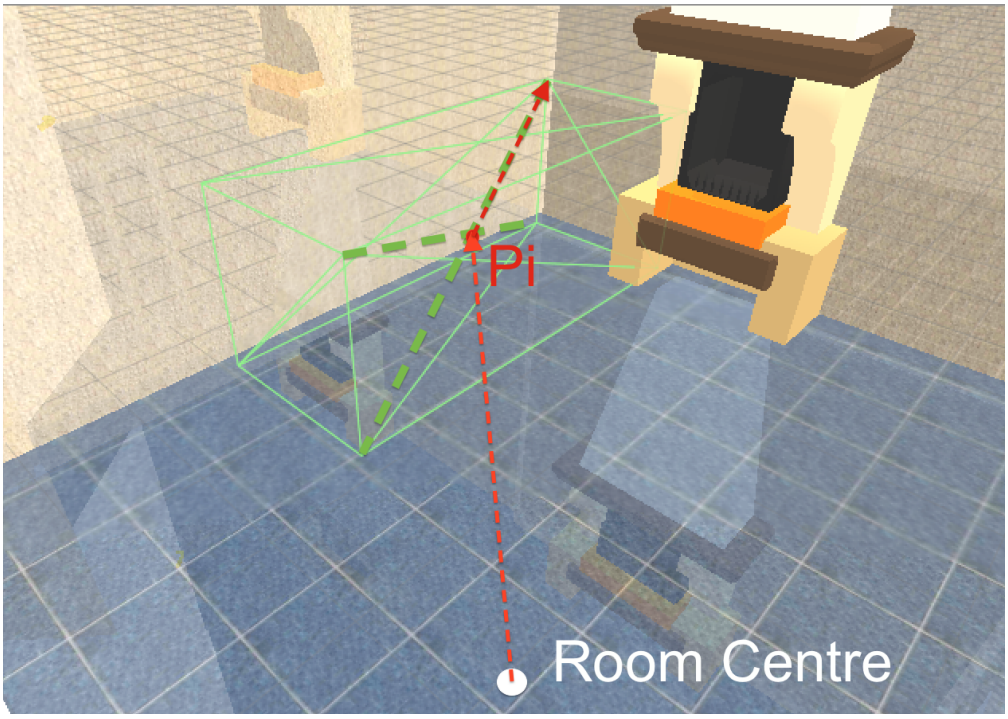

Figure 4.5: Self-defined non-particle estimated 3D distance (the red dot line)

Nonetheless first distance function is effective when the target domain is only with this one term. The weight "2" in Equation 4.9 was got by trials. In later functions, if a concrete weight comes out but without any discussion on why is this magnitude, that implies the value was got by trials and errors.

[2]In Unity3D, gameobject.collider.bounds.extents is half of gameobject.collider.bounds.size. Both of them are 3D vectors.

4.5.2 Nearest Wall Score (Cooperated with Rotations)

Recall the Equation 4.10; it will be mentioned as "Ex" for short, and "Ex.x" means the first number of the vector 'Ex'", "Ex.y" indicates the second number and so on. In Unity, the horizontal plane is defined by x-z axes, y-axis is the vertical axis. So "Ex.x" and "Ex.z" is the two orthogonal half of the object size.

$$NearestWallDistanceScore = \frac{100}{(wallDistance + 1)(10i + 1)} \tag{4.11}$$

where $wallDistance$ is

$$wallDistance = NearestWallDistance - \texttt{Ex}_i\texttt{.z} \tag{4.12}$$

where $\texttt{Ex}_i\texttt{.z}$ equals to the distance from expected facing wall face (encoded as Face 5 in above sections) to the object centre.

$$NearestWallDistance = \min\{distance(P_i, Wall_j)\} \tag{4.13}$$

Below is the rotation check for $NearestWallDistance$, which expects to distinguish the objects giving a corner to nearest wall with the objects has a side parallel the nearest wall. Remind the `Vector3.Angle` only gives the angle between 0 and 180. This function is just for making the room looks not jumbled, as human eyes are sensitive on inhibited rotations than inhibited distances [5].

```
1 /**
2  * Rotation check
3  */
4 float rotationY=boxesT1[i].transform.localEulerAngles.y;
5 float targetedRY=Vector3.Angle(new Vector3(0,0,1),Room.walls[wallID,2]);
6 if(Mathf.Abs(rotationY-targetedRY)>45){
7   NearestWallDistanceScore=NearestWallDistanceScore/2;
8 }
```

4.5.3 Nearest Corner Score

Nearest Corner Score is with the similar expression with nearest wall score, but without the rotation checking and cutting down mechanism.

$$NearestCornerDistanceScore = \frac{100}{(cornerDistance + 1)(10i + 1)} \tag{4.14}$$

where $cornerDistance$ is

$$cornerDistance = NearestCornerDistance - \sqrt{\texttt{Ex}_i\texttt{.x}^2 + \texttt{Ex}_i\texttt{.z}^2} \tag{4.15}$$

$$NearestCornerDistance = \min\{||P_i - Corner_j||\} \tag{4.16}$$

4.5.4 Doors, windows and fireplaces Scores

The three terms has the similar mathematical model, it is adequate to describe one of them here. The code in Appendices B.4 gives all other information.

The key point for these scores is to the angle between the vector which is from the floor plan furniture centre pointing to the object centre and the floor plan furniture facing vector. The main thought is when the angle is tending to zero, and the distance from the object to the floor plan is considered as "small", the score should be low. As it is not so easy to find facing angle or vector exactly, on one hand I used the distance to filter out objects which are not closed to it, by finding objects' nearest wall ID, if it is the same wall with where the floor plan furniture is at, this object would be focused, otherwise, it will be given 100 score. On the other hand, the cosine value of the angle is used to weight the distance, mathematically:

$$FireplaceShieldedScore = \frac{FireplaceDistance + 1}{\cos\theta + 1} \tag{4.17}$$

where the $\cos\theta$ is got by the dot product for the normalised distance vector from the floor plan furniture pointing to the object, and the normalised floor plan furniture normal.

This function has an abuse: maximum of cos is just 1, which is not sensitive

enough for object with a narrow angle. I wish it give a very low score on high object directly in front of it, but welcome to stand in a line with the floor plan furniture. With many different trials, such as enlarge the amplitude of the cosine function, or other structure of expressions. I found this worked nice:

Listing 4.4: The end of the fireplace term

```
1  if(cosTheta>0.87){//expected narrow than window
2          FireplaceShieldedScore=FireplaceShieldedScore- FireplaceDistance/(cosTheta+1)
3            -10/(FireplaceDistance+1);
4        }
```

There are two advantages of this ending: First, the one who standing ahead of the floor plan furniture will get a negative score, but for the thick furniture (whose centre has a distance that could not be ignored to the θ) standing beside it will be protected somehow. Secondly, it makes the programme easier to control which one should sacrifice by the threshold cosine value (the "0.87") in the above listing, when the room is very small. For example, the bath room of terrace house is very small, but with a door and a window. To set up a shower space, the largest distance is also small in this case. If the door threshold is set larger than the window's, the shower would prefer to approach to the window. According to common sense, shielding the window is never worse than blocking the door.

4.5.5 Important Process and Sequence in One Cycle

So far, all important mathematical models have been discussed; now it is time to give the whole algorithm on how this system achieved in Listing 4.5.

Listing 4.5: Important Process and Sequence in One Cycle

```
1  if(! isInitialised  && PopulatingGuide.Instance.isfinished
2     && InRoomRetrieval.Instance.isfinished){
3     /**
4      * getKnownFloorplanFurniture()
5      * determintInitialFurniture()-->and sort by area
6      * importInitialFurniture()-->will stop by if the room isFull()
7      */
8     initialization ();
9    getOverallScore();//-->with getAllDistanceScore() and getSingleScore()
10
11    //-----End-----
12     isInitialised =true;
```

```
13    lastSumOfScores=currentDistanceScore;
14    iteration=new int[nameList.Count];
15  }//if(PopulatingGuide.Instance.isfinished
16
17  if( isInitialised  && !isStable){
18    lastDistanceScore=currentDistanceScore;
19    getOverallScore();
20    if(Mathf.Abs((float)(currentDistanceScore-lastDistanceScore))<step){
21      isStable=true;
22    }
23  }
24  if(isStable  && !isfinished){
25    recordLastPosition();
26    simulatedAnnealingControl();
27    move();
28    ScoresHandling();
29    getOverallScore();
30    compare with globalbest
31    MetropolisHastings();
32    isStable=false;
33  }
```

4.6 IO System & Viewing Version on Android

This system was only developed for testing whether the model scale factor is proper at the beginning, so it is very simple and easy to understand from code. The relavant code is in Appendix B.5.

Next Chapter will give testings results numerically in unit cost function term, and the whole programme testing results graphically.

Chapter 5

Testing & Results

This chapter demonstrates the testing results with 1D, 2D and 3D graphs. The 1D picture gives the score tendency along the iteration times, the 2D picture shows the positions and general rotations of the furnishing, and the 3D picture gives a clear viewpoint on each furniture rotation.

As the time of this project was really limited, this paper probably would be printed in Black and White, if there is any unclear picture that is still attract you, I will upload the Thesis under this project GitHub folder; the address is shown in the beginning of Appendices B. Also, there were several testings on my youtube channel MSc Final Project playlist [1].

5.1 MH Score Unit Testing

5.1.1 Distance-Term Optimisation

Figure 5.1 shows the 1D numerical result of the MH algorithm with single sum of distance term. There are two vertical lines in the graph, which indicates the iteration times equal to 1200 and 2400. The three phases of different beta value which is inversely proportional to the simulated annealing temperature. The beta is set 0.1 initially, as the temperature cool down, beta is set growing up. The first

[1] `https://www.youtube.com/playlist?list=PL7Mta_KGtxPgym-MX9QBSu1GjBfcZ_3vY`
(Welcome scanning for it:)

transition is at the defined criteria iteration times, which equals to the number of objects times in room times 300. In its 2D result, Figure 5.2, shows there are four objects in the room, so the criteria iteration times is 1200, the second transition is at when iteration times reaches 2.5 times of the criteria iteration times. After that, considering the schema is to jump first, and then do MH computation to determine whether rolling back. I also set the step of each random walk reducing when reaching the third cycle of the simulated annealing control algorithm. Because if without it, the programme will end with keeping a bigger (compared to the reduced step) step of walk and bounce by hitting something else (due to the rigidbody physics engine) then roll back to last position. As a result it may miss to capture a possible better score in between. The programme is always keeps a current global best recorded from each history jump, so it does not matter, if the step goes to very small, even if the beta has been raised up, some steps in lower score still have a chance to be accepted. In the end, when the step becomes too small (for this floor plan size, it's 0.1, $\frac{1}{10}$ of the initial step), no matter where the MH algorithm reached, it will go back to the history global best, and written out to the IO system.

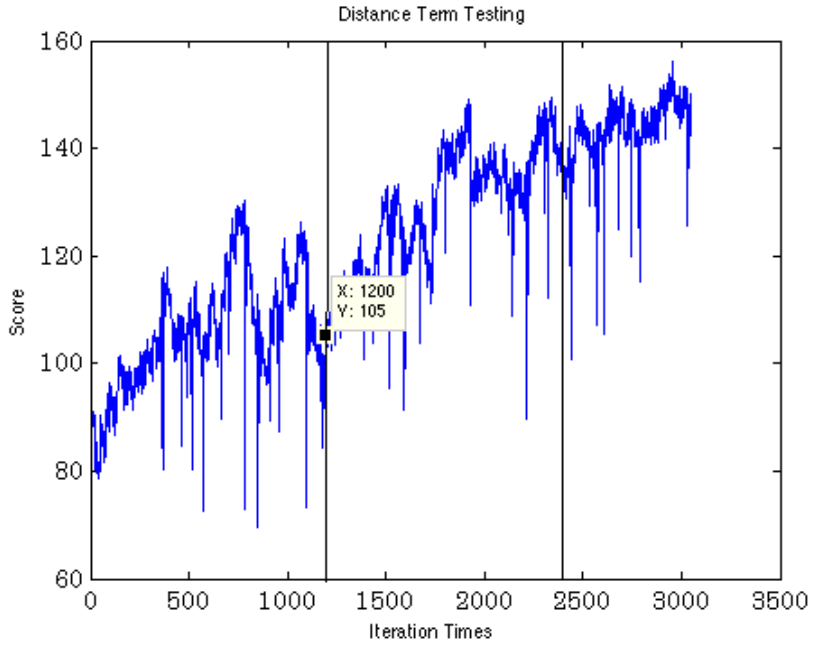

Figure 5.1: Distance-Term Optimisation 1D result

You may noticed there are many single drops in the process of going to global best score of Figure 5.1. That is caused by `jumpOne()`. As I mentioned, to switch two objects was too hard to control and handling the bigger object colliding with walls, especially at the ending stage, most of the objects have been closed to the

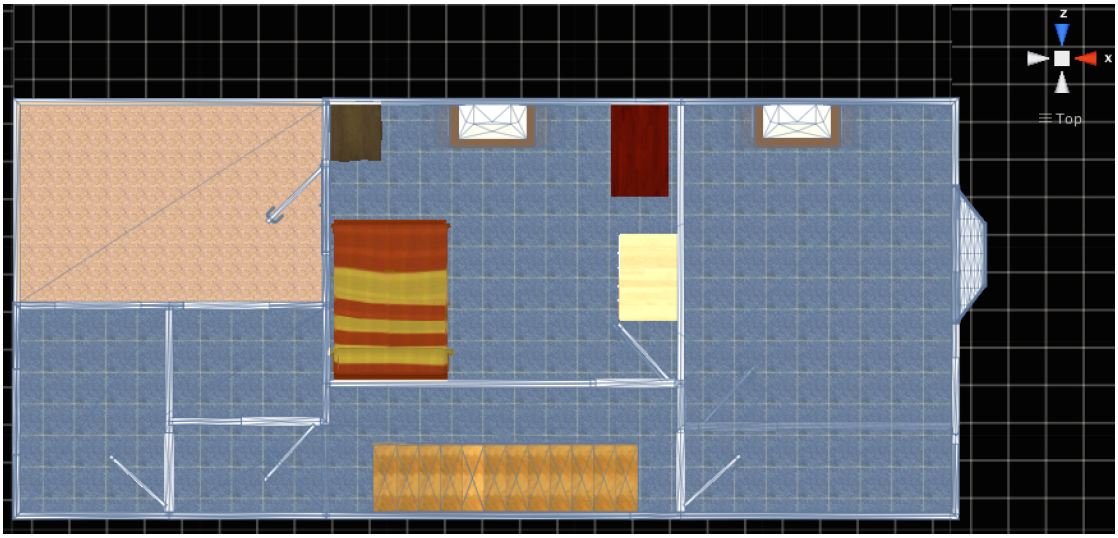

Figure 5.2: Distance-Term Optimisation 2D result

wall. Therefore, my programme have a very low chance (0.5%) to make any one except the biggest one get a new random position in the room before next jump. The `jumpOne()` plays an important role at early stage, when the energy (the score) goes up and down with a big amplitude. It makes it possible to go through all possible global best positions when the room is crowded.

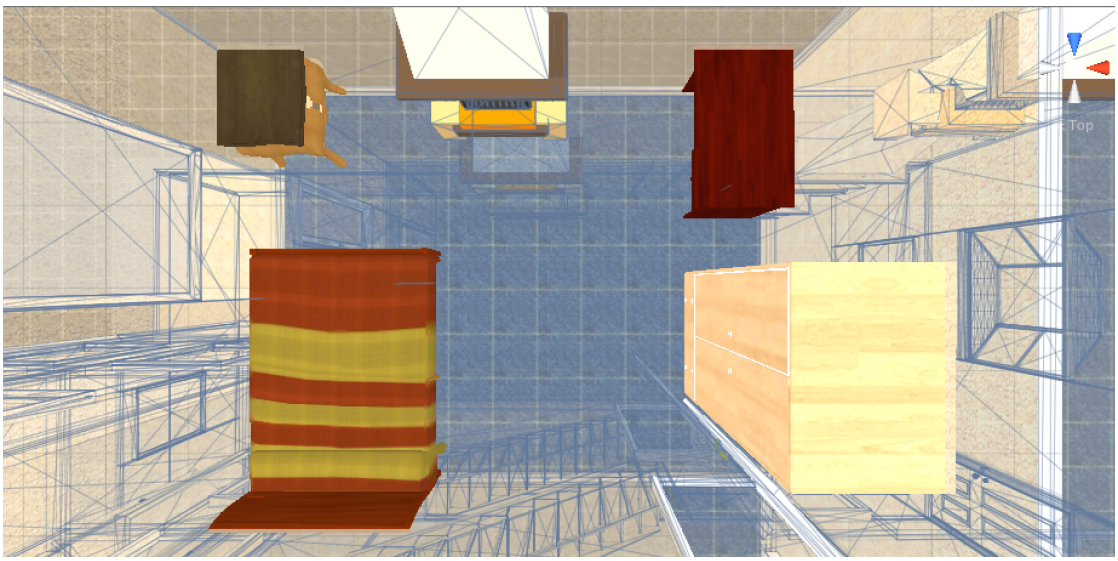

Figure 5.3: Distance-Term Optimisation 3D result

Lastly, Figure 5.3 shows the graphical result in 3D. In 2D, it only can show the rotation of the bed is right. In 3D, it could tell that all objects are in the right rotation. I was very satisfied on the result, after a long term digging into the Unity rotations and angle computing problems.

5.1.2 Along-Walls Optimisation

In this testing, all simulated annealing parameters were set exactly the same with last test, and the furnishing room was changed. As it has been a different problem, in theory, the temperature cooling scheme should be reset. However, we are expected to find a tolerant setting for the whole house, even expected to extend it has a good compatible with other floor plans. So it is not strange that Figure 5.4 got the cooling temperature decreasing too slow. Most of the MH jumps were accepted in the whole progress. However, we can still tell the beta effects on the 1D graph, the amplitude of the oscillation in scores becomes smaller in the end. Including the beta magnitude effect, there are another two possible reasons why this case has a slower cooling speed than last one: one is the weight of the score is set too small to distinguish two adjacent random walks, the other is the target domain is wider than last one, since closed to the wall with a right rotation will get the similar high score. If that, this result should be exactly what we expected, because from the final graphics result, Figure 5.5 and Figure 5.6, the algorithm do reach a satisfied state.

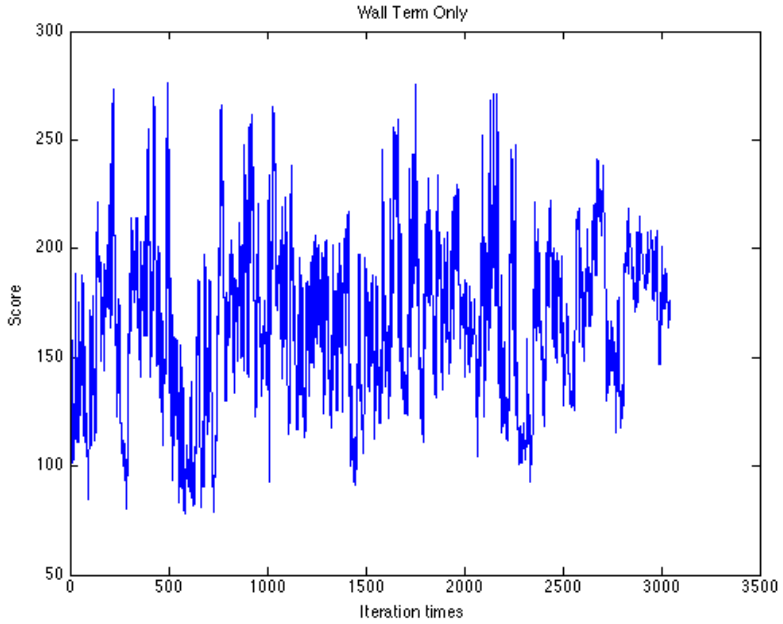

Figure 5.4: Wall-Term Optimisation 1D result

Near-to-Corners Optimisation is even simple than the along-wall optimisation, and the weight is 1 for this term in score, as same as the above one.

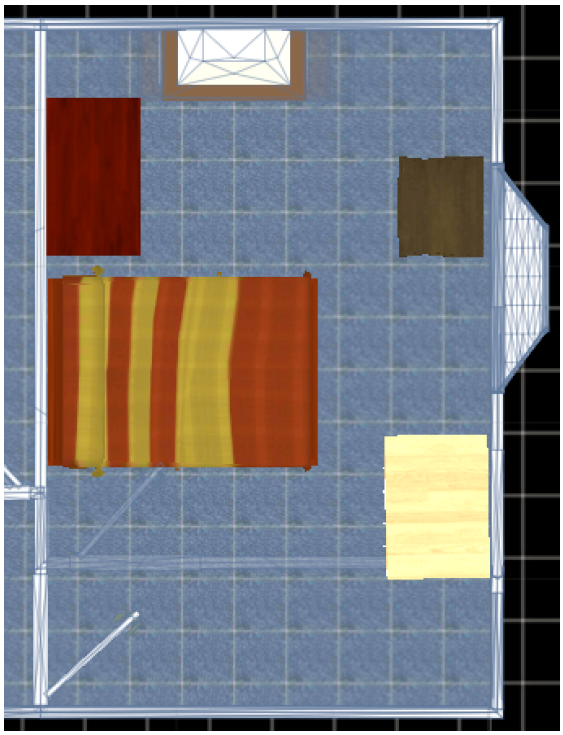

Figure 5.5: Wall-Term Optimisation 2D result

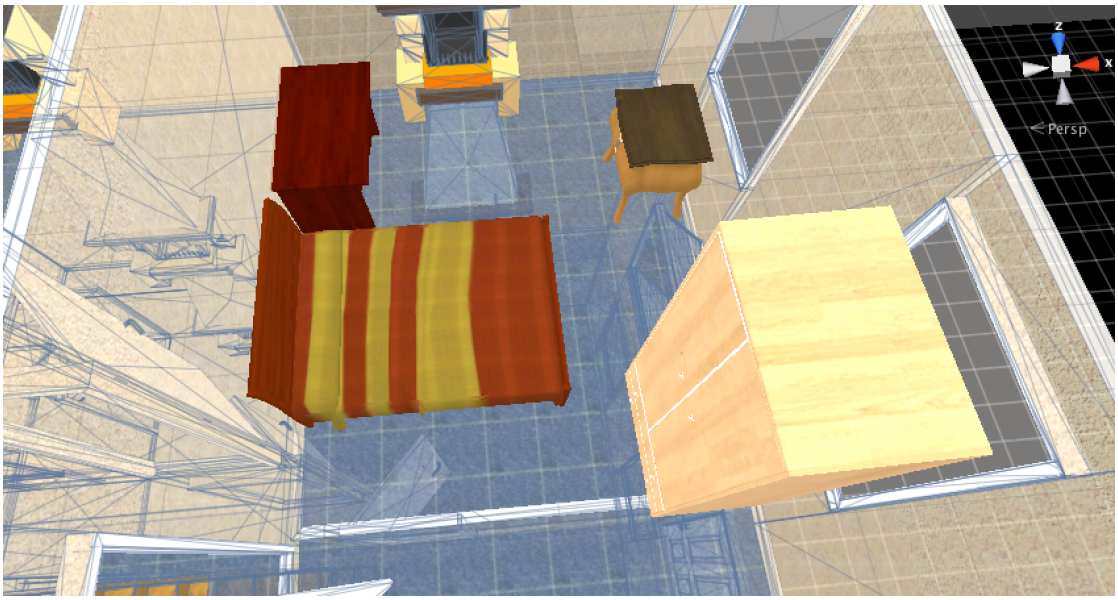

Figure 5.6: Wall-Term Optimisation 3D result

5.1.3 Doors Fireplaces, Windows Terms Optimisation

Because we have discussed elaborately in previous tests, to keep the thesis brief, this testing is on the rest three terms that with a similar expression. Firstly, we can know from Figure 5.7 that there are many global best for this testing case. Because in theory, as long as the furniture does not block the door, or closed to the fireplace, or it is high but stand in front of the window, the positions will be all accepted. Figure 5.8 and Figure 5.9 show the graphical result, which is all right. The global best will be updated if current score is equal to the history global best. Hence, this result

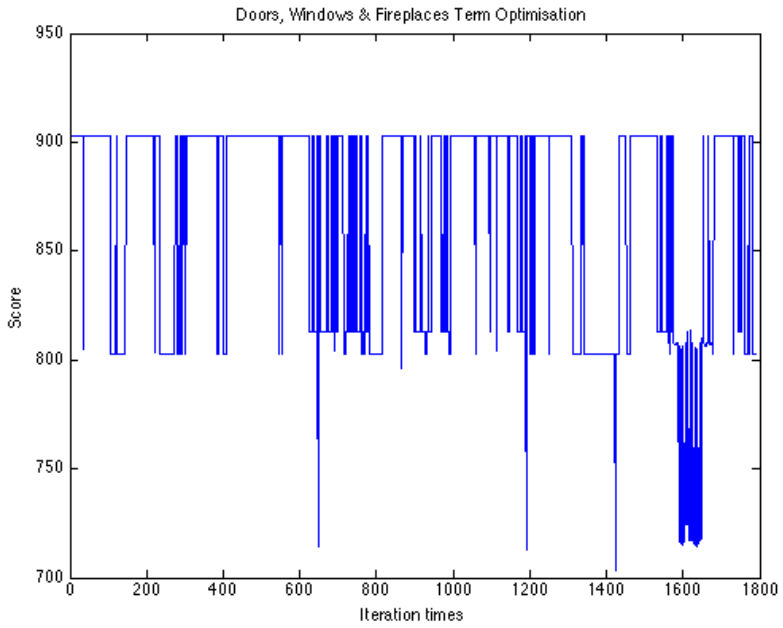

Figure 5.7: Floor Plan Furniture Optimisation 1D result

5.2 Full Terms Score Populating Optimisation

This room was chosen again since it is on top of the building and with fireplace, two windows. Although from the 2D & 3D result in Figure 5.11 and Figure 5.12, this furnishing looks no problem, from the 1D numerical result in Figure 5.10, this system has obviously not cooled down enough but the beta has done the next stage transition. That was why in theory it said for different problems the annealing temperature should be different. I would like to adjust the beta by more trails, however, the time for this thesis was about to run out. Although the system cooled

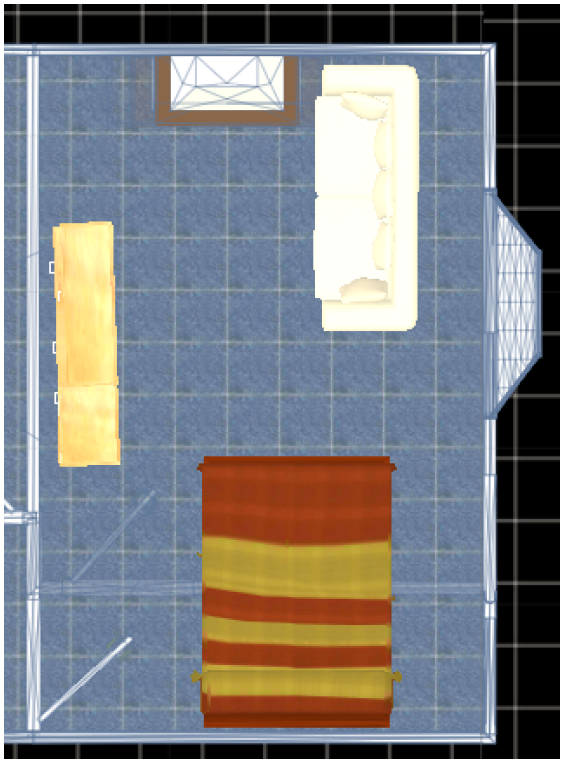

Figure 5.8: Floor Plan Furniture Optimisation 2D result

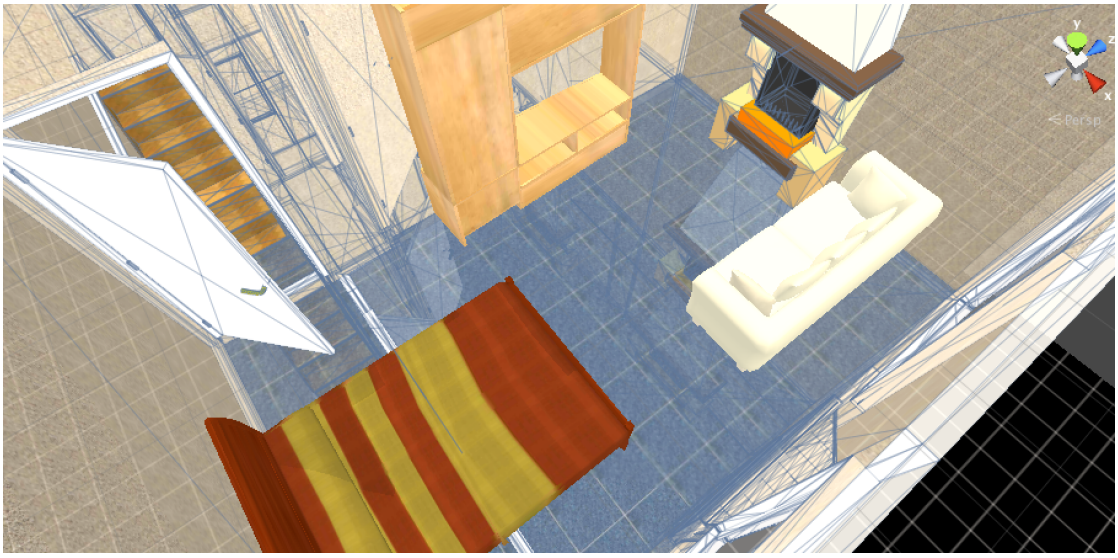

Figure 5.9: Floor Plan Furniture Optimisation 3D result

down too fast in this case, it found a acceptable global best fortunately, and all furniture are in the right facing.

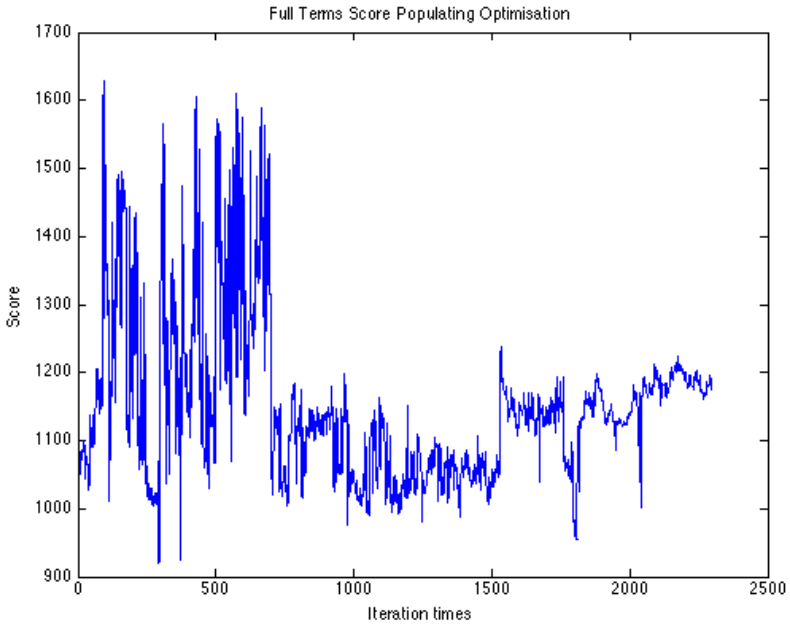

Figure 5.10: Full Terms Score Populating Optimisation 1D result

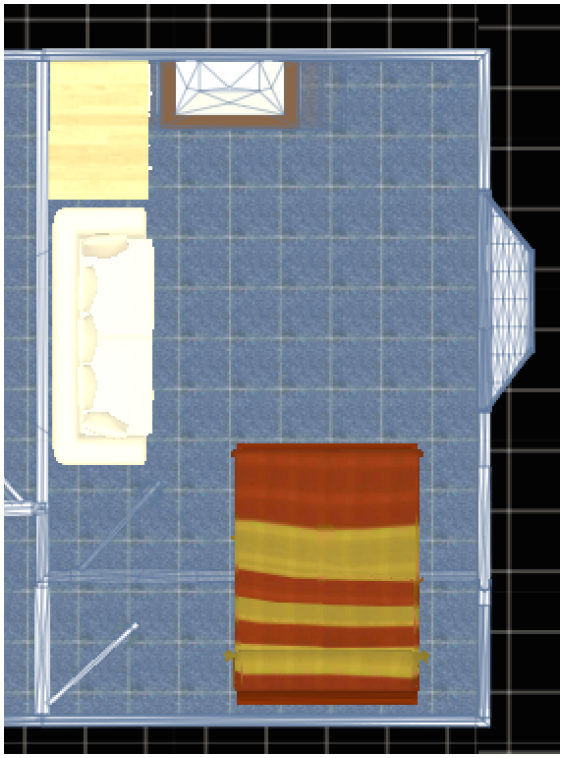

Figure 5.11: Full Terms Score Populating Optimisation 2D result

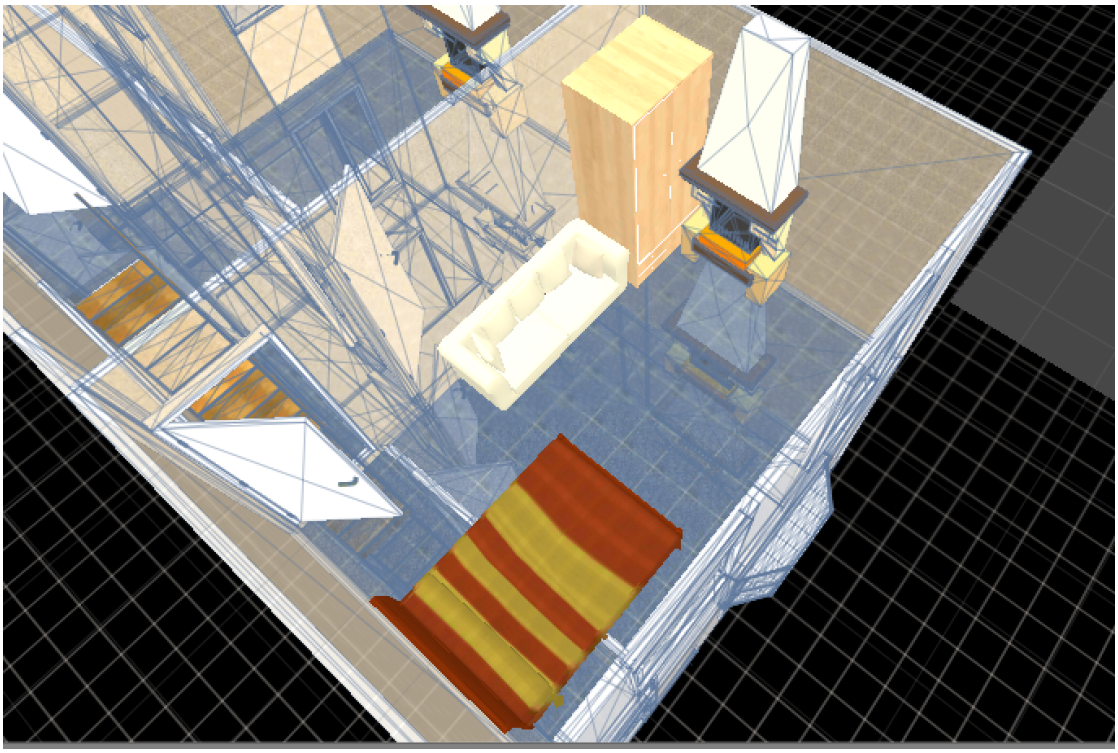

Figure 5.12: Full Terms Score Populating Optimisation 3D result

Chapter 6

Discussion and Conclusion

6.1 Discussion

6.1.1 Evaluation

This project does have a distance to what I expected due to the limitations on painful Unity3D supports on angle's computing and the short time for a project needs amounts of trials and each trial prefer "a longer cooling time" to reach its global best. It sounds sorrowful on external objectives of this project. Only first tier of furniture took part in my programme. But a good point is that it has strong compatibility on different rooms for single input "the floor mesh name", where you want to furnish. That is differently from the "Make it Home" project [5], which needs an extraction step for targeted each room, but they aimed at a different direction from me as well.

For the internal objectives, I gained two optimisation algorithms and experienced on blending them together to achieve a more randomised mechanism. And I had been quite familiar with the Unity3D's computing, and had known how to handle the problem, when the API does not support.

6.1.2 Further Work

The further work can be continue to seek the external objectives, adding the second tier furniture, supporting the switching by years, lighting and so on.

Meanwhile, when I was implementing this project, I was thinking why not to procedure modelling the house floor plan first. Thus, the data could be got directly.

6.2 Conclusion

This thesis is on automatic furnishing using Metropolis-Hastings and Simulated Annealing. The result of the project is successfully on furnishing a room with first tier furniture, which means automatic putting relevant things on those furniture have not supported. The advantage of the programme is less input but adaptive to an abroad range of home rooms.

In this thesis, it describes the problem to solve and the motivation of implementing this idea, talks about the difficulties and limitations in the progress of doing the project. In Chapter 2, it gives an exhaustive background to understand this project and the thesis. Chapter 3 shows the design of the programme, while Chapter 4 demonstrates how to implement it with disscussion on solving the problems. Chapter 5 gives the testing results with 1D numerical and 2D, 3D graphical results. It analysed why the result comes, whether it is expected, and how to improve it.

References

[1] W. Steptoe, "Msc projects: Augmented reality in cultural heritage," http://willsteptoe.com/MScProjects (Accessed on 19 March, 2014).

[2] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. Kraker, "Rule-based layout solving and its application to procedural interior generation," in *in Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS), 2009*, pp. 15–24.

[3] Unity3D, "Unity3d manual: Vector3.angle," http://docs.unity3d.com/ScriptReference/Vector3.Angle.html (Accessed on: 3rd Septemper, 2014).

[4] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Mech, and V. Koltun, "Metropolis procedural modeling." *ACM Trans. Graph.*, vol. 30, no. 2, p. 11, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/tog/tog30.html#TaltonLLDMK11

[5] L.-F. Yu, S. K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. Osher, "Make it home: automatic optimization of furniture arrangement." *ACM Trans. Graph.*, vol. 30, no. 4, p. 86, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/tog/tog30.html#YuYTTCO11

[6] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts." *ACM Trans. Graph.*, vol. 29, no. 6, p. 181, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/tog/tog29.html#MerrellSK10

[7] J. Michalek and P. Papalambros, "Interactive design optimization of architectural layouts," *Engineering Optimization*, vol. 34, no. 5, pp. 485–501, 2002. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/03052150214021

[8] T. Germer and M. Schwarz, "Procedural arrangement of furniture for real-time walkthroughs," *Computer Graphics Forum*, vol. 28, no. 8, pp. 2068–2078, 2009.

[9] T. Gruber, "Definition of ontology," *Encyclopedia of Database Systems, Springer-Verlag*, 2008. [Online]. Available: http://tomgruber.org/writing/ontology-in-encyclopedia-of-dbs.pdf

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[11] W. Hastings, "Monte carlo samping methods using markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 1970.

[12] L. Tierney, "Markov chains for exploring posterior distributions," *Annals of Statistics*, vol. 22, pp. 1701–1762, 1994.

[13] jeffheaton, "Understanding simulated annealing," Sep. 2008.

[14] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 301–308. [Online]. Available: http://doi.acm.org/10.1145/383259.383292

[15] MathWorks, "How simulated annealing works," http://www.mathworks.co.uk/help/gads/how-simulated-annealing-works.html (Accessed on: 3rd Septemper, 2014).

[16] Sweet Home 3D, http://www.sweethome3d.com/ (Accessed on: 3rd Septemper, 2014).

[17] ——, "Sweet home 3d source code version 4.4," http://sourceforge.net/projects/sweethome3d/files/SweetHome3D-source/SweetHome3D-4.4-src/ (Accessed on: 1st Septemper, 2014).

[18] Unity3D, "Unity3d api manual: Model," http://docs.unity3d.com/Manual/FBXImporter-Model.html (Accessed on: 2nd Septemper, 2014).

[19] ——, "Unity3d api manual: How do i fix the rotation of an imported model?" http://docs.unity3d.com/Manual/HOWTO-FixZAxisIsUp.html (Accessed on: 2nd Septemper, 2014).

[20] Asle, "My objects keep on shaking," http://answers.unity3d.com/questions/13937/my-objects-keep-on-shaking.html (Accessed on: 4th Septemper, 2014).

Appendices

Appendix A

Supports for Sweet Home 3D

A.1 How to Build and Run Sweet Home 3D from Source Code

Sweet Home 3D is written in Java. The source code was failed to be built by Eclipse for some technical limitations. With my supervisor Dr. Simon Julier's help, we successfully run the source code by using Ant to build and using Java virtual machines 1.6.0 (an earlier version) to run.

Firstly, make sure you have installed Ant, otherwise open the terminal and use `home brew` to install Ant:

```
brew install ant
```

Then go to the source code directory:

```
cd /Users/<yourname>/Documents/workspace/SweetHome3D
```

Copy this line to your terminal and press enter:

```
keytool -genkey -keystore keys.p12 -alias SweetHome3D -storetype pkcs12
```

Answer their questions, and then type `ant` to build it:

```
ant
```

After building the project, go to the `install` directory:

```
cd install
```

Get your installed Java 1.6.0 sdk path, and use that vertual machine to run it, for example typing:

```
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Commands/java -jar
    SweetHome3D-4.4.jar
```

Finally, after editing the source code, use terminal to rebuild the project by: (assuming under the `project/install` directory)

```
pushd ..
ant
```

Then again run the `.jar` file using 1.6.0 JVM, for example:

```
popd
/System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Commands/java -jar
    SweetHome3D-4.4.jar
```

A.2 A Brief Sweet Home 3D Guide: How to Create a Brighton House Floor Plan

Before starting this project, we visited three houses at Kensington Place [1] in Brighton. The three houses' floor plans appeared quite similar in terrace house style. They are three floors including a basement which could also be accessed from outside of the house. The basement is not entirely under the ground, which includes a garden at the rear of the house. The ground floor is elevated to a height of several stairs. Having the conceptual outline in space, a Brighton House could be started being created.

Procedures:

1. Add another two levels by menu `/Plan/Add Level`;
2. Modify levels elevations by double click corresponding tags:

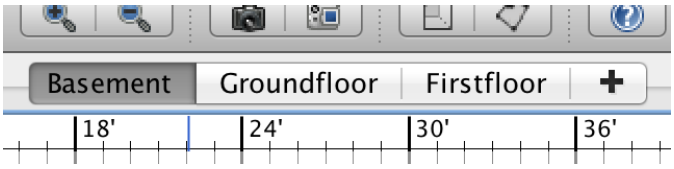

Figure A.1: Sweet Home 3D floor plan level tag

   The default elevation value for the lowest level is zero. Since Sweet Home 3D will not automatically change other upper floors' elevation values, when changing one floor's, each floor elevation value should be calculated and changed manually with the floor and ceiling thickness considered. For the Brighton House, the level attributes for the three floors are:

3. Draw the 2D floor plan in each level with the help of semitransparent outline for adjacent floor plans; Add doors, windows, stairs and fireplaces at suitable places. The floor plan used in this project is shown in Figure A.3:

[1] A street name in Brighton, UK.

Name	Elevation	Floor thickn...	Height
Basement	-5'		8'
Groundfloor	3'4¾"	0'4¾"	8'
Firstfloor	11'9½"	0'4¾"	8'

Figure A.2: The floor plan level attributes

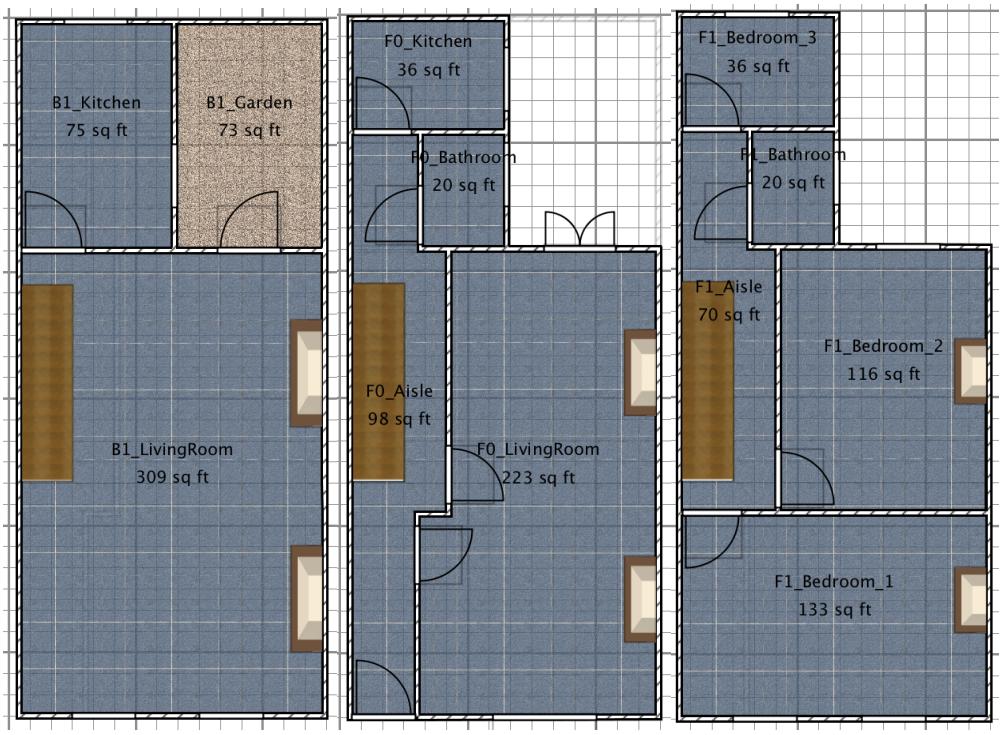

Figure A.3: Floor plan in 2D: left: basement; middle: ground floor; right: first floor

4. Press menu `3D view/Export to OBJ format...`;

   To make the inside of basement and ground floor visible for shown in figure, the wall transparency rate could be changed by in menu `3D view/Modify 3D view...`. The Brighton House 3D model in translucence is like Figure A.4:

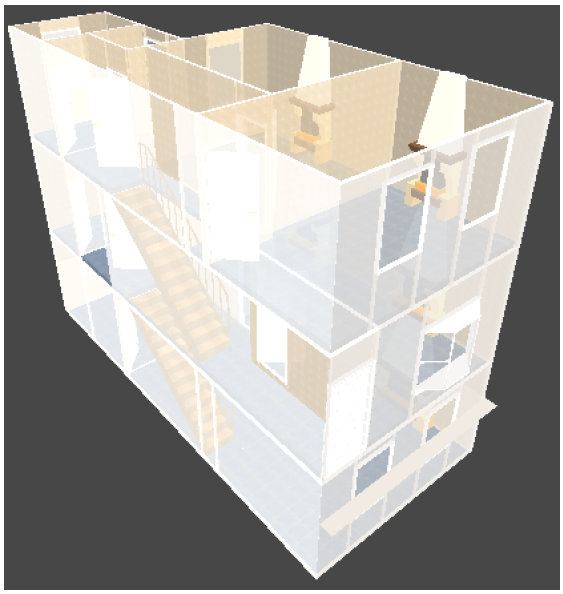

Figure A.4: Floor plan 3D model in translucence

Appendix B

Automatic Furnishing Scripts

This chapter in Appendices listing all valid code for the final version in the Unity3D project. There are still helpful code to understand the automatic furnishing system by the evolution of the core algorithm, and some scripts written for debugging and verifying the limitations on Unity3D computing supports.

The entire Unity3D project is synchronised on GitHub and shared to everybody permanently. The subversion address is `https://github.com/yuen33/AutoFurnishing_MScFinalProject`. To view the code on GitHub website, please go to `https://github.com/yuen33/AutoFurnishing_MScFinalProject`. [1]

B.1 Floor Plan Idenfication Class: `Room.cs`

Listing B.1: `Room.cs`

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Text;
4  using System.IO;
5  using System.Collections.Generic;//for List<T>, Queue<T>
6
7  public class Room : MonoBehaviour {
8    public static bool enabled=false;
9    public GameObject furnishingRoom;
10
11   public static bool isfinished =false;
12   //input
13   string floorplanName="opaqueFloorPlan";
14   //outputs
15   public static int roomID;
16   public static Vector3 roomCenter;
17   public static float roomArea;
18   public static Vector3 shiftedVector;
19   public static Vector3 roomExtents;
```

[1] `https://github.com/yuen33/AutoFurnishing_MScFinalProject` or Scan for it:

```
20  public static float roomDiagonalXZ;
21  public static Vector3[] floorCorners;
22  public static Vector3[,] walls;//(start point) (end point) (normal direction)
23
24  protected FileInfo theSourceFile = null;
25  protected StreamReader reader = null;
26  protected string text = " "; // assigned to allow first line to be read below
27
28
29  float [] FindNearestPoint(Vector3 A, Vector3[]points){
30    float [] nearest=new float[2];
31    nearest[0]=-1;
32    nearest[1]=99999;
33    float [] distances=new float[points.Length];
34    for(int i=0; i<points.Length;i++){
35      distances[i]=(points[i]-A).magnitude;
36      if(distances[i]<nearest[1]){
37        nearest[0]=i;
38        nearest[1]=distances[i];
39      }
40    }
41    return nearest;
42  }
43
44  // Update is called once per frame
45  void Update () {
46    if(enabled){
47      furnishingRoom=populatingT1.floorMesh;
48
49      roomCenter=furnishingRoom.collider.bounds.center;
50      shiftedVector=furnishingRoom.transform.position;
51      roomExtents=furnishingRoom.collider.bounds.extents;
52      roomDiagonalXZ= new Vector2(Room.roomExtents.x,Room.roomExtents.z).magnitude;
53      /**
54      * Read .txt file :
55      * to find the nearest room center
56      */
57      List<Vector3> list=new List<Vector3>();
58      theSourceFile = new FileInfo ("Assets/Autofurnishing/scripts/rooms.txt");
59      reader = theSourceFile.OpenText();
60
61      //read centers coordinates
62      text=reader.ReadLine();
63      do{
64        if(text.StartsWith("c")){
65          string [] word=text.Split(' ');
66          //word[0]="center"
67          //word[1,2,3]=<x,y,z>
68          list.Add(new Vector3(float.Parse(word[1]),
69                               float.Parse(word[2]),
70                               float.Parse(word[3])));
71        }//if text startwith
72
73        text=reader.ReadLine();
74      }while(text != null);
75      reader.Close();
76
77      //save list as a new array
78      Vector3[] centers=list.ToArray();
79      list.Clear();
80
81      //TxtCentersCoord.*0.1+UnityShifted=UnityWorldCoord
82      //so in order to make calculation less,
83      //SearchPoint=(UnityWorldCoord-shifted)*10
84      roomID=(int)(FindNearestPoint(
85        (roomCenter-GameObject.Find(floorplanName).transform.position)*10,
86        centers)[0]+1);
87      roomCenter=centers[roomID-1]*0.1f+furnishingRoom.transform.position;
88
89      /**
```

```
90      * Read txt file find Room floor all corners coordinates
91      */
92       //read corners coordinates
93       reader = theSourceFile.OpenText();
94       text=reader.ReadLine();
95       do{
96          if(text.StartsWith("R")){
97            string [] rID=text.Split(' ');
98
99            if(int.Parse(rID[1])==roomID){
100             roomArea=float.Parse(rID[2]);//RoomID <roomID> <roomArea>
101             text=reader.ReadLine();//center
102
103             text=reader.ReadLine();//corners starts
104             do{
105               string [] word=text.Split(' ');
106               Vector3 corner=new Vector3(float.Parse(word[0]),
107                                          float.Parse(word[1]),
108                                          float.Parse(word[2]));
109               corner=corner*0.1f+GameObject.Find(floorplanName).transform.position;
110                list.Add(corner);
111
112               text=reader.ReadLine();
113             }while(text!=null && !text.StartsWith("R"));
114             break;
115           }//if int (found this room beginning line)
116
117         }//if text startwith
118         text=reader.ReadLine();
119       }while(text != null);
120       reader.Close();
121
122       floorCorners=list.ToArray();
123       list.Clear();
124
125       /**
126        * Get walls infomation
127        */
128       int NumOfCorners=floorCorners.Length;
129       walls=new Vector3[NumOfCorners,3];
130       for(int i=0;i<NumOfCorners;i++){
131          //[][0] wall start point
132         walls[i,0]=floorCorners[i];
133
134          //[][1] wall end point
135         if(i+1<NumOfCorners){
136           walls[i,1]=floorCorners[i+1];
137         }else{
138           walls[i,1]=floorCorners[0];
139         }
140
141         //calculate the wall normal
142         //Unity is left hand for cross product:
143         //  http://docs.unity3d.com/ScriptReference/Vector3.Cross.html
144         //and the output wall order is clockwise
145         //in order to point inside the room:
146         //it should be Y-axis cross the wallline pointing to the end point
147         Vector3 A=walls[i,1]-walls[i,0];// pointing to the wall end point
148         Vector3 normal= Vector3.Cross(new Vector3(0,1,0),A);
149         normal=normal.normalized;
150
151         walls[i,2]=normal;
152
153       }//get all walls lines
154
155        isfinished=true;
156     }
157
158   }//Update()
159 }//class
```

B.2 Floor Plan Furniture Identification Class: `InRoomRetrie`

Listing B.2: `InRoomRetrieval.cs`

```
using System.Collections;
using System.Text;
using System.IO;
using System.Collections.Generic;//for List<T>, Queue<T>

public class InRoomRetrieval : MonoBehaviour {
  private static InRoomRetrieval instance = null;
  // to create a singleton which holds the variables
  public static InRoomRetrieval Instance{
    get{
      if(instance==null){
        instance=new GameObject("InRoomRetrieval").AddComponent<InRoomRetrieval>();
      }
      return instance;
    }
  }

  public bool isfinished ;

  //known
  //(namecode,wallID,0f) (center) (extents) Vector3(width,depth,height)
  public Vector3 [][] floorplanFurniture;

  //unknown
  public Vector3 [,] T1Furniture;//(center)(rotation)(extent)
  public Vector3 [,] T2Furniture;

  protected FileInfo theSourceFile = null;
  protected StreamReader reader = null;
  protected string text = " "; // assigned to allow first line to be read below

  // Use this for initialization
  void Start () {
    isfinished =false;
  }//Start()

  /**
   * Find x-z 2D point to point distance from Vector3 points
   */
  public static float Find2DDistance(Vector3 A, Vector3 B){
    float distance= (new Vector2(A.x,A.z)-new Vector2(B.x,B.z)).magnitude;
    return distance;
  }

  /**
   * To get which walls the floorplan furniture is on
   * it will return the wall id in Room.cs
   */
  public static int FindWall(Vector2 p){
    int NumOfWalls=Room.floorCorners.Length;
    float [] distance=new float[NumOfWalls];
    for(int i=0;i<NumOfWalls;i++){
      Vector2 B=new Vector2(Room.walls[i,0].x,Room.walls[i,0].z);
      Vector2 C=new Vector2(Room.walls[i,1].x,Room.walls[i,1].z);
      distance[i]=DistanceToRay2D(p,B,C);
    }
    return findSmallestIDAt(distance);
  }

  /**
   * get the distance from Point A to the Line of BC
   */
  public static float DistanceToRay2D(Vector2 A, Vector2 B, Vector2 C){
    Ray ray=new Ray(B,C-B);
    float distance=Vector3.Cross(ray.direction,A-B).magnitude;
```

```
66  //      distance=distance/(B-C).magnitude;
67      return distance;
68    }
69
70    /**
71     * get the smallest number id of an array
72     */
73    public static int findSmallestIDAt(float [] array){
74      float smallest=9999f;
75      int ID=-1;
76      for(int i=0;i<array.Length;i++){
77        if(array[i]<smallest){
78          smallest=array[i];
79          ID=i;
80        }
81      }
82
83      return ID;
84    }//int findsmallestIDat
85
86    // Update is called once per frame
87    void Update () {
88      if(Room.isfinished && !isfinished){
89
90        float floorElevatedHeight=(Room.roomCenter.y-Room.shiftedVector.y)*10f;
91        List<Vector3[]> list=new List<Vector3[]>();
92        /**
93       * Read .txt file :
94       * to find the this floor furniture whose
95       * 2D distance to the room center is equal/smaller than related roomextents
96       */
97        theSourceFile = new FileInfo ("Assets/Autofurnishing/scripts/floorplanFurniture.txt");
98        reader = theSourceFile.OpenText();
99
100       Vector3 center;
101       Vector3 extents;
102       text=reader.ReadLine();
103       do{
104         //each floorplanFurniture block:
105         //<elevation> "--------------"
106         //Furniture: <furniture name>
107         //Vector3 <center>
108         //Vector3 <width, depth, height>
109         string [] elevation_str =text.Split (' ');
110         if( elevation_str [1]. StartsWith("-")){
111
112           float elevation=float.Parse( elevation_str [0]) ;
113
114           if(elevation<=floorElevatedHeight+4 && elevation>=floorElevatedHeight-4 ){
115             //belong to this floor
116             text=reader.ReadLine();//name--
117             string [] word=text.Split(' ');
118             int lastOne=word.Length-1;
119             int namecode=0;
120             if(word[lastOne].Equals("door")){
121               namecode=1;
122             }else if(word[lastOne].Equals("window")){
123               namecode=2;
124             }else if(word[lastOne].Equals("Fireplace")){
125               namecode=3;
126             }
127
128             text=reader.ReadLine();//center----------------
129             string [] center_str =text.Split (' ');
130             center.x=float.Parse( center_str [0]) ;
131             center.y=float.Parse( center_str [1]) ;
132             center.z=float.Parse( center_str [2]) ;
133
134             center=center*0.1f+Room.shiftedVector;//to unity coord.
135
```

```
136             //find whether in the room
137             if(Find2DDistance(Room.roomCenter,center)<=Room.roomDiagonalXZ+2){
138               //if it is in this room
139
140               //then find which wall it belongs to:
141               int wallID=FindWall(new Vector2(center.x,center.z));
142
143
144             /**
145              * After the wallID is determined,
146              * it needs to check whether the furniture is in room
147              * (e.g. the case TWO DOORS like:
148              *   |------|
149              *   |      |
150              * _\|room  |   _\: another room's opened door
151              *   |      |
152              *   |/     |   |/: this room's opened door
153              *   -------
154              *
155              *  ^z
156              *  |-->x
157              * )
158              */
159               Vector2 x_axis= new Vector2(1,0);
160               Vector3 wallVector= Room.walls[wallID,0]-Room.walls[wallID,1];
161               Vector2 onthewall=new Vector2(wallVector.x,wallVector.y);
162
163               float angle=Vector2.Angle(x_axis,onthewall);//in degree 0 to 360(=0)
164               float max_dist;
165               float relatedAxis_dist;
166               if(angle<45 || (angle>=180 && angle<225)){
167                 //the wall is along x_axis
168                 max_dist=Room.roomExtents.z;
169                 relatedAxis_dist= Mathf.Abs(Room.roomCenter.z-center.z);
170               }else{
171                 //the wall is along z_axis
172                 max_dist=Room.roomExtents.x;
173                 relatedAxis_dist= Mathf.Abs(Room.roomCenter.x-center.x);
174               }
175               if(relatedAxis_dist>max_dist+0.4){
176                 continue;
177               }
178
179
180               text=reader.ReadLine();//width, depth, height
181               string[] size=text.Split(' ');
182               float width=float.Parse(size[0]);
183               float depth=float.Parse(size[1]);
184               float height=float.Parse(size[2]);
185               Vector3 localSize=new Vector3(width,depth,height)*0.1f;//to Unity coord. unit
186               if(angle<45 || (angle>=180 && angle<225)){
187                 //is on the wall along x_axis
188                 extents=new Vector3(width,height,depth)/2f;
189               }else{
190                 //is on the wall along z_axis
191                 extents=new Vector3(depth, height,width)/2f;
192               }
193               extents=extents*0.1f;//to Unity coord.
194
195               Vector3[] listline =new Vector3[4];
196                listline[0]=new Vector3((float)namecode,(float)wallID,0f);
197                listline[1]=center;
198                listline[2]=extents;
199                listline[3]=localSize;
200                list.Add(listline);
201
202             }//if it is in the room
203
204
205
```

```
206             }//if elevation: if the furniture is on this room floor
207           }//if elevation_str: if it is the new block first line
208
209           text=reader.ReadLine();
210         }while(text != null);
211         reader.Close();
212
213         floorplanFurniture=list.ToArray();
214         isfinished =true;
215       }//if Room.isFinished
216     }//Update()
217
218 }//Class
```

B.3 Room Type & Furniture Type Decision Class: `PopulatingGuide.cs`

Listing B.3: `PopulatingGuide.cs`

```
221 using System.Collections;
222 using System.IO;
223 using System.Collections.Generic;//for List<T>, Queue<T>
224
225 public class PopulatingGuide : MonoBehaviour {
226   private static PopulatingGuide instance = null;
227   // to create a singleton which holds the variables
228   public static PopulatingGuide Instance{
229     get{
230       if(instance==null){
231         instance=new GameObject("PopulatingGuide").AddComponent<PopulatingGuide>();
232       }
233       return instance;
234     }
235   }
236   public bool isfinished =false;
237   public string roomType;
238   public string [][] furnitureArray;
239   public double areaRank;
240
241   /**
242    * The second line: choose any one
243    * 3rd and so on: together with being populated or not being populated
244    */
245   string [][] bedroomT1={
246     new string[]{"bed","bedside_table","wardrobe","table","chair"},
247     new string[]{"shelf","armchair","bookcase"},
248     new string[]{"sofa","teatable"},
249     new string[]{"soundbox"}
250   };
251   string [][] singlebedroomT1={
252     new string[]{"bed","table","chair"},
253     new string[]{"wardrobe","bedside_table"},
254     new string[]{"bookcase","armchair"},
255     new string[]{"shelf","soundbox"}
256   };
257   string [][] livingroomT1={
258     new string[]{"tv","sofa","teatable","bookcase","dining_table"},
259     new string[]{"table","armchair"},
260     new string[]{"shelf","soundbox"},
261     new string[]{"kitchen_table"}
262   };
263   string [][] bathroomT1={
264     new string[]{},//empty
265     new string[]{"basins","toilet","washer"},//bathtub or shower
```

```
266       new string[]{"shower"},
267     };
268     string [][] kitchenT1={
269       new string[]{"kitchen_table","fridge","cupboard"},
270       new string[]{"washer","cupboard"},
271       new string[]{"dining_table"}//they are suits with chairs
272     };
273     string [][] readingroomT1={
274       new string[]{"table","chair","bookcase"},
275       new string[]{"sofa","shelf","wardrobe"},
276       new string[]{"doublechairs"}
277     };
278
279     protected FileInfo theSourceFile = null;
280     protected StreamReader reader = null;
281     protected string text = " "; // assigned to allow first line to be read below
282     List<int> list1;
283     List<int> list2;
284
285     // Use this for initialization
286     void Start () {
287       /**
288        * Read .txt file :
289        * to find the nearest room center
290        */
291       list1 =new List<int>();
292       list2 =new List<int>();
293       theSourceFile = new FileInfo ("Assets/Autofurnishing/scripts/rooms.txt");
294       reader = theSourceFile.OpenText();
295
296       //read centers coordinates
297       text=reader.ReadLine();
298       do{
299         if(text.StartsWith("RoomID")){
300           string [] word=text.Split(' ');
301           //word[0]="RoomID"
302           //word[1]=<RoomID>
303           //word[2]=<RoomArea>
304           list1.Add(int.Parse(word[1]));
305           list2.Add((int) (double.Parse(word[2])/100));
306
307         }//if text startwith
308
309         text=reader.ReadLine();
310       }while(text != null);
311       reader.Close();
312
313     }//Start()
314
315     void getFurniture(int RoomID, int[] AreaSortedRoomIDs){
316       areaRank=0;
317       for(int i=0;i<AreaSortedRoomIDs.Length;i++){
318         if(RoomID==AreaSortedRoomIDs[i]){
319           areaRank=i+1;
320           break;
321         }//if roomID is found
322       }//for
323
324       double k=areaRank/AreaSortedRoomIDs.Length;
325       areaRank=k;
326       //2.5/12=0.208333
327       //7.5/12=0.625
328       //10.5/12=0.875
329       if(k<0.208){//bathroom
330         roomType="bathroom";
331         furnitureArray=bathroomT1;
332         //Debug.Log("is_bathroom");
333
334       }else if(k>0.875){//livingroom (+)
335         roomType="livingroom";
```

```
336         furnitureArray=livingroomT1;
337         //Debug.Log("is_living_room");
338 
339       }else if(k>0.625){//bedroom
340         if(Random.value>0.8){
341           roomType="singlebedroom";
342           furnitureArray=singlebedroomT1;
343           //Debug.Log("is_single_bedroom");
344         }else{
345           roomType="bedroom";
346           furnitureArray=bedroomT1;
347           //Debug.Log("is_bedroom");
348         }//if random else
349 
350       }else{//single bedroom/reading room/kitchen
351         switch((int)(Random.value*3)%3){
352         case 0: //can be 0...0 and 1...0 (I hope singlebedroom can be more than the others)
353           roomType="singlebedroom";
354           furnitureArray=singlebedroomT1;
355           //Debug.Log("is_single_bedroom");
356           break;
357 
358         case 1: //only can be 0...1
359           roomType="kitchen";
360           furnitureArray=kitchenT1;
361           //Debug.Log("is_kitchen");
362           break;
363 
364         default://only can be 0...2
365           roomType="readingroom";
366           furnitureArray=readingroomT1;
367           //Debug.Log("is_reading_room");
368           break;
369         }//switch
370       }//if else if ...
371     }//void getRoomType()
372 
373     // Update is called once per frame
374     void Update () {
375       if(Room.isfinished && !isfinished){
376         //save list as a new array
377         int [] AreaSortedRoomIDs=list1.ToArray();
378         int [] RoomAreas=list2.ToArray();
379         System.Array.Sort(RoomAreas,AreaSortedRoomIDs);
380         list1.Clear();
381         list2.Clear();
382 
383         getFurniture(Room.roomID,AreaSortedRoomIDs);
384 
385         isfinished =true;
386       }//if Room.isfinished
387     }//Update()
388 }
```

B.4 Furniture Populating Optimisation System Code: `populatingT1.cs`

Listing B.4: `populatingT1.cs`

```
389 using System.Collections;
390 using System.Collections.Generic;//for List<T>, Queue<T>
391 using System.IO;
392 
393 public class populatingT1 : MonoBehaviour {
```

```
394   float isFullFactor=0.3f;
395   //for testing
396   string path;
397   string filename;
398   System.IO.StreamWriter file;
399   public int sumOfIterations=0;
400   //Input:
401   public static string floorName;
402   public static GameObject floorMesh;
403
404   //--------------------Do not support many-to-one
          ----------------------------
405   //e.g. "table_to_chair", "tv_to_chair" is not valid
406   public string [][] pairwiseTAG={
407     new string[]{"table","chair"},//on Face 2
408     new string[]{"tv","sofa"},//on Face 2, as far as possible
409     new string[]{"sofa","teatable"},//on Face 2
410     new string[]{"bed","bedside_table"}, //on Face 3 or 4, and (itself) Face 5 along wall
411     new string[]{"single_bed","bedside_table"}//not all bed needs a bedsidetable ...(as above)
412   };
413   //-----------------------------------------------
414   public bool isInitialised =false;
415   public bool isStable=false;
416   public static bool isfinished=false;
417
418   List<string> nameList;
419   List<string> secondaryPairList;//{its primary pair name, its name}
420   List<string> primaryPairList;
421   public static Vector3[,] globalBestT1;//(center)(rotation)(extents) for global best record
422   Vector3[,] lastPosition;
423   public static List<GameObject> boxesT1;
424
425   Vector3 [][] Doors;//(namecode,wallID,0)(center)(extents) Vector3(width,depth,height)
426   Vector3 [][] Fireplaces;//(namecode,wallID,0)(center)(extents) Vector3(width,depth,height)
427   Vector3 [][] Windows;//(namecode,wallID,0)(center)(extents) Vector3(width,depth,height)
428
429   int populatingState=0;//max=num Of Populated furniture
430
431   public double globalBest=-999999;
432   public double currentDistanceScore;
433   double lastDistanceScore;
434   public double distanceScoreDifference;
435
436   public double currentSumOfScores;
437   double lastSumOfScores=-99999;
438   public double sumOfScoresDifference;
439
440   double[] lastSingleScores;
441   public double[] currentSingleScores;
442   public float step=1f;
443   public double beta=0.1f;
444   public int[] iteration;
445   double distanceFactor=1;
446   Vector3 floorplanRotation;
447
448   // Use this for initialization
449   void Start () {
450   /** Rooms to be furnished: (tested roomtype example result)
451    * F1:
452    * room_4_646 ->bedroom
453    * room_3_644 ->bedroom
454    * room_2_642 ->bathroom
455    * room_1_640 ->*reading room*
456    *
457    * G0:
458    * room_8_653 is living room
459    * room_11_659 is bathroom
460    * room_12_661 is *reading room*
461    *
462    * B1:
```

```
463     * room_10_657 is living room
464     * room_5_648 is *kitchen*
465     *
466     * **:can be single bedroom/kitchen/reading room
467     */
468      floorName="room_4_646";
469      floorMesh=GameObject.Find(floorName);
470      Room.enabled=true;
471      boxesT1=new List<GameObject>();
472 //     Random.seed=Room.roomID;
473      floorplanRotation=gameObject.transform.eulerAngles;
474      path="Assets/Autofurnishing/scripts/";
475      filename=floorName+"_NumericalResults.txt";
476      // Write the string to a file.
477       file = new System.IO.StreamWriter(path+filename);
478    }
479
480    // Update is called once per frame
481    void Update () {
482       if(! isInitialised && PopulatingGuide.Instance.isfinished
483          && InRoomRetrieval.Instance.isfinished){
484          initialization ();
485 //         Time.timeScale=0;//pause the game
486 //         getScore();
487         getOverallScore();
488 //         Time.timeScale=1;//unpause the game
489         //-----End-----
490          isInitialised =true;
491         lastSumOfScores=currentDistanceScore;
492
493          iteration=new int[nameList.Count];
494       }//if(PopulatingGuide.Instance.isfinished
495
496       if( isInitialised && !isStable){
497         lastDistanceScore=currentDistanceScore;
498         getOverallScore();
499          if(Mathf.Abs((float)(currentDistanceScore-lastDistanceScore))<step){
500            isStable=true;
501          }
502       }
503       if(isStable && !isfinished){
504         recordLastPosition();
505         simulatedAnnealingControl();
506         //move
507         for(int i=0;i<populatingState;i++){
508            if(Random.value>0.995){
509              int k=populatingState-1;
510              int idx=Mathf.FloorToInt(Random.value*k %k)+1;//won't be the biggest cube
511              jumpOne(idx);
512            }
513            if(!isInRoom(boxesT1[i])){
514              moveintotheRoom(boxesT1.IndexOf(boxesT1[i]));
515            }else{
516              rotate(boxesT1[i]);
517              move(boxesT1[i],i);
518            }
519         }
520         /**
521          * Scores handling
522          */
523         lastSingleScores=new double[populatingState];
524         if(populatingState>currentSingleScores.Length){
525            for(int i=0;i<populatingState;i++){
526              if(i<populatingState-1){
527                lastSingleScores[i]=currentSingleScores[i];
528              }else{
529                //i=populatingState-1
530                lastSingleScores[i]=0;
531              }
532            }
```

```
533         }else{
534           System.Array.Copy(currentSingleScores,lastSingleScores,populatingState);
535         }
536         getOverallScore();
537         //compare with globalbest
538         if(currentSumOfScores>=globalBest){
539           globalBest=currentSumOfScores;
540           recordToGlobalBest();
541         }
542
543         MetropolisHastings();
544
545         distanceScoreDifference=currentDistanceScore-lastDistanceScore;
546         sumOfScoresDifference=currentSumOfScores-lastSumOfScores;
547
548         lastSumOfScores=currentSumOfScores;
549         isStable=false;
550       }
551
552       if(Input.GetKeyDown(KeyCode.S)){
553         if(Time.timeScale==1){
554           Time.timeScale=0;
555           isStable=false;
556         }else{
557           Time.timeScale=1;
558         }
559       }
560     }//Update()
561
562     void MetropolisHastings(){
563       sumOfIterations++;
564       writeline(sumOfIterations,currentSumOfScores);
565
566       //Metropolis-Hastings
567       double MHDelta=beta*(currentSumOfScores-lastSumOfScores);
568       float lnp= Mathf.Log(Random.value);
569
570       if(MHDelta < lnp){
571 //        sumOfIterations--;
572         iteration[populatingState-1]--;
573         Debug.Log("lnp="+lnp);
574         Debug.Log("beta*(currentSumOfScores-lastSumOfScores)="+ MHDelta);
575         for(int i=0;i<populatingState;i++){
576           boxesT1[i].transform.position=lastPosition[i,0];
577           boxesT1[i].transform.localEulerAngles=lastPosition[i,1];
578         }
579       }
580     }
581
582     void recordToGlobalBest(){
583 //      globalBestT1=new Vector3[nameList.Count,3];
584       for(int i=0;i<populatingState;i++){
585         globalBestT1[i,0]=boxesT1[i].collider.bounds.center;
586         globalBestT1[i,1]=new Vector3(0,0,0);
587         globalBestT1[i,1].y=boxesT1[i].transform.localEulerAngles.y;
588         globalBestT1[i,2]=boxesT1[i].collider.bounds.extents;
589       }
590     }
591
592     void recordLastPosition(){
593       //Record position
594       lastPosition=new Vector3[populatingState,3];
595       Vector3[] entity=new Vector3[3];
596       entity[0]=new Vector3(0,0,0);
597       entity[1]=new Vector3(0,0,0);
598       entity[2]=new Vector3(0,0,0);
599       for(int i=0;i<populatingState;i++){
600         entity[0]=boxesT1[i].transform.position;
601         entity[1].y=boxesT1[i].transform.localEulerAngles.y;
602         entity[2]=boxesT1[i].collider.bounds.extents;
```

```
603         lastPosition [i,0]=entity [0];
604         lastPosition [i,1]=entity [1];
605         lastPosition [i,2]=entity [2];
606       }
607     }
608
609     void simulatedAnnealingControl(){
610       double iterationTimesCritaria=populatingState*300;
611
612       if(sumOfIterations>iterationTimesCritaria){
613
614         if(sumOfIterations>2.5*iterationTimesCritaria) {
615           beta=10000;
616           step=(float)(step*0.995);
617         }else if(sumOfIterations>2*iterationTimesCritaria) {
618           beta=1000;step=0.5f;
619         }else if(sumOfIterations>iterationTimesCritaria) beta=1;
620       }
621
622       if(step<0.1){
623         goToGlobalBest();
624          isfinished =true;
625       }
626     }
627
628
629
630
631     //============================goToGlobalBest()====================
632     void goToGlobalBest(){
633       for(int  i=0;i<populatingState;i++){
634         boxesT1[i].transform.position=globalBestT1[i,0];
635         boxesT1[i].transform.localEulerAngles=globalBestT1[i,1];
636       }
637     }
638
639     //============================jumpOne()=====================
640     void jumpOne(int idx){
641       Vector3 newposition=getRandomPosition(idx);
642       newposition.y=boxesT1[idx].collider.bounds.extents.y+Room.roomCenter.y;
643       boxesT1[idx].transform.position=newposition;
644     }
645
646     //============================switch()=====================
647     void switchAnyTwo(){
648       //switch two objects randomly
649       int []  RandomList=getRandomList(populatingState);
650       int  a=RandomList[0];
651       int  b=RandomList[1];
652
653       //switch two objects position
654       boxesT1[a].transform.position=lastPosition[b,0];
655       boxesT1[b].transform.position=lastPosition[a,0];
656     }
657
658
659
660     //============================rotate()=====================
661     void  rotate(GameObject furniture){
662
663       Vector3 Pi=furniture.collider.bounds.center;
664       Vector3 Ei=furniture.collider.bounds.extents;
665       int  wallID=InRoomRetrieval.FindWall(new Vector2(Pi.x,Pi.z));
666
667       float  walldistance=InRoomRetrieval.DistanceToRay2D(
668         new Vector2(Pi.x,Pi.z),
669         new Vector2(Room.walls[wallID,0].x,Room.walls[wallID,0].z),
670         new Vector2(Room.walls[wallID,1].x,Room.walls[wallID,1].z));
671
672       Vector3 from=new Vector3(0,0,1);
```

```
673       Vector3 to=Room.walls[wallID,2];
674       float rotationY=Vector3.Angle(from,to);
675       rotationY=rotationY;//-floorplanRotation.y;
676
677       if(Pi.x<Room.roomCenter.x && Pi.z<Room.roomCenter.z){
678         //in III phase
679         if(rotationY<10){
680           rotationY=0;
681         }
682         if(rotationY>70){rotationY=90;}
683       }else if(Pi.x<Room.roomCenter.x && Pi.z>Room.roomCenter.z){
684         //in II phase
685         if(rotationY>70){
686           rotationY=90;
687         }
688         if(rotationY<10){
689           rotationY=180;
690         }
691       }else if(Pi.x>Room.roomCenter.x && Pi.z>Room.roomCenter.z){
692         //in I phase
693         if(rotationY<10){
694           rotationY=180;
695         }
696         if(rotationY>70){
697           rotationY=270;
698         }
699       }else{
700         //in IV phase
701         if(rotationY<10){
702           rotationY=0;
703         }
704         if(rotationY>70){
705           rotationY=270;
706         }
707
708       }
709
710       if(Ei.z< Ei.x || walldistance >Ei.z){
711         furniture.transform.localEulerAngles=new Vector3(0,rotationY,0);
712       }
713
714     }
715
716     }
717     //-------------------------move()-----------------------------
718
719     bool isInRoom(GameObject box){
720       Vector3 center=box.collider.bounds.center;
721       center.y=floorMesh.collider.bounds.center.y;
722       return floorMesh.collider.bounds.Contains(center);
723     }
724
725     int findNearestCornerID(Vector3 Pi){
726       int wallID_Pi=InRoomRetrieval.FindWall(new Vector2(Pi.x,Pi.z));
727       //find nearest corner
728       float [] distances2D=new float[Room.floorCorners.Length];
729       for(int j=0;j<Room.floorCorners.Length;j++){
730         distances2D[j]=(new Vector2(Pi.x,Pi.z)-
731                         new Vector2(Room.floorCorners[j].x,
732                     Room.floorCorners[j].z)).magnitude;
733       }
734       int cornerID_Pi=InRoomRetrieval.findSmallestIDAt(distances2D);
735       return cornerID_Pi;
736     }
737
738     //-------------------------getScore()-----------------------------
739     void getOverallScore(){
740 //      distanceFactor=10*nameList.Count/populatingState;
741       getAllDistanceScore();
742       currentSumOfScores=currentDistanceScore;
```

```
743       currentSingleScores=new double[populatingState];
744       for(int i=0;i<populatingState;i++){
745         getSingleScore(i);
746         currentSumOfScores+=currentSingleScores[i];
747       }
748
749     }
750
751     void getAllDistanceScore(){
752       double SumsumOfDistance0=0;
753       double SumsumOfDistance1=0;// large= :)
754       double SumsumOfDistance2=0;// large= :)
755
756       for(int i=0;i<populatingState;i++){
757         Vector3 Pi=boxesT1[i].collider.bounds.center;
758         for(int j=i+1;j<populatingState;j++){
759           Vector3 Pj=boxesT1[j].collider.bounds.center;
760           float distance=(new Vector2(Pi.x,Pi.z)-new Vector2(Pj.x,Pj.z)).magnitude;
761           if(!isInRoom(boxesT1[i])) distance=0;
762           SumsumOfDistance0+=distance;
763         }
764       }
765
766       currentDistanceScore=distanceFactor*SumsumOfDistance0;
767
768 //      Debug.Log("    SumsumOfDistance="+SumsumOfDistance);
769     }//getDistanceScore()
770
771     void getSingleScore(int i){
772       /**
773        * Door, windows and fireplace term:
774        * Vector3 [][] Doors windows Fireplaces: (namecode,wallID,0)(center)(extents) Vector3(width,depth,height)
775        * Vector3[,] walls: (start point)(end point)(normal to inside the room)
776        */
777       /**
778        * Nearest wall and corner term
779        */
780       Vector3 Pi=boxesT1[i].transform.position;
781       Vector3 Ei=boxesT1[i].collider.bounds.extents;
782
783       Vector2 A=new Vector2(Pi.x,Pi.z);
784       int wallID=InRoomRetrieval.FindWall(A);
785       int cornerID=findNearestCornerID(new Vector3(A.x,0,A.y));
786
787       float walldistance=InRoomRetrieval.DistanceToRay2D(
788         A,
789         new Vector2(Room.walls[wallID,0].x,Room.walls[wallID,0].z),
790         new Vector2(Room.walls[wallID,1].x,Room.walls[wallID,1].z))
791         -Ei.z;
792       walldistance=Mathf.Max(walldistance,0);
793       float cornerdistance=(new Vector2(Room.floorCorners[cornerID].x,
794                                         Room.floorCorners[cornerID].z)
795                             -A).magnitude-new Vector2(Ei.x,Ei.z).magnitude;
796       cornerdistance=Mathf.Max(cornerdistance,0);
797
798       double NearestWallDistanceScore=100/(walldistance+1)/(10*i+1)*Ei.y;
799       double NearestCornerDistanceScore=100/(cornerdistance+1)/(10*i+1)*Ei.y;
800       /**
801        * Rotation check
802        */
803       float rotationY=boxesT1[i].transform.localEulerAngles.y;
804       float targetedRY=Vector3.Angle(new Vector3(0,0,1),Room.walls[wallID,2]);
805       if(Mathf.Abs(rotationY-targetedRY)>45){
806         NearestWallDistanceScore=NearestWallDistanceScore/2;
807       }
808
809       /**
810        * Window shielded score
811        */
812       double WindowShieldedScore=0;
```

```
813         for(int j=0;j<Windows.GetLength(0);j++){
814           float cosTheta=0;
815           float windowDistance=0;
816           int windowWall=(int)Windows[j][0].y;
817           //if the box nearest wall is the window wall
818           if(wallID==windowWall){
819             //the box should not be higher than the window:
820             //windowcenter.y-boxcenter.y
821             float heightDelta=Windows[j][1].y-Pi.y;
822             //>0 and even > sum of two extents in Y
823             //alow furniture that a litter higher than window lowest bounds
824             if(heightDelta<Windows[j][2].y*0.3f+Ei.y){
825               //cos(theta)
826               Vector3 pointingtoPi=Pi-Windows[j][1];//pointing to Pi
827               pointingtoPi.y=0;
828               //----2D distance
829               windowDistance=pointingtoPi.magnitude;
830               windowDistance-=Mathf.Max(boxesT1[i].collider.bounds.extents.x,
831                                         boxesT1[i].collider.bounds.extents.z);
832               windowDistance=Mathf.Max(windowDistance,0);
833
834               pointingtoPi=pointingtoPi.normalized;//and normalise it
835               //the normalized wall normal * pointingtoPi
836               cosTheta=Vector3.Dot(Room.walls[windowWall,2],pointingtoPi);
837             }//if box is lower than the window, it doesn't matter
838             else{
839               WindowShieldedScore=100;
840             }
841           }//if the box is not near to the window, it's doesn't matter too
842           else{
843             WindowShieldedScore=100;
844           }
845
846           WindowShieldedScore+=windowDistance/(10*cosTheta+1);
847
848           if(cosTheta>0.71){//if cos(theta)>1/sqrt(2)
849             WindowShieldedScore=WindowShieldedScore+(windowDistance+1)/(cosTheta+1)
850               -10/(windowDistance+1);
851           }
852         }
853
854         /**
855          * Fireplace shielded score
856          */
857         double FireplaceShieldedScore=0;
858         for(int j=0;j<Fireplaces.GetLength(0);j++){
859           float cosTheta=0;//when theta= 90 degree, very important
860           float FireplaceDistance=0;
861           int fireplaceWall=(int)Fireplaces[j][0].y;
862           //if the box nearest wall is the fireplace wall
863           if(wallID==fireplaceWall){
864             //cos(theta)
865             Vector3 pointingtoPi=Pi-Fireplaces[j][1];//pointing to Pi
866             pointingtoPi.y=0;
867             //----2D distance
868             FireplaceDistance=pointingtoPi.magnitude;
869             FireplaceDistance-=Mathf.Max(boxesT1[i].collider.bounds.extents.x,
870                                          boxesT1[i].collider.bounds.extents.z);
871             FireplaceDistance=Mathf.Max(FireplaceDistance,0);
872             //if center 2d distance larger than fireplace depth*1.5
873             if(FireplaceDistance<=Fireplaces[j][3].y*1.5f){
874               pointingtoPi=pointingtoPi.normalized;//and normalise it
875               //the normalized wall normal * pointingtoPi
876               cosTheta=Vector3.Dot(Room.walls[fireplaceWall,2],pointingtoPi);
877             }//else ignore
878             else{
879               FireplaceShieldedScore=100;
880             }
881           }//else ignore
882           else{
```

```
883             FireplaceShieldedScore=100;
884           }
885
886           FireplaceShieldedScore+=(FireplaceDistance+1)/(cosTheta+1);
887           if(cosTheta>0.87){//expected narrow than window
888             FireplaceShieldedScore=FireplaceShieldedScore- FireplaceDistance/(cosTheta+1)
889               -10/(FireplaceDistance+1);
890           }
891         }
892
893         /**
894          * Door path score
895          */
896         double DoorPathScore=0;
897         for(int j=0;j<Doors.GetLength(0);j++){
898           float cosTheta=0;
899           float DoorDistance=0;
900           int doorCorner=findNearestCornerID(Doors[j][1]);
901           if(cornerID==doorCorner){
902             NearestCornerDistanceScore=0;
903 //            Debug.Log(boxesT1[i].name+" near door");
904             int doorWall=(int)Doors[j][0].y;
905             //cos(theta)
906             Vector3 pointingtoPi=Pi-Doors[j][1];//pointing to Pi
907             pointingtoPi.y=0;
908             //----2D distance
909             float doorDistance=pointingtoPi.magnitude;
910             doorDistance-=Mathf.Max(boxesT1[i].collider.bounds.extents.x,
911                                     boxesT1[i]. collider .bounds.extents.z);
912             doorDistance=Mathf.Max(doorDistance,0);
913             pointingtoPi=pointingtoPi.normalized;//then normalise it
914             //the normalized wall normal * pointingtoPi
915             cosTheta=Vector3.Dot(Room.walls[doorWall,2],pointingtoPi);
916           }//else it should move far away from doors
917           else{
918             DoorPathScore=100;
919           }
920           float theta=Mathf.Acos(cosTheta)*180/Mathf.PI;
921           DoorPathScore+=DoorDistance*theta;
922 //          DoorPathScore+=(DoorDistance+1)/(cosTheta+1);
923           if(cosTheta>0.8){//expected wider than window
924             DoorPathScore=DoorPathScore- //DoorDistance/(cosTheta+1)
925               -10/(DoorDistance+1);
926           }
927         }
928         currentSingleScores[i]=
929           NearestWallDistanceScore+NearestCornerDistanceScore
930             +DoorPathScore+WindowShieldedScore+FireplaceShieldedScore;
931
932       }//getSingleScore()
933       void initialization (){
934         getKnownFloorplanFurniture();
935         determineInitialFurniture();
936         /**
937          * import the furniture from name
938          */
939         int i=0;
940         foreach(string name in nameList){
941           if(isFull()) break;
942           i++;
943           importInitialFurniture(name);
944         }
945       }
946       //------------------------initialization()-------------------------
947
948       void determineInitialFurniture(){//is a big room of the house
949         nameList=new List<string>();
950         /**
951          * get all initially imported furniture names
952          */
```

```
953     //add all first line furniture
954     string [][] furarray=PopulatingGuide.Instance.furnitureArray;
955     int NumOfItems=furarray[0].Length;
956     for(int i=0;i<NumOfItems;i++){
957       nameList.Add(furarray[0][i]);
958       Debug.Log("Furniture tags add: "+furarray[0][i]);
959     }
960
961     //add other furniture randomly
962     int NumOfRows=furarray.GetLength(0);
963     double rank=PopulatingGuide.Instance.areaRank;
964 //    Debug.LogError(rank);
965     if(nameList.Count==0){
966       //add all furniture from one of rest lines
967       if(NumOfRows>1){
968         int restrows=NumOfRows-1;
969         int rowth=1+ Mathf.FloorToInt(Random.value *restrows %restrows);
970         //Mathf.CeilToInt will exceed idx range
971         foreach(string element in furarray[rowth]){
972           Debug.Log("Furniture tags add: "+element);
973
974           nameList.Add(element);
975         }
976       }//if numofrow>1
977
978     }else if(rank>0.5){//is a large room of the house
979
980       //add any one in 2nd line
981       NumOfItems=furarray[1].Length;
982       int idx=Mathf.FloorToInt(Random.value*NumOfItems %NumOfItems);
983       nameList.Add(furarray[1][idx]);
984       Debug.Log("Furniture tags add: "+furarray[1][idx]);
985
986       //add all furniture from one of rest lines
987       if(NumOfRows>2){
988         int restrows=NumOfRows-2;
989         int rowth=2+ Mathf.FloorToInt(Random.value *restrows %restrows);
990         //Mathf.CeilToInt will exceed idx range
991         foreach(string element in furarray[rowth]){
992           nameList.Add(element);
993           Debug.Log("Furniture tags add: "+element);
994
995         }
996       }//if numofrow>2
997
998     }else if(rank>0.25){//is a small room of the house
999       //any one of the rest
1000      int restrow=NumOfRows-1;
1001      int rowth=1+Mathf.FloorToInt(Random.value *restrow %restrow);
1002      NumOfItems=furarray[rowth].Length;
1003      int idx=Mathf.FloorToInt(Random.value*NumOfItems %NumOfItems);
1004      nameList.Add(furarray[rowth][idx]);
1005      Debug.Log("Furniture tags add: "+furarray[rowth][idx]);
1006
1007    }//else: rank<0.25 add nothing more
1008
1009
1010    /**
1011     * Sort these initial furniture by volumn==> change to by X-Z 2D area
1012     */
1013    Vector3[] extents=new Vector3[nameList.Count];
1014    double[] areas=new double[nameList.Count];
1015    int [] indiceT1=new int[nameList.Count];
1016    for(int i=0;i<nameList.Count;i++){
1017      //Find objects with the tag in name, and choose any one of them
1018      Debug.Log("Import furniture with tag="+nameList[i]);
1019      GameObject[] gos;
1020      gos = GameObject.FindGameObjectsWithTag(nameList[i]);
1021      int idx=Mathf.FloorToInt(Random.value* gos.Length %gos.Length);
1022      nameList[i]=gos[idx].name;
```

```
1023          Vector3[] array=GetVerticesInChildren(gos[idx]);
1024          extents[i]=GetFurnitureExtents(array);
1025          areas[i]=4*extents[i].x *extents[i].z;
1026          indiceT1[i]=i;
1027        }
1028        System.Array.Sort(areas,indiceT1);
1029
1030        string [] names=nameList.ToArray();
1031        nameList.Clear();
1032
1033  //      foreach(int i in indiceT1) Debug.Log(i);
1034
1035        //(center)(rotation)(extents)
1036        globalBestT1=new Vector3[indiceT1.Length,3];
1037        int counter=0;
1038        for(int i=indiceT1.Length-1;i>=0;i--){
1039          //I wish to make the larger furniture has positioning priority
1040          globalBestT1[counter,0]=new Vector3(0,0,0);
1041          globalBestT1[counter,1]=new Vector3(0,0,0);
1042          globalBestT1[counter,2]=extents[indiceT1[i]];
1043          counter++;
1044          nameList.Add(names[indiceT1[i]]);
1045        }
1046
1047
1048      }//determineFurniture()
1049
1050
1051      //-----------------------------------------------------------------------------

1052      bool isFull(){
1053        double occupiedArea=0;
1054        foreach(GameObject cube in boxesT1){
1055          occupiedArea+=cube.collider.bounds.extents.x *cube.collider.bounds.extents.z;
1056        }
1057        Vector3 floorMeshEx=floorMesh.collider.bounds.extents;
1058  //      Debug.LogWarning("floorMesh extents="+floorMeshEx);
1059        double critariaArea=floorMeshEx.x*floorMeshEx.z*isFullFactor;
1060
1061        Debug.Log(occupiedArea+" < "+critariaArea+" ?");
1062
1063        if(occupiedArea>critariaArea){
1064          Debug.Log("Room is full");
1065          return true;
1066        }else{
1067          Debug.Log("Room is not full");
1068          return false;
1069        }
1070      }
1071
1072      void importInitialFurniture(string name){
1073        if(populatingState<10){
1074          /**
1075           * Make sure the decided furniture is not too big for the room
1076           */
1077          Vector3 position=new Vector3(0,0,0);
1078          float rotationY=0;
1079          //1. half diagonal lines comparing
1080          Vector3 PiEx=globalBestT1[populatingState,2];
1081          Vector3 RoEx=floorMesh.collider.bounds.extents;
1082          // first check whether there is a wall is long enough
1083          float [] wallLengths=new float[Room.walls.GetLength(0)];
1084          int [] indices=new int[wallLengths.Length];
1085          for(int i=0;i<wallLengths.Length;i++){
1086            //wall's starting point - ending point
1087            wallLengths[i]=(Room.walls[i,0]-Room.walls[i,1]).magnitude;
1088            indices[i]=i;
1089          }
1090          System.Array.Sort(wallLengths,indices);//from shortest to largest
1091          if(Mathf.Max(PiEx.x,PiEx.z)>Mathf.Max(RoEx.x,RoEx.z) ||
```

```
1092            Mathf.Min(PiEx.x,PiEx.z)>Mathf.Min(RoEx.x,RoEx.z) ||
1093            new Vector2(PiEx.x,PiEx.z).magnitude>Room.roomDiagonalXZ ||
1094            wallLengths[wallLengths.Length-1]<PiEx.x*2f){
1095          //last one: the longest wall is shorter than this furniture
1096          //it will waste too many space even if it could be imported
1097          Debug.LogError("Unexpected situation: Furniture "+name+"is too large for this room");
1098          nameList.Remove(name);
1099          return;
1100        }else if(new Vector2(PiEx.x,PiEx.z).magnitude>=Mathf.Max(RoEx.x,RoEx.z)-1){
1101          Debug.LogWarning(name+" is huge box");
1102          //extremly big furniture but importable with fixed suitable rotation
1103          //e.g. a kitchen oven ect. table can fill half kitchen
1104          //put it in center and with longest wall normal rotation
1105          position=new Vector3(0,PiEx.y,0)+Room.roomCenter;
1106          rotationY=Vector3.Angle(new Vector3(0,0,0),
1107                                  Room.walls[indices[indices.Length-1],2]);
1108          GameObject huge;
1109          huge=GameObject.CreatePrimitive(PrimitiveType.Cube);
1110          huge.transform.localScale=PiEx*2;//extentes
1111  //          Debug.Log(huge.transform.localScale);
1112          huge.AddComponent<BoxCollider>();
1113          huge.AddComponent<Rigidbody>();
1114          //with right rotation
1115          huge.transform.localEulerAngles=new Vector3(0,rotationY,0);
1116          huge.transform.position=position;
1117  //          huge.rigidbody.mass=nameList.Count+pairwiseTAG.GetLength(0)-populatingState;
1118          huge.rigidbody.drag=10*(nameList.Count+pairwiseTAG.GetLength(0)-populatingState);
1119          huge.rigidbody.angularDrag=0f;
1120          huge.rigidbody.constraints =RigidbodyConstraints.FreezePositionY;
1121          huge.rigidbody.freezeRotation=true;
1122          huge.rigidbody.interpolation=RigidbodyInterpolation.Extrapolate;
1123          huge.rigidbody.collisionDetectionMode=CollisionDetectionMode.ContinuousDynamic;
1124          //it can't be rotate by physics engine
1125          huge.rigidbody.constraints &=~RigidbodyConstraints.FreezeRotationY;
1126          huge.rigidbody.rotation=Quaternion.identity;
1127          string boxTag=GameObject.Find(name).tag;
1128          huge.tag=boxTag;
1129          huge.name=name;
1130          boxesT1.Add(huge);
1131          populatingState++;
1132          Debug.Log(name);
1133
1134          return;
1135
1136          //but if it's bed.. it's OK, accessible too
1137        }else{
1138          commonlyImport(name);
1139        }
1140      }//if populating state<2
1141      else{
1142        commonlyImport(name);
1143      }//populating state>=2 imported anyway
1144    }//importInitialFurniture()
1145
1146    void commonlyImport(string name){
1147      GameObject box;
1148      box=GameObject.CreatePrimitive(PrimitiveType.Cube);
1149      box.transform.localScale=globalBestT1[populatingState,2]*2;//extents
1150  //    Debug.LogError("populatingState"+populatingState);
1151  //    Debug.Log(box.transform.localScale);
1152
1153      box.AddComponent<BoxCollider>();
1154      box.AddComponent<Rigidbody>();
1155      box.transform.position=new Vector3(0,box.collider.bounds.extents.y,0)+
1156        getRandomPosition(populatingState);
1157      box.rigidbody.mass=nameList.Count+pairwiseTAG.GetLength(0)-populatingState;
1158      box.rigidbody.drag=10*(nameList.Count+pairwiseTAG.GetLength(0)-populatingState);;
1159      box.rigidbody.angularDrag=0f;
1160      box.rigidbody.constraints =RigidbodyConstraints.FreezePositionY;
1161      box.rigidbody.freezeRotation=true;
```

```
1162      box.rigidbody.interpolation=RigidbodyInterpolation.Extrapolate;
1163      box.rigidbody.collisionDetectionMode=CollisionDetectionMode.ContinuousDynamic;
1164      box.rigidbody.constraints &=~RigidbodyConstraints.FreezeRotationY;
1165      //      box.rigidbody.constraints &=RigidbodyConstraints.FreezeRotationX;
1166      //      box.rigidbody.constraints &=RigidbodyConstraints.FreezeRotationZ;
1167      box.rigidbody.rotation=Quaternion.identity;
1168      string boxTag=GameObject.Find(name).tag;
1169      box.tag=boxTag;
1170      box.name=name;
1171      boxesT1.Add(box);
1172      populatingState++;
1173      Debug.Log(name);
1174    }
1175
1176    Vector3 getRandomPosition(int id){
1177      Vector3 randomPosition=new Vector3(0,0,0);
1178      Vector3 extentsDifference= Room.roomExtents- globalBestT1[id,2];//extents
1179  //      float radius=Mathf.Sqrt(extentsDifference.x*extentsDifference.x
1180  //                              + extentsDifference.z *extentsDifference.z);
1181
1182      bool isFound=false;
1183      do{
1184        //make sure the randomposition is in room
1185        randomPosition.x=Room.roomCenter.x-extentsDifference.x+ Random.value*2*extentsDifference.x;
1186        randomPosition.z=Room.roomCenter.z-extentsDifference.z+ Random.value*2*extentsDifference.z;
1187        randomPosition.y=Room.roomCenter.y;
1188
1189        if(populatingState!=0){
1190          for(int j=0;j<populatingState;j++){
1191            Vector3 PiEx=globalBestT1[id,2];
1192            Vector3 c0=randomPosition-PiEx;
1193            if(boxesT1[j].collider.bounds.Contains(c0)) break;
1194            Vector3 c2=randomPosition+PiEx;
1195            if(boxesT1[j].collider.bounds.Contains(c2)) break;
1196            Vector3 c1=c0;
1197            c1.z=c2.z;
1198            if(boxesT1[j].collider.bounds.Contains(c1)) break;
1199            Vector3 c3=c2;
1200            c3.x=c0.x;
1201            if(boxesT1[j].collider.bounds.Contains(c3)) break;
1202
1203            if(j==populatingState-1) isFound=true;
1204          }
1205
1206        }else{
1207          //populatingstate==0
1208          isFound=true;
1209        }
1210
1211      }while(!isFound);
1212
1213      return randomPosition;
1214    }//getPosition()
1215
1216
1217
1218    Vector3[] GetVerticesInChildren(GameObject go) {
1219      MeshFilter[] mfs = go.GetComponentsInChildren<MeshFilter>();
1220      List<Vector3> vList = new List<Vector3>();
1221      foreach (MeshFilter mf in mfs) {
1222        vList.AddRange (mf.mesh.vertices);
1223      }
1224      return vList.ToArray ();
1225    }
1226
1227    Vector3 GetFurnitureExtents(Vector3[] vertices){
1228      float xmax=-99999;
1229      float xmin=99999;
1230      float ymax=-99999;
1231      float ymin=99999;
```

```
1232        float zmax=-99999;
1233        float zmin=99999;
1234        for(int i=0;i<vertices.Length;i++){
1235          xmax = Mathf.Max (xmax, vertices[i].x);
1236          xmin = Mathf.Min (xmin, vertices[i].x);
1237          ymax = Mathf.Max (ymax, vertices[i].y);
1238          ymin = Mathf.Min (ymin, vertices[i].y);
1239          zmax = Mathf.Max (zmax, vertices[i].z);
1240          zmin = Mathf.Min (zmin, vertices[i].z);
1241        }//for
1242        Vector3 extents=new Vector3(xmax-xmin,ymax-ymin,zmax-zmin);
1243        extents=extents*0.5f;
1244        return extents;
1245      }
1246
1247      void getKnownFloorplanFurniture(){
1248        /**
1249         * Room preprocess:
1250         * 1. Add DoorBlock;
1251         * 2. Doors Fireplaces [][]( Doors and fireplaces should not be hidden);
1252         * 3. Windows[][](Window shouldn't be hidden by "high_object"
1253         *  whose heightest point is higher than the window center);
1254         */
1255        List<Vector3[]> list1=new List<Vector3[]>();
1256        List<Vector3[]> list2=new List<Vector3[]>();
1257        List<Vector3[]> list3=new List<Vector3[]>();
1258
1259        //floorplanFurniture [][]:( namecode,wallID,0.0)(center)(extents) Vector3(width,depth,height)
1260        for(int i=0;i<InRoomRetrieval.Instance.floorplanFurniture.GetLength(0);i++){
1261          switch((int)InRoomRetrieval.Instance.floorplanFurniture[i ][0]. x){
1262          case 1://is door
1263            Instantiate(GameObject.Find("DoorBlock"),
1264                        InRoomRetrieval.Instance.floorplanFurniture[i ][1], Quaternion.identity);
1265            list1 .Add(InRoomRetrieval.Instance.floorplanFurniture[i]);
1266            break;
1267          case 2://is window
1268            list2 .Add(InRoomRetrieval.Instance.floorplanFurniture[i]);
1269            break;
1270          case 3://is fireplace
1271            GameObject fireplaceBlock= GameObject.CreatePrimitive(PrimitiveType.Cube);
1272            fireplaceBlock .renderer.enabled=false;
1273            fireplaceBlock .transform.localScale=InRoomRetrieval.Instance.floorplanFurniture[i][2]*2f;
1274            fireplaceBlock .AddComponent<BoxCollider>();
1275            int wallID=(int) InRoomRetrieval.Instance.floorplanFurniture[i ][0]. y;
1276            float depth=InRoomRetrieval.Instance.floorplanFurniture[i][3].y*0.5f;// fireplace depth
1277            //position is the fireplace center move towards wall normal with fireplace depth
1278            fireplaceBlock .transform.position=InRoomRetrieval.Instance.floorplanFurniture[i][1]+
1279              Room.walls[wallID,2]*depth;
1280
1281            fireplaceBlock .AddComponent("CollisionHandler");
1282
1283            list3 .Add(InRoomRetrieval.Instance.floorplanFurniture[i]);
1284            break;
1285          default://=0 uninitialised; it should be floorstairs
1286            break;
1287          }//switch
1288        }//for floorplanFurniture
1289        Doors=list1.ToArray();
1290        Windows=list2.ToArray();
1291        Fireplaces=list3 .ToArray();
1292        list1 .Clear();
1293        list2 .Clear();
1294        list3 .Clear();
1295      }//getKnownFloorplanFurniture()
1296
1297      //=============================================================================

1298      int [] getRandomList(int k){
1299        List<int> list=new List<int>();
1300        for(int i=0;i<k;i++){
```

```
        list .Add(i);
      }
      int [] RandomList= new int[k];
      for (int j=0;j<k;j++){
//        Debug.Log("list.Count="+list.Count);
        int idx=(int) (Random.value * list.Count);
//        Debug.Log("Remove:␣list["+idx+"]="+list[idx]);
        RandomList[j]=list[idx];
//        Debug.Log("RandomList["+j+"]="+idx);
        list .RemoveAt(idx);
      }
      return RandomList;
    }//getRandomList()

    //data output for testing numerical result in Matlab
    void writeline(int x, double y){
      file .WriteLine(x.ToString() + "␣" + y.ToString());
    }


}//Class
```

B.5 IO System & Viewing Version on Android

```
using System.Collections;
using System.IO;

public class WriteToTXT : MonoBehaviour {
  bool isFinished=false;
  string path;
  string filename;
  // Use this for initialization
  void Start () {
    path="Assets/Autofurnishing/scripts/";
  }
  // Update is called once per frame
  void Update () {
    if(populatingT1.isfinished && !isFinished){
      filename=populatingT1.floorName+".txt";
      // Write the string to a file.
      System.IO.StreamWriter file = new System.IO.StreamWriter(path+filename);
      for(int i=0;i<populatingT1.boxesT1.Count;i++){
        file .WriteLine("name␣"+ populatingT1.boxesT1[i].name);
        file .WriteLine(populatingT1.globalBestT1[i,0].x.ToString() +"␣"
                        +populatingT1.globalBestT1[i,0].y.ToString() +"␣"
                        +populatingT1.globalBestT1[i,0].z.ToString() +"␣");
        file .WriteLine(populatingT1.globalBestT1[i,1].x.ToString() +"␣"
                        +populatingT1.globalBestT1[i,1].y.ToString() +"␣"
                        +populatingT1.globalBestT1[i,1].z.ToString() +"␣");
        file .WriteLine(populatingT1.globalBestT1[i,2].x.ToString() +"␣"
                        +populatingT1.globalBestT1[i,2].y.ToString() +"␣"
                        +populatingT1.globalBestT1[i,2].z.ToString() +"␣");

      }
      file .Close();
      isFinished=true;
    }
  }//Update()
}
```

```
using System.Collections;
using System.IO;
using System.Collections.Generic;//for List<T>, Queue<T>
public class ReadFurnishedRoom : MonoBehaviour {
  protected FileInfo theSourceFile = null;
```

```
1364  protected StreamReader reader = null;
1365  protected string text = "␣"; // assigned to allow first line to be read below
1366  string path="Assets/Autofurnishing/scripts/";
1367  string [] fileName;//room name
1368  // Use this for initialization
1369  void Start () {
1370    fileName=new string[9];
1371    fileName[0]="room_4_646";
1372    fileName[1]="room_3_644";
1373    fileName[2]="room_2_642";
1374    fileName[3]="room_1_640";
1375    fileName[4]="room_8_653";
1376    fileName[5]="room_11_659";
1377    fileName[6]="room_12_661";
1378    fileName[7]="room_10_657";
1379    fileName[8]="room_5_648";
1380  }
1381
1382  // Update is called once per frame
1383  void Update () {
1384    if(Input.GetKeyDown(KeyCode.S)){
1385      for(int i=0;i<9;i++){
1386        readfile(fileName[i]);
1387      }
1388    }//if press s
1389  }
1390
1391  void readfile(string roomname){
1392    List<string> name=new List<string>();
1393    List<Vector3> position=new List<Vector3>();
1394    List<Vector3> rotation=new List<Vector3>();
1395    List<Vector3> extents=new List<Vector3>();
1396    theSourceFile = new FileInfo (path+roomname+".txt");
1397    reader = theSourceFile.OpenText();
1398    //read centers coordinates
1399    text=reader.ReadLine();
1400    do{
1401      if(text.StartsWith("n")){
1402        string [] word=text.Split(' ');
1403        //word[0]="name"
1404        //word[1,2,3]=<name>
1405        name.Add(word[1]);
1406        //-------------------------
1407        text=reader.ReadLine();
1408        string [] pos=text.Split(' ');
1409        position.Add(new Vector3(float.Parse(pos[0]),
1410                                 float.Parse(pos[1]),
1411                                 float.Parse(pos[2])));
1412        //-------------------------
1413        text=reader.ReadLine();
1414        string [] rot=text.Split(' ');
1415
1416        rotation.Add(new Vector3(float.Parse(rot[0]),
1417                                 float.Parse(rot[1]),
1418                                 float.Parse(rot[2])));
1419        //-------------------------
1420        text=reader.ReadLine();
1421        string [] ext=text.Split(' ');
1422        extents.Add(new Vector3(float.Parse(ext[0]),
1423                                float.Parse(ext[1]),
1424                                float.Parse(ext[2])));
1425      }//if text startwith
1426      text=reader.ReadLine();
1427    }while(text != null);
1428    reader.Close();
1429    GameObject[] furniture=new GameObject[name.Count];
1430    for(int i=0;i<name.Count;i++){
1431      furniture[i]=(GameObject)Instantiate(GameObject.Find(name[i]),
1432                                           position[i],
1433                                           Quaternion.identity);
```

```
1434            furniture[i].transform.eulerAngles=rotation[i];
1435            furniture[i].name=name[i];
1436        }
1437    }//readfile()
1438 }
```

Appendix C

Testing Data

C.1 Brighton House Floor Plan Data Files (Output by Edited Sweet Home 3D)

C.1.1 `rooms.txt`

Listing C.1 is the output txt file (`rooms.txt`) content.

Listing C.1: Contents in output file `rooms.txt`

```
RoomID 1 33428.965
center 86.01169 359.40997 126.42672
8.337913 359.40997 18.576708
8.934485 359.40997 234.08142
163.68547 359.40997 234.27673
163.68547 359.40997 18.576708
RoomID 2 18477.105
center 251.33197 359.40997 176.54895
171.30547 359.40997 118.82117
171.30547 359.40997 234.27673
331.32547 359.40997 234.27673
331.3585 359.40997 118.82117
RoomID 3 108059.875
center 523.8071 359.40997 305.3368
338.94547 359.40997 159.38223
338.6389 359.40997 451.29138
708.9754 359.40997 451.29135
708.9754 359.40997 159.38222
RoomID 4 123520.91
center 859.2529 359.40997 235.12422
716.70795 359.40997 18.489197
716.70795 359.40997 451.75925
1001.7978 359.40997 451.75925
1001.79785 359.40997 18.489212
RoomID 5 69560.07
center 170.02882 -152.4 126.203316
8.927675 -152.4 18.181425
8.927675 -152.4 234.08142
331.12994 -152.4 234.2252
331.12997 -152.4 18.346317
RoomID 6 67481.72
center 169.9844 -152.4 346.58737
8.838854 -152.4 241.89671
8.838854 -152.4 451.278
331.12994 -152.4 451.278
331.12994 -152.4 241.89673
RoomID 7 90698.06
center 586.49023 103.505005 84.87574
1001.67505 103.505005 18.3463
171.30545 103.505005 18.346264
171.30545 103.505005 111.20117
338.74994 103.505005 111.20117
338.77396 103.505005 151.4052
710.8599 103.505005 151.40521
710.8599 103.505005 106.5052
1001.67505 103.505005 106.5052
RoomID 8 206968.5
center 670.385 103.505005 283.2602
338.74994 103.505005 159.02519
338.74994 103.505005 452.3952
1001.68524 103.505005 451.44678
1002.02014 103.505005 114.1252
```

```
718.4799 103.505005 114.1252
718.4799 103.505005 158.8927
RoomID 9 64864.38
center 440.14044 359.40997 85.06949
171.30547 359.40997 18.575167
171.30547 359.40997 111.20119
338.9455 359.40997 111.20119
338.9455 359.40997 151.76225
708.9754 359.40997 151.76224
708.9754 359.40997 18.376719
RoomID 10 287419.66
center 670.2176 -152.4 235.37076
338.74997 -152.4 18.37671
338.74994 -152.4 452.3952
1001.68524 -152.4 451.44678
1001.67505 -152.4 18.346298
RoomID 11 18477.105
center 251.33197 103.505005 176.54895
171.30547 103.505005 118.82117
171.30547 103.505005 234.27673
331.32547 103.505005 234.27673
331.3585 103.505005 118.82117
RoomID 12 33428.965
center 86.01169 103.505005 126.42672
8.337913 103.505005 18.576708
8.934485 103.505005 234.08142
163.68547 103.505005 234.27673
163.68547 103.505005 18.576708
```

C.1.2 `floorplanFurniture.txt`

Listing C.2 is the output txt file (`floorplanFurniture.txt`) content.

Listing C.2: The contents in output file `floorplanFurniture.txt`

```
Furniture: Double window
333.4278 237.48999 342.70184
101.6 10.451842 173.98999
103.505005 ----------
Furniture: Fixed window
91.77942 257.4925 238.0352
91.44 7.6202497 133.985
103.505005 ----------
Furniture: Staircase
526.02484 271.14502 55.191383
73.659996 283.21 335.28
359.40997 ----------
Furniture: Fixed window
4.8144298 513.39746 125.01912
91.44 7.6202497 133.985
359.40997 ----------
Furniture: Fixed window
228.8436 513.39746 238.08673
91.44 7.6202497 133.985
359.40997 ----------
Furniture: Fixed window
334.60544 513.39746 340.73926
91.44 7.6202497 133.985
359.40997 ----------
Furniture: Fixed window
1005.49524 513.39746 136.87473
91.44 7.6202497 133.985
359.40997 ----------
Furniture: Fixed window
1005.49524 513.39746 319.74966
91.44 7.6202497 133.985
-152.4 ----------
Furniture: Fixed window
5.1176753 1.5875015 125.39935
91.44 7.6202497 133.985
-152.4 ----------
Furniture: Fixed window
227.11702 1.5875015 238.08673
91.44 7.6202497 133.985
-152.4 ----------
Furniture: Fixed window
1005.49524 1.5875015 136.87473
91.44 7.6202497 133.985
-152.4 ----------
Furniture: Fixed window
1005.49524 1.5875015 320.42465
91.44 7.6202497 133.985
-152.4 ----------
Furniture: Open door
305.24988 -48.259995 346.07004
91.44 69.03237 208.28
-152.4 ----------
Furniture: Open door
305.2499 -48.259995 64.25754
91.44 69.03237 208.28
103.505005 ----------
Furniture: Fixed window
228.55441 257.4925 238.0352
91.44 7.6202497 133.985
103.505005 ----------
Furniture: Picture window
1019.69415 218.44 292.59476
147.31999 37.61534 135.89
103.505005 ----------
Furniture: Exterior door
```

1004.77386 198.47324 62.50007
91.44 9.042649 189.93646
103.505005 −−−−−−−−−−−−−−−−−
Furniture: Open door
773.5043 207.645 140.00526
83.82 69.03237 208.28
103.505005 −−−−−−−−−−−−−−−−−
Furniture: Open door
658.4169 198.96667 184.90527
83.82 69.03237 190.92334
359.40997 −−−−−−−−−−−−−−−−−
Furniture: Open door
664.17706 454.81873 185.26228
83.82 69.03237 190.81749
359.40997 −−−−−−−−−−−−−−−−−
Furniture: Open door
742.47546 454.81873 61.095093
83.82 69.03237 190.81749
359.40997 −−−−−−−−−−−−−−−−−
Furniture: Open door
286.51416 454.81873 85.32111
83.82 69.03237 190.81749
359.40997 −−−−−−−−−−−−−−−−−
Furniture: Open door
137.8054 454.81873 60.967457
83.82 69.03237 190.81749
103.505005 −−−−−−−−−−−−−−−−−
Furniture: Fireplace
512.1275 228.44125 428.59387
123.5075 45.4025 249.8725
−152.4 −−−−−−−−−−−−−−−−−
Furniture: Fireplace
511.9905 −27.463745 428.57678
153.9875 45.4025 249.8725
−152.4 −−−−−−−−−−−−−−−−−
Furniture: Fireplace
836.9219 −27.463745 428.7455
153.9875 45.4025 249.8725
103.505005 −−−−−−−−−−−−−−−−−
Furniture: Fireplace
836.93 228.44125 428.5939
123.5075 45.4025 249.8725
359.40997 −−−−−−−−−−−−−−−−−
Furniture: Fireplace
512.1275 484.34622 428.79398
93.027504 45.4025 249.8725
359.40997 −−−−−−−−−−−−−−−−−
Furniture: Fireplace
836.93 484.34622 428.94553
93.0275 45.4025 249.8725
103.505005 −−−−−−−−−−−−−−−−−
Furniture: Open door
286.51416 198.91376 85.32111
83.82 69.03237 190.81749
103.505005 −−−−−−−−−−−−−−−−−
Furniture: Open door
137.8054 198.91376 60.967457
83.82 69.03237 190.81749
−152.4 −−−−−−−−−−−−−−−−−
Furniture: Staircase
526.02484 15.2400055 55.191383
73.659996 283.21 335.28