

# Room Placement in Public House Floor plans

Alexander John Dominick Hoare

Submitted in accordance with the requirements for the degree of  
BSc Computer Science

2018/19

40 credits

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Source Code	GitLab Repository	Public (DD/MM/YY)
Report	2 Physical Bound Copies	SSO (DD/MM/YY)
Report	PDF Report	Minerva (DD/MM/YY)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) \_\_\_\_\_

## Summary

We present a method for automated generation of floor plans of the modern-day British-style pub to be used in computer graphics applications. Given the footprint (outline of exterior walls), randomly generated interior walls and a front door, our approach finds the optimal location of the rooms using stochastic hill climbing, calculated by a cost function.

The program must adhere to the legal requirements of number of lavatories, having facilities of a suitable size to handle the number of patrons, and having the rooms within the floor plan positioned in a sensible way that would not cause issues (such as having a bathroom next to a place where beverages are served).

The result is then displayed using Pillow[1], a fork from the Python Imaging Library (PIL).[2]

### **Acknowledgements**

I would like to thank Tom Kelly for his invaluable assistance during this project. His kindness, temperance and patience when working with me was beyond what I expected and his extensive knowledge kept me on the right track. I would also like to thank He Wang for his advice during this project and helping expand the quality of the report.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Aim . . . . .	2
1.3	Objectives . . . . .	3
1.4	Timeline . . . . .	3
1.5	Deliverables . . . . .	4
<b>2</b>	<b>Background Research</b>	<b>5</b>
2.1	Public Houses . . . . .	5
2.2	Relevance to Degree . . . . .	6
2.2.1	COMP1911 Professional Computing . . . . .	6
2.2.2	COMP1721 Object Oriented Programming . . . . .	6
2.2.3	COMP3811 Computer Graphics . . . . .	7
2.3	Alternative Solutions . . . . .	7
2.3.1	CityEngine . . . . .	7
2.3.2	Houdini . . . . .	8
2.4	Chosen Solution . . . . .	8
2.4.1	Initial Implementation . . . . .	8
2.4.2	Geometry Libraries . . . . .	9
2.4.3	Footprint Changes . . . . .	9
2.4.4	Rooms . . . . .	9
2.4.5	Removal of Furniture Placement . . . . .	10
2.4.6	Interior Walls . . . . .	10
2.5	Potential Search Algorithms . . . . .	10
2.5.1	Genetic Programming . . . . .	10
2.5.2	Simulated Annealing . . . . .	10
<b>3</b>	<b>Development and Methodology</b>	<b>12</b>
3.1	Environment . . . . .	12
3.1.1	Python 2.7.5 . . . . .	12
3.1.2	Pillow . . . . .	12
3.1.3	SymPy . . . . .	13
3.1.4	Version Control . . . . .	13
3.2	Footprint Creation . . . . .	13
3.2.1	Rectangle Shape . . . . .	14
3.2.2	L-Shape/T-Shape . . . . .	14

3.2.3	Cross-Shape . . . . .	14
3.2.4	Exterior Wall Recognition . . . . .	14
3.3	Interior Wall Creation . . . . .	15
3.4	Door Creation . . . . .	15
3.5	Room Placement . . . . .	16
3.6	Optimization . . . . .	17
3.6.1	Distance between rooms . . . . .	17
3.6.2	Invalid spaces . . . . .	17
3.6.3	Mean distance from patrons . . . . .	18
3.6.4	Area usage and room size . . . . .	18
3.7	Fitness Function . . . . .	19
3.8	Exhaustive Search . . . . .	21
3.9	Stochastic Hill Climbing . . . . .	22
<b>4</b>	<b>Evaluation</b>	<b>23</b>
4.1	Using the Program . . . . .	23
4.2	Potential Bugs . . . . .	23
4.3	Project Evaluation . . . . .	24
4.4	Self Evaluation . . . . .	26
4.5	Potential Improvements . . . . .	28
4.5.1	Interior Wall Changes . . . . .	28
4.5.2	Footprint Changes . . . . .	28
4.5.3	Room Changes . . . . .	29
<b>5</b>	<b>Conclusion</b>	<b>30</b>
5.1	Summary . . . . .	30
5.2	Future Work . . . . .	30
5.2.1	Furniture . . . . .	30
5.2.2	Displaying Results . . . . .	31
	<b>References</b>	<b>32</b>
	<b>Appendices</b>	<b>34</b>
<b>A</b>	<b>External Material</b>	<b>35</b>
<b>B</b>	<b>Ethical Issues Addressed</b>	<b>36</b>
<b>C</b>	<b>Gant Charts</b>	<b>37</b>
<b>D</b>	<b>User Evaluated Images</b>	<b>38</b>

# Chapter 1

## Introduction

### 1.1 Introduction

Public Houses are a critical part of British life, culture and the economy, with over 48,350 in the United Kingdom. Owning and running a pub in recent years has become increasingly difficult, with pub numbers dropping by 17% since 2000. One of the more challenging areas with pubs is creating a floor plan for a pub owner who wishes to open a new venue, whilst still adhering to legal requirements.[3]

Public houses are increasingly being used in Computer Graphics. With many video games set in a fantasy setting or in modern day British urban setting, developers are spending a lot of time hand-crafting pubs/taverns within their game worlds, with many developers opting to use the same tavern template. They are also used by architects who want to design a public house floor plan, ensuring that the facilities within are both legally acceptable and adequately positioned.

### 1.2 Aim

The aim of this project is for the program to produce a layout of a public house that is both sensible in design, as well as adhering to legal requirements based on a given footprint. To demonstrate what is meant by a footprint, see the sketch below:

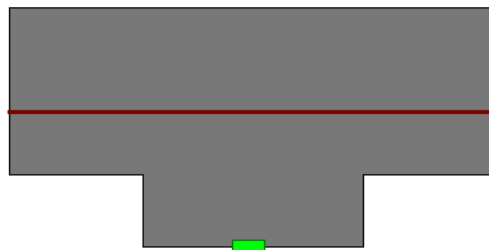


Figure 1.1: Sketch of footprint as input

Figure 1.1 shows a sketch of a footprint that would be used as an input for the program,

where the green bar is the door to enter the building, the red line is one of the interior walls and the gray area is the free space within the public house.

### 1.3 Objectives

The main objective of this project is to produce a floor plan that has the rooms within placed in sensible locations, a layout is deemed as good if it adheres to the following:

- Ensuring the rooms within the footprint are not clustered together, creating an aesthetically pleasing venue for the patron.
- The rooms are not too small given the size of the pub and the number of patrons it can hold.
- The rooms are not too close together such that it creates a bottleneck in which certain areas are too crowded.
- The rooms are positioned so that there are no hygiene concerns, such as having a lavatory next to the bar.

### 1.4 Timeline

The first three weeks will be used to select and decide on a particular project to be worked on. Then, during weeks 4-8, background research will be conducted in order to find the optimal way to analyze, develop, implement, test and evaluate the project. During weeks 8-10, an intermediate report will be written for the Assessor, in which there will be a break afterwards over the Christmas. During weeks 11-16, the implementation of the project will be undertaken. This will likely take be one of the main sections of the project, with the majority of the work being conducted here. During weeks 17-18, the project's code will be tested, evaluated and then tweaked based on the results of the testing. Weeks 19-20 over the Easter break, will be spent on and concluding with the writing up of the final report itself.

WEEK	1	2	3	4	5	6	7	8	9	10	BREAK	11	12	13	14	15	16	17	18	19	BREAK	20
TASK																						
Choosing project																						
Research																						
Writing Intermediate Report																						
Implementation of project																						
Testing																						
Evaluation																						
Writing final report																						

Figure 1.2: Planned timeline of the project



## 1.5 Deliverables

The project will have the following deliverables:

Deliverable	Method of Delivery
Source Code	GitLab
Final Report	PDF

Figure 1.3: Table containing deliverables and methods of delivery

# Chapter 2

## Background Research

Procedural modelling of urban environments including both residential and commercial buildings is an exciting and relatively new space, with one of the first large explorations of the subject shown in the Procedural Modeling of Cities by Yoav I. H. Parish and Pascal Mueller[4] in 2001. In this paper, it uses L-Systems that are typically used to generate plant eco-systems, to generate cities on a large scale. It employs this L-System method to create buildings in a similar style to how plants grow.

This chapter explores the history of public houses, as well as the initial direction of the project, and how the scope changed over time as the requirements increased.

### 2.1 Public Houses

Public Houses (pubs) are establishments licensed for the sale of alcohol and other drinks, as well as occasionally food. They are primarily found in the United Kingdom and the Republic of Ireland, but pubs can now be found across the globe, with an estimated 7000 Irish pubs worldwide[5].

Pubs are often incorrectly compared to bars, which highlights a critical difference between the two. Bars differ from pubs due to the fact that bars have a literal bar, in which drinks are served by a bartender. However in a pub, it is more loosely defined as a drinking establishment for the public to enjoy. This brings to light the idea of what a pub is; it is a place to eat, drink and be together. Pubs often find themselves as the cornerstones of communities. This means that the design, aesthetic and practicality of a pub is of absolute importance.

From results taken from RightMove[6], a website used to browse properties for sale, we were able to find that the average size of a public house for sale 40 miles from the city centres of London, Manchester and Glasgow to be 3292ft<sup>2</sup>, taken from 19 total pubs found. This will allow for a fair comparison to be made in how many pixels will represent a foot, allowing for comparable sizes between the output of the program, and a realistic pub floorplan.

Pubs are especially difficult to design, with a number of different legal requirements put in place to have a functional and acceptable establishment such as:

- Legal requirements surrounding how large the bathrooms must be to accommodate a number of patrons based on the size of the pub.[7]
- Licensing laws surrounding how many patrons an establishment is allowed to serve. [8]
- Floor plan laws describing measures that need to be taken to ensure patrons can easily leave in the event of a fire. [9]
- Accommodations made to avoid disability discrimination. [10]

It was decided to focus mainly on the legal requirements on bathroom size given a number of patrons. This was done firstly because we will operate under the assumption that the pub, for which the floor plan is being designed for, already has the correct license regarding the number of patrons it can serve. Secondly, it was deemed that fire escape architectural planning was beyond the scope of the project, as it would prove difficult to assess how good a particular layout is for an evacuation in the event of a fire, as it would require crowd-simulation. Finally, accommodations to avoid disability discrimination were also not considered as they are not critical components of the floor plan, but rather of the design and structure of the establishment.

Another aspect that is important to focus on is how to place the individual rooms within the layout, which would require the use of a large amount of optimization to find the best layout. The likely avenue pursued for optimization will probably be a fitness function that takes into account the previous scores of other layouts.

## 2.2 Relevance to Degree

This project requires the use of prior knowledge obtain from previous modules of the degree. In particular, the following modules' content will be applied:

### 2.2.1 COMP1911 Professional Computing

The foundations of ethical issues that may be faced in this project, such as fire escape issues or disability access.

### 2.2.2 COMP1721 Object Oriented Programming

This module allowed for me to make polymorphic, easily testable and scalable code, which will undoubtedly prove useful to add/remove additional rooms or constraints for the project.

### 2.2.3 COMP3811 Computer Graphics

This module will allow me to cleanly display the graphical elements of the floor plan, as well as perform simple graphical checks (e.g to see if a pixel falls above or below a line).

## 2.3 Alternative Solutions

In order to create a floor plan, multiple different technologies were explored. In this section, detail will be given on the possible solutions and why they were/were not selected.

### 2.3.1 CityEngine

In the beginning, it was strongly considered to have both the interior and exterior of the pub designed by CityEngine[11], an advanced 3D Modelling Software designed for the creation of immersive 3D urban environments developed by Esri RD Center Zurich. It was also a strong choice as many of the best solutions to procedural urban modelling approaches, such as a grammar-based approach[12] (the use of a set of production rules for strings in a formal language) was designed by Pascal Mueller, the creator of CityEngine, this encouraged the easy application of methods such as split-grammars[13] for the creation of building facades. Split-grammars are the application of a rule-set defining at which point a building will be 'split' into different sections. This allows for the definition of where features of a facade will be, such as windows or doors.

However, CityEngine proved challenging due to the intended purpose of the software. For example, CityEngine was designed to create large, procedurally generated cities based on grammar-rules, rather than individual structures. Secondly, CityEngine's design focused primarily on building exteriors, likely requiring an external piece of software to place individual rooms, place furniture or perform rendering. CityEngine's employment of a rule-based modelling language called Computer Generated Architecture (CGA) did not allow the level of flexibility, computation or resources that a typical programming language like Python does.

Whilst CityEngine did provide a free trial, it was time-limited to 30 days, causing a lot of difficulty after getting comfortable with the software to maintain a continuous workflow. After experimenting for a month, it was deemed that due to CityEngine neither being designed to the task at hand, and the weak supported languages, the project would provide better results if taken in a different direction.

### 2.3.2 Houdini

After deciding not to use CityEngine, Houdini[14] was explored as a different possible option. Houdini is a 3D application software developed by SideFX. It offers strong attention to procedural generation, which made it a strong option to consider due to the random and procedural nature of a random 3D pub model. However, its complex user interface and broad capabilities made it difficult to create a product that fitted the scope of the project. Houdini as a solution also proved unwise due to the difficulty it had in the performance of optimization.

## 2.4 Chosen Solution

As the scope of the project continued to grow, it was quickly realised that it was impossible to fit in all of the original ideas of:

- Generation of a realistic pub exterior, including realistic exterior shapes and textures.
- Generation of a realistic pub interior space, whilst maintaining the shape of the exterior building
- Automatic placement of rooms and facilities within the establishment.
- Automatic placement of furniture based on room types and other constraints.
- Exporting of the interior and exterior of the pub into a .obj file to be used in 3D modelling software.
- Importing the pub to perform realistic lighting and rendering to create an aesthetically pleasing finished product.

As a result, the scope of the project was forced to focus on the perfection of a smaller part, of which the procedural generation of the floor plan with regards to room placement was selected.

### 2.4.1 Initial Implementation

At first, the large bulk of development was largely spent on the geometry of the shapes in the footprint, such as the building outline, interior walls and the walls of the rooms to be placed. In order to create clear, concise, scalable and polymorphic functions to allow for new footprint shapes, new rooms and interior walls to be added/removed easily

A lot of planning went into ensuring a fairly bug-free codebase, which would prevent small mathematical errors from having a cascading effect, resulting in problems further down the line on results from important functions, such as the cost function.

### 2.4.2 Geometry Libraries

Due to the fact that at this point, an entire geometrical library was being created from the ground up, it was considered a good option to bring in an external geometrical library, in this case SymPy[15] was deemed to be the best choice given that the functionality offered was almost identical to that which had been developed in-house.

However, the adoption of the SymPy library caused a lot of functionality to become either over-cumbersome or obsolete, resulting in a massive rewrite of a lot of pre-existing functionality. The introduction of line-segments by Sympy, which had fixed length, made all line distance and intersection calculations redundant.

### 2.4.3 Footprint Changes

As the scope of the project expanded, a few measures were taken to avoid unnecessary complexity during the later stages on the implementation. One of the key measures taken in order to do this was by changing the guidelines defining the shape of the footprint. This was done by making the footprints comprise of exclusively parallel horizontal and vertical lines, meaning an entire footprint would be only made up of squares and rectangles. This decision was made in order to allow for simpler interior wall placement, room placement as well as area calculations and overall evaluations.

### 2.4.4 Rooms

The background research that went into the rooms was of an increasingly critical nature as time went on. Firstly, with the decision made to keep the footprints comprised of horizontal and vertical lines, both the shape and placement of the rooms will be significantly changed. Firstly, with horizontal and vertical lines only, it made sense to keep the shape of the rooms to a similar style, with rooms either being square or rectangular. The placement of the rooms was also heavily influenced by the footprint shape, as it made the implementation of a 'sliding' affect along the walls easy to implement and evaluate.

The decision was made also on the types of rooms that would be implemented into the public houses' floorplans. In order to keep the floorplans generated as accessible as possible to every single pub, the room types were constrained to only being facilities that one could find in every single pub in the world: a bathroom, a bar and a storage room. Without these three key essential rooms, no pub would be legal nor practical.

### 2.4.5 Removal of Furniture Placement

Initially, the project's implementation would also contain furniture placed within the interior of the pub, creating an interactive furniture layout[16], which would subsequently place different furniture types at different locations and angles based on a series of separate Anthropometric constraints[17], meaning that certain distances and constraints are defined to allow for furniture to be placed in a sensible and suitable fashion to be used by humans, such as distance between a chair and a table.

### 2.4.6 Interior Walls

The interior walls were kept quite simple, as it was suspected that the implementation would prove difficult. The footprint shapes would allow for an easy application, and with all the lines of the interior walls, exterior walls and walls of the rooms being either parallel or perpendicular, finding intersections during run-time would ideally prove trivial.

## 2.5 Potential Search Algorithms

A variety of different search algorithms were explored to deal with the problem of the vast search space of different permutations of room placement within the foot print. Although Stochastic Hill Climbing was eventually selected, the following algorithms were also visited:

### 2.5.1 Genetic Programming

Genetic Programming is a particular technique in which programs are given a 'set' of 'genes', which are subsequently evolved using a genetic algorithm. The evolution's are in fact computer programs themselves, which are then able to perform well in a predefined task, with a function used to determine the programs 'fitness'. This would have been an excellent and exciting method to use for the search space, however due to the fact that the implementation of genetic programming would be quite cumbersome, it is also an area of active research meaning documentation available would likely be fairly limited.

### 2.5.2 Simulated Annealing

Simulated Annealing was also explored as a possible solution to the search space, as it would prove more effective than Stochastic Hill Climbing as it is more likely to return a global optimum for the fitness function. However it was determined that despite this, stochastic hill climbing would be better as although it is not quite as complex and does

not find a global maximum, it is simple and easy to implement and both display similar run-times.



# Chapter 3

## Development and Methodology

This chapter provides detailed information regarding the development of the program. It focuses on the environment, the libraries selected and why. It also focuses on the creation of the layouts, including the creation of the base footprints themselves, the creation of the interior walls and the placement of rooms and the door. The method used to calculate the effectiveness of a particular layout is also discussed, as well as reference to the different search methods used to explore the problem space.

### 3.1 Environment

The program was written in Python 2.7.5 using the Atom text editor. To run the program, a virtual environment was used with the Pillow[1] graphical library, with a vast amount of processing done by the SymPy[15] library, which aided with the geometry of the problem.

#### 3.1.1 Python 2.7.5

Although Python 2.7.5 was released in May 2013, and Python 3 has been the choice of many Python developers, Python 2 has the advantages of having a much older, more stable and larger base of external libraries to use. Whilst both Pillow and SymPy can operate in python 3.5/3.6, it was decided that in order to have the ability to easily incorporate additional libraries in future, that Python 2.7.5 would be the best option.

#### 3.1.2 Pillow

Pillow adds image processing capabilities to the Python interpreter, and is a fork based on the Python Imaging Library (PIL). PIL's last update was on the 15th November 2009, and was picked up by Alex Clark and Contributors in order to deal with the large number and frequency of issues reported, as well as PIL not being setuptools compatible. As a result, Pillow was chosen in order to display geometrical shapes once their positions and co-ordinates had been determined. Pillow was the imaging library of choice due to its extensive file format support, powerful image representation capabilities and simplicity of use.

### 3.1.3 SymPy

SymPy is a Python library for symbolic mathematics. SymPy was used throughout the development of this program to detect collisions between both the interior/exterior walls and the rooms, measure distances between rooms and to make geometric comparisons between shapes effortlessly. SymPy was chosen for its simplicity, comprehensibility and extendability. SymPy is also seen as being quite lightweight, due its reliance on only the mpmath[18] library, used for arbitrary floating point arithmetic. This lightweight functionality proved critical later in the development cycle as the search space of the problem expanded exponentially.

### 3.1.4 Version Control

To keep track of progress and to maintain a backup, the source code was frequently uploaded to GitLab[19] using Git.

## 3.2 Footprint Creation

A footprint to be used by the program is comprised purely of horizontal and vertical lines, and can take the shapes of either a rectangle, a T-shape, a cross shape or an L-shape.

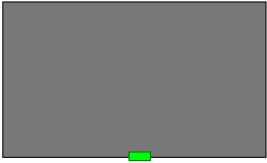

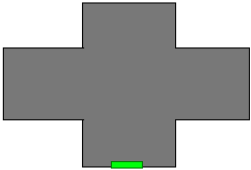
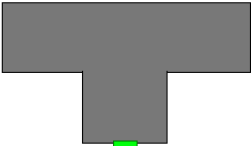
Shape Type	Shape Sketch
Rectangle	
L-Shape	
Cross Shape	
T-Shape	

Figure 3.1: Table containing footprint shape types and sketches

A pseudo-random number generator built-in to Python is used to select a number between (0-3) inclusive, which is in turn used to select which footprint layout will be used. The co-ordinate system used by PIL places the origin at the top-left of the image. The construction method used for each shape is outlined below:

### 3.2.1 Rectangle Shape

Beginning at a starting co-ordinate (typically 100,100) to allow for visual clarity regarding the overall size of the layout, a function is also passed an ending co-ordinate. These co-ordinates are passed into one single tuple of tuples, which contains all of the vertices of the footprint.

### 3.2.2 L-Shape/T-Shape

The function is passed a starting coordinate (typically 100,100), and follows a set path describing the configuration of the L/T. The size of the individual edges between the corners/vertices of the footprint are determined again by a pseudo-random number generator, picking values between (150, 850) on the x-axis and (150, 430) on the y-axis for the L-Shape, and (150, 430) on the x-axis and (150,566) on the y-axis for the T-shape. This ensures that the footprint does not become larger than the window itself. The difference of range between the potential values on the x and y axis are due to the rectangular shape of computer screens, making the image displayed appear more aesthetic.

### 3.2.3 Cross-Shape

For aesthetic purposes, the cross-shape footprint works for symmetry, deciding at the beginning of the function how large the sections will be, in regards to a number of pixels it will move on the x/y axis. Unlike the other sections, the cross does not begin at the typical starting point (100, 100), but rather begins at  $(100, 100) + (\text{section\_size\_x}, \text{section\_size\_y})$ , before following the contours and configuration of a simple cross-shape. The section size increases on the x-axis are between (150, 400) and (150, 250) for the y-axis.

### 3.2.4 Exterior Wall Recognition

The vertices are passed into a function that orders the vertices beginning at the top-leftmost moving anti-clockwise. This sorting allows for the rooms to be placed along the exterior walls, and recognising which vertices are connected by an edge (the exterior wall).

### 3.3 Interior Wall Creation

A variable number of interior walls are created within the footprint, in an either horizontal or vertical direction. Two random connected vertices are selected, and a random value from 0.0 to 0.9 is selected to indicate the position of how far along the exterior wall the interior wall will appear. Due to the anti-clockwise sorting of the vertices of the exterior walls, the direction of the interior wall is determined by a 90° anti-clockwise rotation, which will always point the interior walls towards the direction of the centre of the footprint.

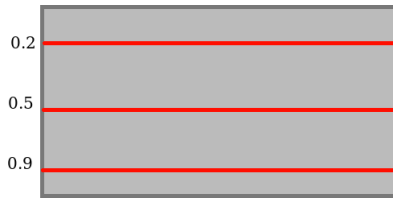


Figure 3.2: Example sketch of how objects are positioned along the exterior wall

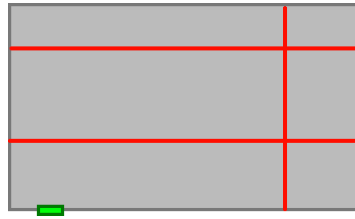


Figure 3.3: Example sketch of three walls placed in footprint.

The interior wall is then checked for intersections across every exterior wall, whilst ensuring that the intersection occurs on the exterior wall segment and that it does not lie outside of the footprint. Once an intersection has been found, it is drawn and the points comprising the interior wall are returned.

### 3.4 Door Creation

The door is placed at a random point between two connected vertices of the exterior walls. The door is drawn from a centre-point and expanded outwards when drawn to give a more realistic depiction of what a door looks like. The centre-point of the door is returned to be used in the cost function. The door can be placed at any point along any exterior wall.

### 3.5 Room Placement

A room is placed along two connected vertices of the exterior wall, with its position on the edge denoted by a number between 0.0 and 0.9 inclusively and a size given in pixels.

There are three different types of rooms; the bathroom, the kitchen and the storeroom. Once the initial vertex of the room has been placed, the new points of the room are drawn in an anti-clockwise pattern based on the size of the room given, in pixels. The storeroom and bathroom are squares, whilst the bar is shaped as a rectangle, in order to more realistically reflect the shape of a bar.

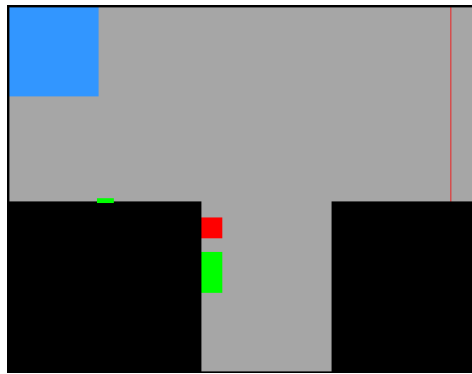


Figure 3.4: Simple layout of rooms in a floor plan.

In the figure above, light-blue is used for the bathroom, green for the bar and red for the storeroom.

Each of the rooms must be checked to ensure there is no intersection with either the exterior walls, the interior walls or the walls of another room within the footprint. If no intersection is found, the room is placed and recognized as a valid placement.

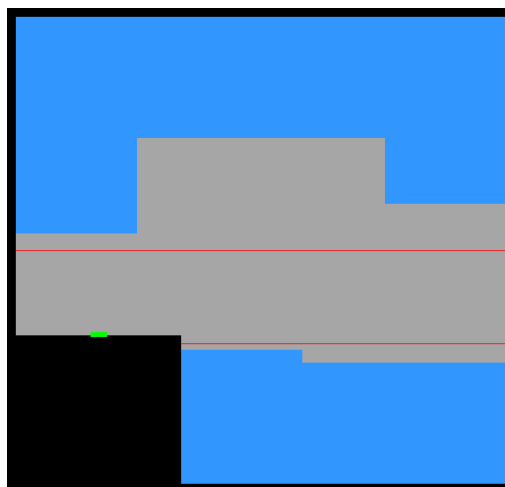


Figure 3.5: Sketch of simple layout with two interior walls, showing available valid placements

The above sketch shows the available placements that a bathroom (blue) may have in a particular layout. It is shown that it cannot be placed on the lower-left wall, as it intersects with both the door and the interior walls. The centre of the right exterior wall also shows no valid placement for a bathroom, as the two interior walls would cause an intersection. It should also be noted that the gaps between the interior walls and the valid bathroom positions are caused by the increments that valid placements are checked (of 0.1/iteration, where 0.0 is the minimum and 1.0 is the maximum).

## 3.6 Optimization

### 3.6.1 Distance between rooms

One of the key constraints that determines the layout of the floor plan and the positions of the rooms is the distance between the different rooms from other rooms or objects in the space. The first example of which can be seen with the importance of the distance between the bathroom and the bar itself. This is a very important constraint due to the critical nature of hygiene for any establishment that serves food or drink. The constraint describes that the distance between the bathroom and bar must be kept as large as possible. This is very similar to the constraint to increase the distance between the bathroom and the door, as the potentially smell and unsightly nature of a bathroom may discourage customers from entering the pub.

Another of the constraints is the distance between the door and the bar. This distance must be kept as short as possible, as patrons entering the establishment want to be able to see the bar, to know where it is, as well as be able to quickly access the bar, rather than having to walk through the entire establishment to get served. This constraint's main attributes are also shared with the need to keep distance between the storeroom and the bar as small as possible. This is done in order to reduce time it would take to fetch additional supplies or the refill a barrel if needed.

This links heavily to the fitness function, as the distance between the rooms is the most important, resulting in a large increase of the weights affecting the room values.

### 3.6.2 Invalid spaces

When a room is placed, a check is made to check if there are any spaces between the walls of the room and the interior/exterior walls. If a space exists, the program checks as to whether or not it is below a certain threshold. This is done because it allows for areas that are simply too small to hold any patrons in to be taken into account when searching the problem space.

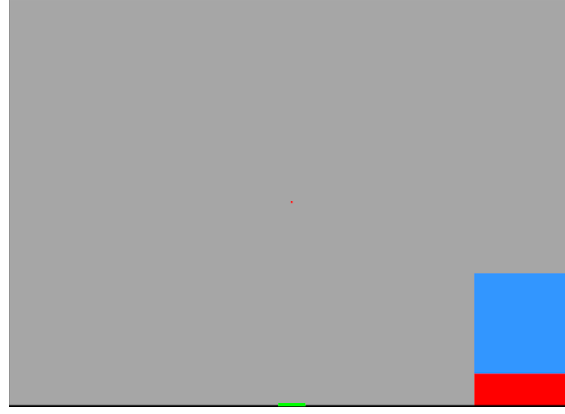


Figure 3.6: Sketch of simple layout with error spaces shown in red.

In this sketch, a bathroom (blue) is drawn in a simple rectangular layout. As a result, a red rectangle has been drawn in the space that is below the minimum, highlighting it as too small for use by patrons.

### 3.6.3 Mean distance from patrons

The mean distance from the drinking space to the rooms is also very important, as it prevents certain rooms from being hidden around corners - ensuring easy accessibility to the patrons at all times. This is calculated by dropping several points in a grid-like pattern, every 100 pixels on the x and y-axis. The point is detected as being either inside or outside of the footprint. A calculation is made to determine the total distance between all of the rooms and all of the points, and then an average is found by dividing by the number of points.

$$m = \frac{\Sigma d(p, \text{bathroom}) + \Sigma d(p, \text{bar}) + \Sigma d(p, \text{storeroom})}{n}$$

Where m is the mean distance from all the rooms to the drinking space, d() is the function calculating the distance between the point and a room, and n is the total number of points used for the calculation. This equation calculates the sum of all points and their distances to all of the rooms and finds an average by dividing by the number of points.

### 3.6.4 Area usage and room size

The sizes of the rooms are also variable based on the overall size of the footprint. The bathroom size is required to adhere to the following legal requirements: [7]

Facility	Male	Female
WC	1 for up to 150 males plus 1 for every additional 150 males or part thereof	1 for up to 12 females plus 1 for 13 to 30 females plus 1 for every additional 25 fe- males or par thereof
Urinal	2 for up to 75 males plus 1 for every additional 75 males or par thereof	N/A
Wash Basin	1 per WC and in addition 1 per 5 urinals or part thereof	1 per 2 WCs

Figure 3.7: Table containing washroom facility types and how many are required by law given the number of male and female patrons

The sizes of the facilities when determining the size of the bathroom is determined as follows[20]:

Facility	Size
WC	1.275m <sup>2</sup>
Urinal	0.1105m <sup>2</sup>
Wash Basin	0.0957m <sup>2</sup>

Figure 3.8: Table containing sizes in metres squared of each washroom facility

With the assumption of 75% male to 25% female, we can calculate the number of patrons by assuming 4 patrons per  $3m^2$ . Converting from metres to pixels in the ratio  $0.00021777003m^2 : 1pixel^2$ , we can translate the total area required for all the legal facilities within the bathroom, and assign that value to the size of the bathroom.

The ratio of pixels to metres was assigned by finding the square footage of a large public house (  $4000ft^2$ ) and comparing it to the maximum square-pixels of the largest rectangular layout possible by the program (1280000). The ratio of square feet to metres was found, and then compared to the ratio of square feet to square pixels.

### 3.7 Fitness Function

Once a valid placement has been found for all three of the different rooms within the footprint, an objective fitness function is used to determine how effective the layout is, based on the series of constraints listed above.

$$c = (w_1\Delta_{dt} + w_2\Delta_{bt} - w_3\Delta_{bd} - w_4\Delta_{sb}) + w_5(\alpha_f - \alpha_t - \alpha_b - \alpha_s) - (w_6\lambda_t + w_7\lambda_b + w_8\lambda_s) - w_9\mu$$

The values of each of the symbols in the cost function is listed below:



Symbol	Definition
$\Delta_{dt}$	The distance between the door and the bathroom (toilet)
$\Delta_{bt}$	The distance between the bar and the bathroom (toilet)
$\Delta_{bd}$	The distance between the bar and the door
$\Delta_{sb}$	The distance between the storeroom and the bar
$\alpha_f$	Area of the entire footprint
$\alpha_t$	Area of the bathroom (toilet)
$\alpha_b$	Area of the bar
$\alpha_s$	Area of the storeroom
$\lambda_t$	Invalid spaces left by placement of bathroom (toilet)
$\lambda_b$	Invalid spaces left by placement of bar
$\lambda_s$	Invalid spaces left by placement of storeroom
$\mu$	Mean distance of all rooms from drinking space

Figure 3.9: Table containing symbols used in the cost function and their definitions

Weight	Value
$w_1$	0.9
$w_2$	0.9
$w_3$	1.0
$w_4$	0.9
$w_5$	0.0003
$w_6$	0.05
$w_7$	0.05
$w_8$	0.05
$w_9$	0.1

Figure 3.10: Table containing weights used in the cost function and their values

In the tables above,  $w$  is used to symbolize a different weight given to each of the parameters in the cost function. Weights  $w_1 - w_4$  are used for the distances between objects (rooms, interior/exterior walls, doors). All of the weights except for the weight regarding the distance between the bar and the door are 0.9, this was done in order to prioritize the importance of having a bar close to the entrance of a bar,  $w_5$  symbolizes the weight used on the leftover area subtracted from the total footprint area. A low value (0.0003) relative to the other values was used due to the large size of the leftover area, this low weight brings all of the numbers back down to a more normal size.  $w_6 - w_8$  are the weights used to affect the invalid spaces left by the rooms and  $w_9$  is used for the mean distance of all rooms from the drinking space. It should be noted that all of

the variables in the fitness function except for the weights are fixed values and do not change during run-time.

### 3.8 Exhaustive Search

The exhaustive search checks for every possible permutation of room placements along every exterior wall. Each room begins at position 0.0 along the exterior wall on the first three exterior walls. For example, the bathroom begins on the first exterior wall, the bar on the second and the storeroom on the third.

The exhaustive search will then slide the storeroom across all of the exterior walls, calculating a cost for each position. Once that has been done, the bar will move 0.1 along its currently positioned exterior wall, before running the storeroom across every possible location again. The search will continue this process, moving the bar across all of the exterior walls until it runs out of new permutations for both the bar and the storeroom. At this point, the process will repeat except the bathroom will be moving 0.1 along all of the exterior walls, exploring every possible position for both the bar and storeroom. At every change to any room, a cost is calculated and the highest cost from all of the permutations is selected and displayed.

Python is locked to using a single CPU processor as a result of the Global Interpreter Lock (GIL), the GIL results in a drastic increase in the run-time of the program in a search space that is already vast. To mitigate this, multiprocessing was introduced in order to utilise all of the CPU cores to calculate the costs of each layout. An issue that arose was the difficulties of having shared memory, especially seeing as in order to find the best cost, the currently running process would need to have knowledge of the highest cost, which cannot be done simultaneously across multiple processes. As a result, every single multiprocessed thread writes one after the other into a text file (results.txt), which stores, the cost of the layout, and the co-ordinates of the rooms within the layout. At the end of the program, this text file parses all of the data found in the text file, picking the co-ordinates from the line that has the highest cost stored. This function however has a massive run-time, averaging at over 3 hours per run. This is likely to be because of the vast amount of cost calculations it has to make over a large search space. In the scenario of a cross-section footprint shape, there are over 1,685,040 different possible permutations of room placement at 0.1 increments, and 1,723,682,400 permutations at 0.01 increments. With this many different permutations, each with their own cost having to be calculated, the run-time was unusable.

### 3.9 Stochastic Hill Climbing

As a result of the vast search space of the room placement, an exhaustive search is currently not the most effective search method, in regards to time taken. As a result, Stochastic Hill Climbing was introduced to reduce the search space. The Hill Climbing method takes initial starting points for every room, similar to the exhaustive search, and then calculates 'neighbours'/'nodes' for alternative layouts. For example, each room is given a  $\pm 0.1$  change in its position and a  $\pm 1$  change in the exterior wall it is placed on, with the number rolling over back to the start or end of the list if the value exceeds the number of exterior walls or available positions. This results in 12 different neighbours being created per iteration.

From there, each neighbour's layout has a cost calculated and assigned to it. If there exists a neighbour with a higher cost than the current node, then it is selected as the main node and the process is repeated, creating new neighbours and calculating their scores. If no higher cost is found, then the program terminates and displays the current node's layout. Although this search solution is faster with a run-time of only 91.909 seconds (averaged over 10 runs), it has weaknesses against an Exhaustive Search due to the fact that it is likely only to find a local maximum of the search space, whereas an exhaustive search will always find the global maximum.

# Chapter 4

## Evaluation

### 4.1 Using the Program

To use the program, using one of the machines located in Dec10, simply create a virtual environment with SymPy and Pillow installed, and run the program `main.py` with the following command in the terminal:

```
python main.py
```

This will then run the program. The program will display its progress by printing two different messages.

Search Method	Print Statement	Definition
Hill Climbing	Creating neighbours	A new node has been selected and neighbours are being created for it.
Hill Climbing	New best found	A new best score has been found when checking through the neighbours.
Exhaustive	Looping	Every possible permutation of the second and third rooms have been tested, now the location of the first will be tried.

Figure 4.1: Table containing search method running, the statement printed to the terminal and what the statement means

These messages are displayed during runtime firstly in order to give some feedback to the user about the operations of the program, and also because due to the large amounts of time the program can take, the regular print statements can act as an assurance that the program is working and has not crashed.

In order to change what kind of search algorithm is used, comment/uncomment the line that describes which function is used on lines 26-28 in `main.py`.

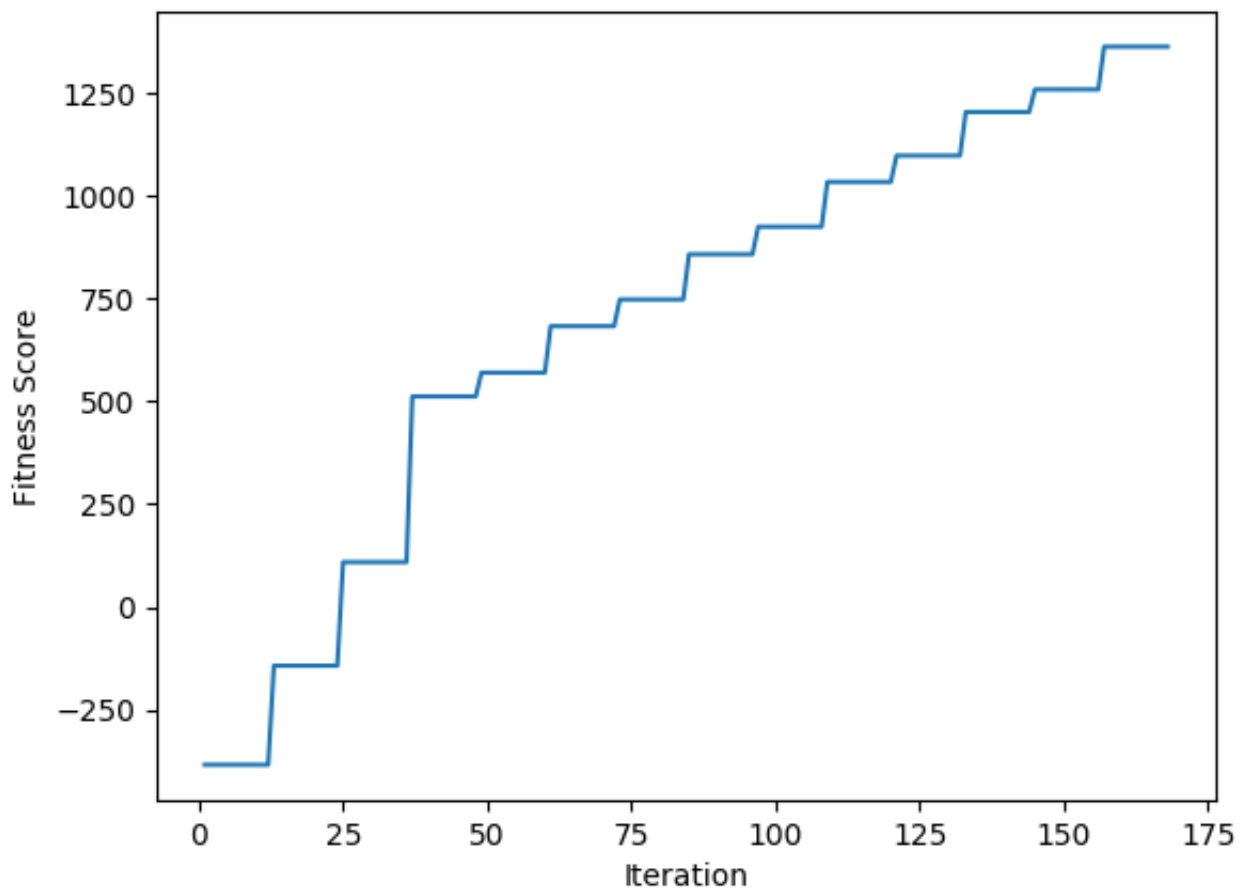
### 4.2 Potential Bugs

There are a few known potential bugs that currently exist in the final deployment of the project. The first of which can cause a crash at the start of the program. This occurs

when the first node used for the Stochastic Hill Climbing solution does not return a valid placement, due to either intersections with interior walls or the exterior walls being too small to place a valid room placement. When this error occurs, the exception will be "TypeError: argument must be sequence". To fix this error, just restart the program as before, and it is unlikely to occur again. The second known bug is door/interior wall placement in the middle of the footprint, unconnected to any exterior wall. Unfortunately there is no solution to this problem, and restarting the program for a different layout is the only workaround.

### 4.3 Project Evaluation

The fitness function successfully converges on a maximum value over time and as the number of iterations increases. This is shown in the graph below:



Another method of evaluation was to get to get user feedback. Five students who all study Engineering at University were selected to evaluate the effectiveness of the program of generating public house layouts. They were told to examine the layout and give each image presented with a score out of ten. The students only had the images available to form an opinion on the images, and the constraints used in the program were not disclosed. The results are disclosed below:

Layout Number	Student 1 Rating	Student 2 Rating	Student 3 Rating	Student 4 Rating	Student 5 Rating <sup>†</sup>	Average Rating	Master Rating
0	4	4	3	3	3	3.4	960
1	6	8	6	4	4	5.6	522
2	7	9	8	6	5	7	721
3	5	8	8	4	6	6.2	349
4	5	8	6	7	6	6.4	792
5	4	4	1	2	6	3.4	480
6	N/A	N/A	N/A	N/A	N/A	N/A	N/A
7	2	4	3	3	7	3.8	-296
8	1	2	3	2	3	2.2	-823
9	3	7	5	3	8	5.2	143
10	7	5	6	8	7	6.6	582
11	7	7	7	4	6	6.2	427
12	5	6	8	6	8	6.6	1096
13	7	8	7	5	8	7	1285
14	9	8	5	5	8	7	237
15	N/A	N/A	N/A	N/A	N/A	N/A	N/A
16	9	8	6	7	7	7.4	785
17	2	6	2	1	2	2.6	-110
18	7	7	4	4	7	5.8	-22
19	8	8	7	3	8	6.8	795
20	6	8	6	3	8	6.2	191
21	10	8	8	8	9	8.6	1380
22	3	6	7	5	8	5.8	519
23	4	6	5	1	6	4.4	-343
24	10	9	7	8	10	8.8	79
25	3	8	3	1	7	4.4	-605
26	7	3	6	5	8	5.8	809
27	7	5	6	6	9	6.6	600
28	1	2	2	2	4	2.2	-795
29	6	1	6	6	5	4.8	926
30	N/A	N/A	N/A	N/A	N/A	N/A	N/A

These ratings were noted, averaged and then compared to a 'master rating', which was the result produced by the objective cost function. Three results also have the value 'N/A', which corresponds to an image that was produced that was bugged, meaning the layout created no longer remotely represented a public house.

Unfortunately, however, a correlation between the user's ratings and the master ratings was not able to be determined. Though there were some circumstances in which a high master rating did produce a decent rated user score, such as layout 21, this was directly contradicted in scenarios in which a very low master rating was met with the highest average user rating given, as seen with layout 24.

Upon seeing this lack of correlation between the two ratings, more user feedback was required in order to determine what constraints the users used to determine the rating of the layout. The student's reported a general feeling that the rooms felt too big, and that having such a large gulf between rooms felt "weird" and "uncanny". The students also took a particular interest in the placement of the interior wall, a part of the project which was not included in the constraints nor able to dynamically change positions. This resulted in an otherwise good layout being written off as a poor layout due to unusual interior wall placement. The ratings were also influenced rather than by the positioning and placement of the rooms, but by the shape of the footprint itself. Many of the cross-shaped footprints suffered point losses due to the shape itself being "unlikely" and "unheard of".

To include the additional aesthetic constraints suggested by the students were require a vast increase of the complexity and run-time of the cost function. As the cost function is required to be run so often, expanding the run-time would cause the program to produce layouts at an unreasonable rate, making it difficult to both use and test.

## 4.4 Self Evaluation

At the end of this project, there any many things that I would do differently in hindsight. Firstly, I would take more care in the planning stages and phase of the project, and use this knowledge to produce a timeline plan that was more suitable to what was likely to take the most time. As a result, the timeline plan shown in the Introduction looked like this:

WEEK	1	2	3	4	5	6	7	8	9	10	BREAK	11	12	13	14	15	16	17	18	19	BREAK	20
TASK																						
Choosing project																						
Research																						
Writing Intermediate Report																						
Implementation of project																						
Testing																						
Evaluation																						
Writing final report																						

Figure 4.2: Actual timeline of the project

As shown, there was a large deviation in the beginning from the initial plan. This occurred due to the difficulties found in choosing a project to begin with. The expanding scope of the project, along with the different directions and possible solutions meant a lot of ideas and implementations were started, but then later scrapped. This had a further knock-on effect and reduced the amount of time available for research, resulting in the issues regarding a lack of knowledge of the existence of geometrical libraries such as SymPy, which would have reduced a lot of time spent on coding.

The implementation of the project, mostly in the form of the Python programming, also proved to be a much larger task that originally envisaged. The initial issues, such as the lack of research, resulted in a lot of restarts. The way that the project's scope and solution changed on a regular basis also meant that the code created had to be polymorphic in design, so that it could easily expand to additional requirements later on down the line. This polymorphic design also had a drastic affect on how long the program took to create, and it is difficult to design a program around specifications that you are unsure of yet.

The testing and evaluation stages of the plan were something I would also like to have spent more time on. With the project write-up looming, it was a necessary step to limit the testing and evaluation to the space of only a week, which was compounded with the long runtimes of each iteration of the program, making it much more difficult to test and look for bugs in the code. During the testing phase of development, no bugs besides Bug-1 was found. Then, whilst preparing the large amount of images to be evaluated by the Engineering students, broken layouts were found, which was very unusual and proof that more time was needed to have been spent on the testing stage. The consistent delays and large amount of time spent on other areas limited the time leftover to work on the final report, which I would have much preferred to be longer, allowing more time to evaluate and write a more articulate piece on this project.

However, a few areas in the project went well. Despite the implementation of the project taking much longer than expected, the code itself, bar from a small few bugs, works very well. The code is designed in a way to make it very easy to expand and scale, with



simple ways to add more interior walls, or even add additional rooms and constraints. I believe the structure, even without its modularity, to be clear, concise and well labelled.

Even though the selection of a project was a lot more time-consuming than planned, I believe that the time spent deliberating which avenue of public house design/generation proved worthwhile long-term, as I think that public house floorplan generation is interesting, an unexplored area of research and well as having a lot of potential usage scenarios in the future.

## 4.5 Potential Improvements

There are many areas in which the program could be improved, in respect to not only the functionality and quality of layouts, but also a more aesthetic and impressive display method of the layouts themselves upon completion.

### 4.5.1 Interior Wall Changes

The interior walls could have done with a large amount of changes to the way that they are designed and the way that they behave. Firstly, it would have been good to implement dynamically allocated interior walls, to prevent the error space from appearing with the interior walls themselves, meaning that the interior walls would no longer be placed in areas that left spaces behind that were too small to be considered effective drinking space.

Another way the interior walls could have been improved is by including doors at some position on them. This would allow not only a more realistic floorplan to be created (as all traversable interior walls have doors), but it would also have added an additional constraint preventing rooms from being placed in front of an interior wall door that may create issues for patrons.

### 4.5.2 Footprint Changes

The footprints themselves also had a lot of scope for improvement. The possibility of changing footprints to be comprised of any shape would have been a more realistic way of displaying a pub, especially since not all public houses are comprised of exclusively horizontal and vertical lines. This change would have vastly increased the complexity of the program, as well as the size of the search space.

The changes to the shape of the initial footprint would also allow for additional features to be implemented to both the interior walls and the placements of the rooms. The

interior walls and walls of the rooms could have as a result been placed at any rotation within the space, allowing for much more intricate designs to be created.

### 4.5.3 Room Changes

The implementation of including additional rooms would also have been a welcome change to the program. With the current basic outfit of only three rooms, new rooms such as a kitchen, a fridge or a smoking area could also have been implemented in order to create a much more realistic layout.

A few aesthetic changes could also have been explored, especially in regards to the bathroom. During the user evaluation, many of the students inquired as to why the bathroom was one single room, and not separated by gender. Although this would not increase the complexity or effectiveness of the layout produced, it would help the end-user understand the images better.

# Chapter 5

## Conclusion

### 5.1 Summary

Overall, I am satisfied with the way that this project has turned out. Although the results may not look aesthetically particularly impressive, the algorithms used to search the space and the amount of effort spent on coding and writing up the functionality to complete the project is something that I am proud of. The program produces a layout of a pub given a particular footprint and interior wall very effectively, despite room for improvement.

The project also however did have a few hiccups along the way, the changing scope of the project meant many different revisions had to be made, perhaps compromising the scale of the project that was originally envisaged. If I was to do this project again, I would spend more time looking for a more efficient tool-set to complete the task, and spend less time working on features for testing purposes but were not usable later in development.

Even though the user-evaluations did not correlate with the scores generated by the cost function, I believe that the cost function still does create good layouts with sensible room placements. This project is also something I am very interested in working on further in the future.

### 5.2 Future Work

In the future, I would like to make this project Open Source, using the MIT License. This would allow for the rights of anybody who obtains the software to copy, use, modify, merge, distribute, publish and sell copies of the software without restrictions. I would also like to continue to work on this project, implementing changes discussed in the evaluation section as well as implement some features such as furniture and different methods of displaying the layout generated.

#### 5.2.1 Furniture

In the beginning of the project, a large amount of research was deployed into the placement of furniture within the interior of the pub. However it was deemed too complex to include, and the amount of time required to implement was too much. It

would be a good idea to work on implementing furniture in a sensible and aesthetically pleasing manner in the future. Having furniture placement in the program will also allow for a re-evaluation of the number of patrons within the footprint, as it is possible to directly attribute one patron to a particular seat.

### 5.2.2 Displaying Results

In the future, I would also like to see the implementation of a different way to display the layouts once they have been calculated. Though the Pillow Python library creates a simple, clear and concise image; it is not particularly aesthetic or impressive. Firstly, the extrusion of the exterior/interior/room walls along the z-axis in a 3D space would allow for the exporting of the layout to 3D rendering software, such as Blender[21] to be displayed with both proper texturing and lighting.

# References

- [1] Alex Clark and Contributors. Pillow. <https://pillow.readthedocs.io/en/stable/>.
- [2] Fredrik Lundh and Contributors. Python imaging library.  
<https://www.pythonware.com/products/pil/>.
- [3] British Beer and Pub Association (BBPA).  
<https://beerandpub.com/statistics/pub-numbers>, 2017.
- [4] Pascal Muller Yoav I H Parish. Procedural modeling of cities.  
<https://graphics.ethz.ch/Downloads/Publications/Papers/2001/ppar01.pdf>, 2001.
- [5] Jon Kelly. Even ulan bator has irish pubs.  
<https://www.bbc.co.uk/news/magazine-27504204>, 2014.
- [6] RightMove.co.uk. <https://www.rightmove.co.uk/commercial-property-for-sale>.
- [7] North Norfolk District Council. Toilet provision in premises open to the public.  
<https://www.north-norfolk.gov.uk/media/1831/publictoiletpolicy2014.pdf>, page 9, 2014.
- [8] Licensing act uk.  
[http://www.legislation.gov.uk/ukpga/2003/17/pdfs/ukpga\\_20030017\\_en.pdf](http://www.legislation.gov.uk/ukpga/2003/17/pdfs/ukpga_20030017_en.pdf), 2003.
- [9] The regulatory reform (fire safety) order.  
[http://www.legislation.gov.uk/uksi/2005/1541/pdfs/uksi\\_20051541\\_en.pdf](http://www.legislation.gov.uk/uksi/2005/1541/pdfs/uksi_20051541_en.pdf), 2005.
- [10] Disability discrimination act uk.  
[http://www.legislation.gov.uk/ukpga/1995/50/pdfs/ukpga\\_19950050\\_en.pdf](http://www.legislation.gov.uk/ukpga/1995/50/pdfs/ukpga_19950050_en.pdf), 1995.
- [11] Cityengine. <https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview>.
- [12] Simon Haegler Andreas Ulmer Luc Van Gool Pascal Muller, Peter Wonka.  
Procedural modeling of buildings.  
[https://www.researchgate.net/profile/Peter\\_Wonka/publication/220183823\\_Procedural\\_Modeling](https://www.researchgate.net/profile/Peter_Wonka/publication/220183823_Procedural_Modeling), 2008.
- [13] Peter Wonka Luc Van Gool Pascal Muller, Gang Zeng. Image-based procedural modeling of facades.  
<http://www.cis.pku.edu.cn/faculty/vision/zeng/pdf/MullerZWG07tog.pdf>, 2007.
- [14] SideFX. Houdini. <https://www.sidefx.com/>.

- [15] Sympy. <https://www.sympy.org/en/index.html>.
- [16] Zeyang Li Maneesh Agrawala Vladlen Koltun Paul Merrell, Eric Schkufza.  
Interactive furniture layout using interior design guidelines.  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.261.1441rep=rep1type=pdfcite.preaid-75>,  
2011.
- [17] J Panero and N. Repetto. Anatomy for interior designers.  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.261.1441rep=rep1type=pdfcite.preaid-75>, 3rd ed,  
1975.
- [18] Fredrik Johansson. mpmath. <http://mpmath.org/>, 2018.
- [19] Gitlab. <https://about.gitlab.com/>.
- [20] The Building Regulations 2010. Access to and use of buildings.  
[https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment\\_data/file/441786/BR](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/441786/BR), 2010.
- [21] Blender. <https://www.blender.org/>.

# Appendices

# Appendix A

## External Material

During the project I have used the following external resources:

- SymPy for geometry-based functions on lines and shapes.
- Pillow, a PIL fork used to display the layouts and save the images.
- GitLab was used for source code version control (<https://about.gitlab.com>).
- L<sup>A</sup>T<sub>E</sub>X was used for the creation of the final report.



# Appendix B

## Ethical Issues Addressed

Designing a floor plan for real-world use is surrounded by a huge amount of complex legal requirements and moral issues. In this project a few of the ethical issues had to be explored and dealt with. Firstly, issues surrounding the washroom requirements of patrons with disabilities in the pub is not only a legal requirement for any establishment, but it is also a basic human right. As the project developed however, the problem space being researched changed from usable real-world layouts to issues tackling the expanding search space. As a result, toilets with disabled access was not included.

Another ethical issue faced was the effectiveness of a particular layout being used in the event of a fire, or when an immediate evacuation of the building is required. Informative and thorough fire evacuation plans need to be created for any commercial establishment as well as alternative fire doors, meeting points and spaces designed to not bottleneck a crowd are all required. However, this was deemed to be beyond the scope of the project and too complex to be assessed by the cost function.

# Appendix C

## Gant Charts

### 1. Initial Plan

WEEK	1	2	3	4	5	6	7	8	9	10	BREAK	11	12	13	14	15	16	17	18	19	BREAK	20
TASK																						
Choosing project																						
Research																						
Writing Intermediate Report																						
Implementation of project																						
Testing																						
Evaluation																						
Writing final report																						

### 2. Final Plan

WEEK	1	2	3	4	5	6	7	8	9	10	BREAK	11	12	13	14	15	16	17	18	19	BREAK	20
TASK																						
Choosing project																						
Research																						
Writing Intermediate Report																						
Implementation of project																						
Testing																						
Evaluation																						
Writing final report																						

# Appendix D

## User Evaluated Images

Image 0

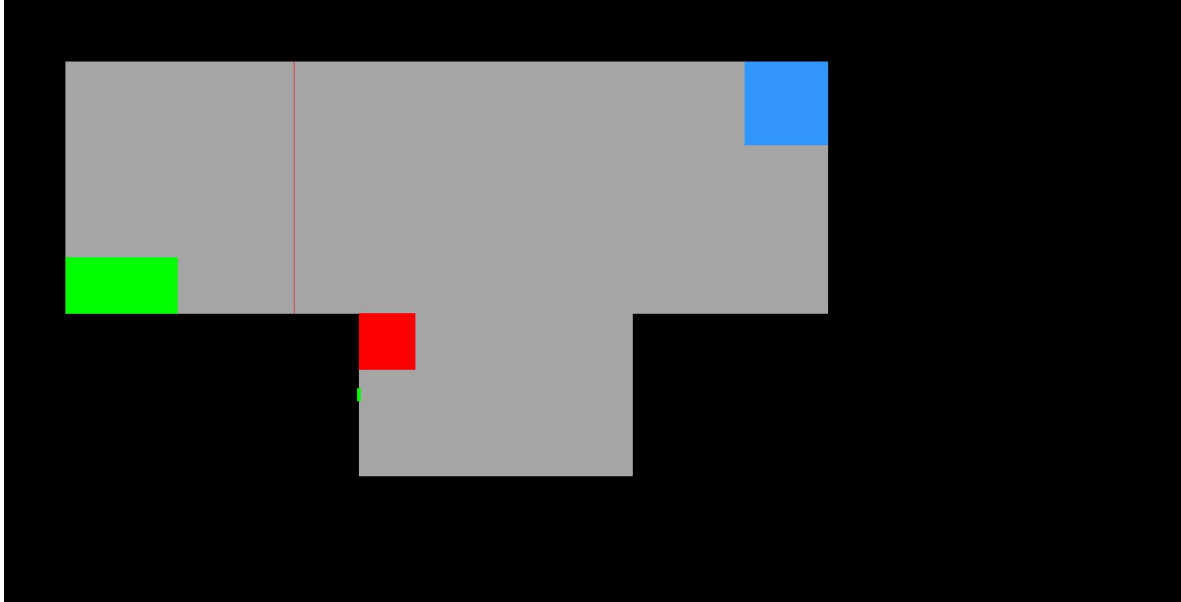


Image 1

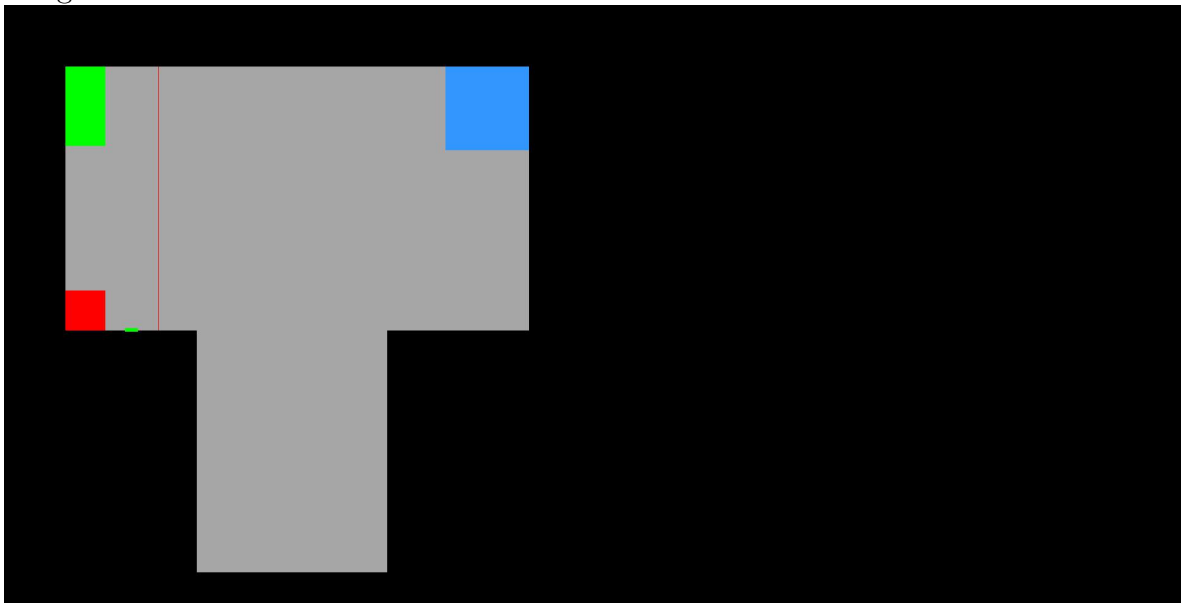


Image 2

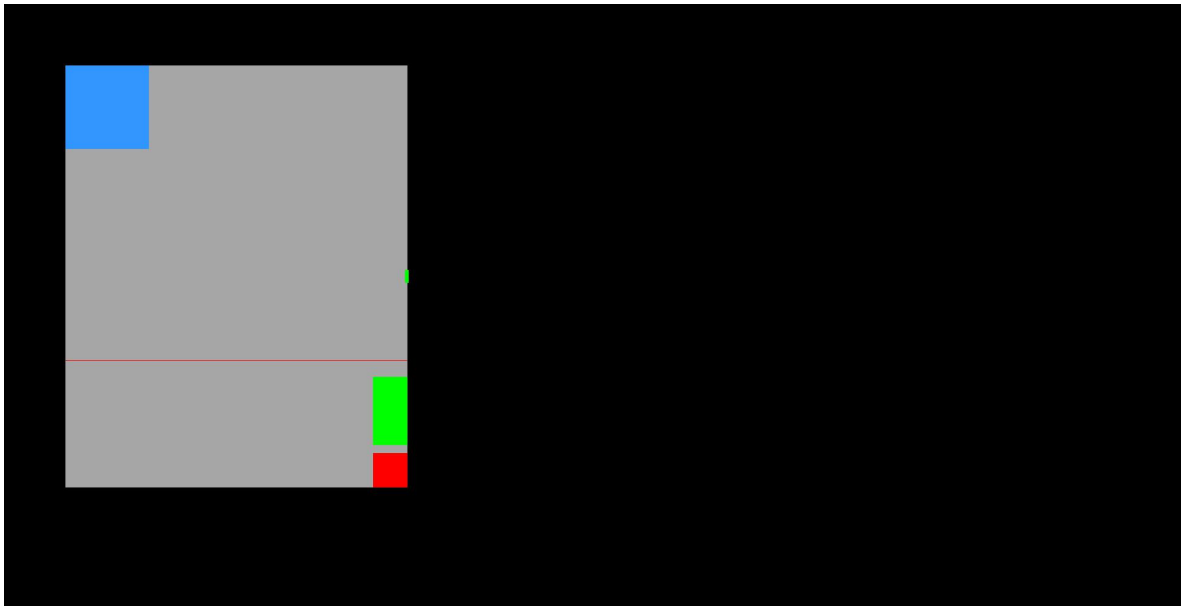


Image 3

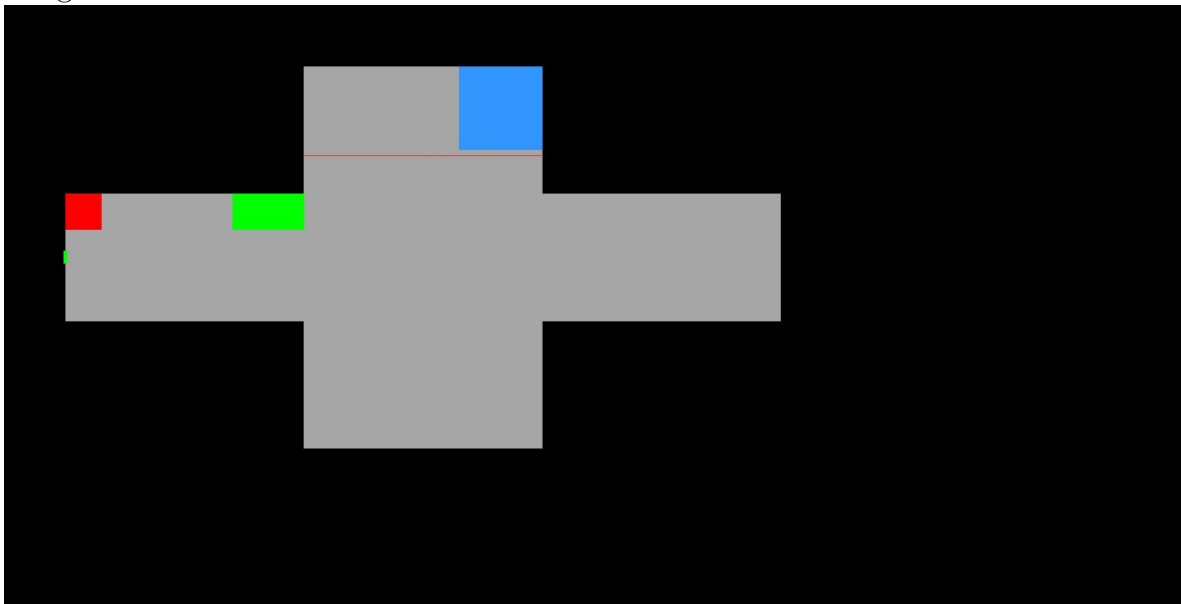


Image 4

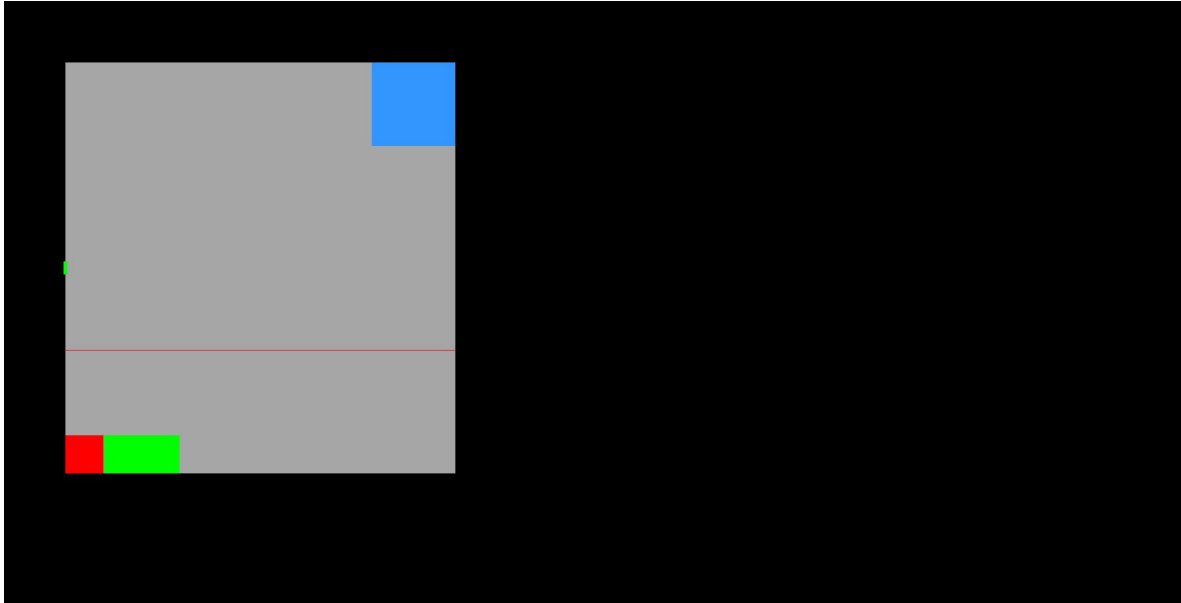


Image 5

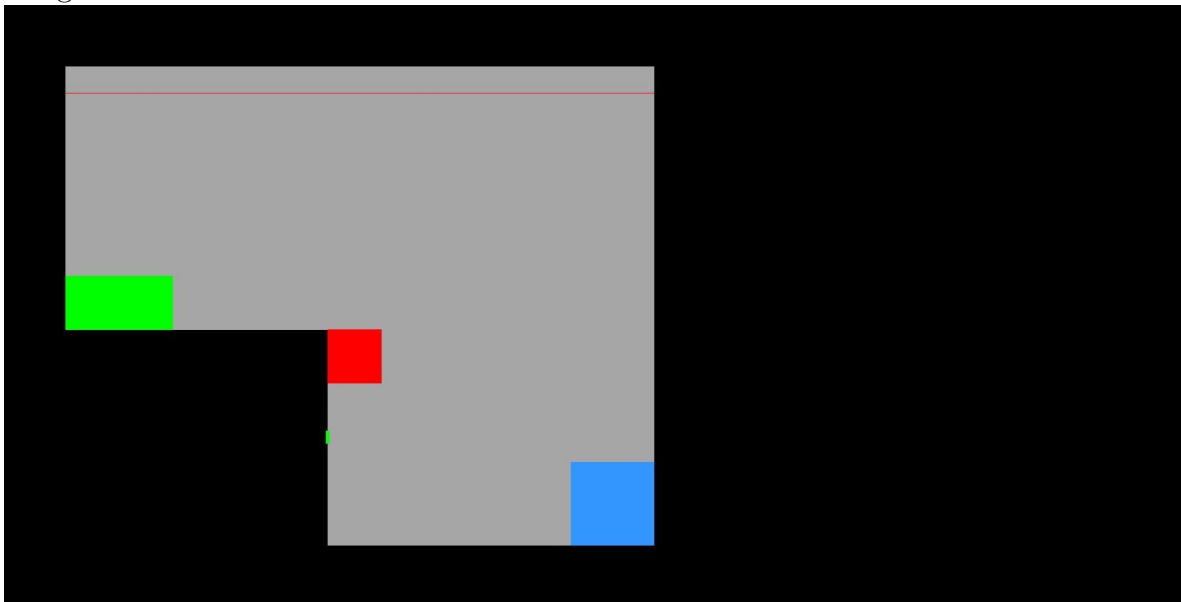


Image 6 - OMITTED

Image 7

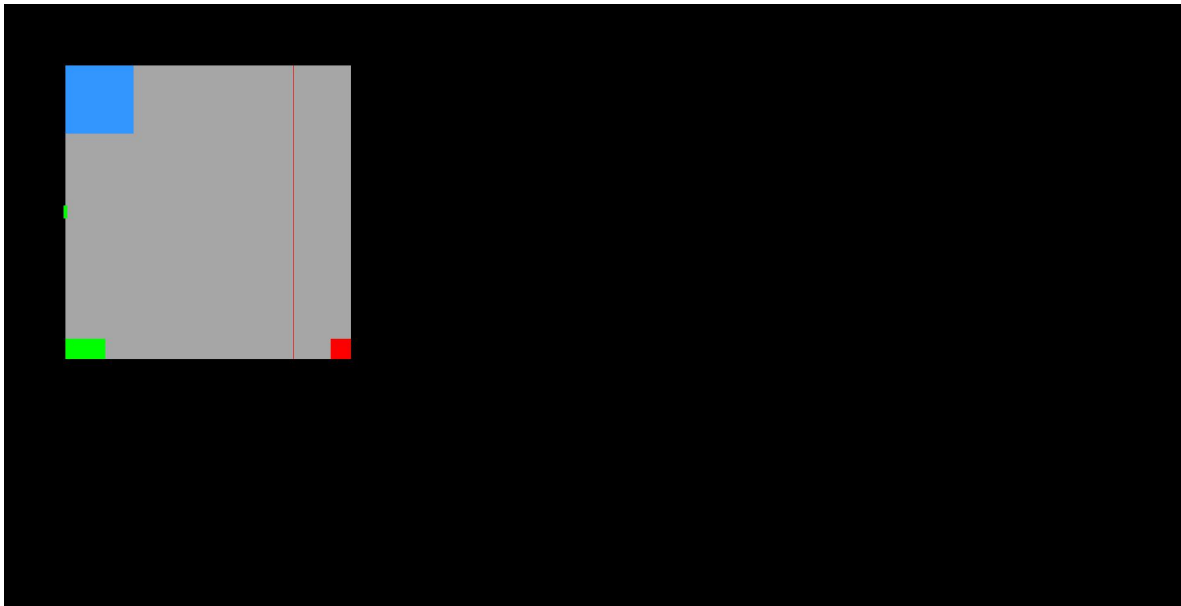


Image 8

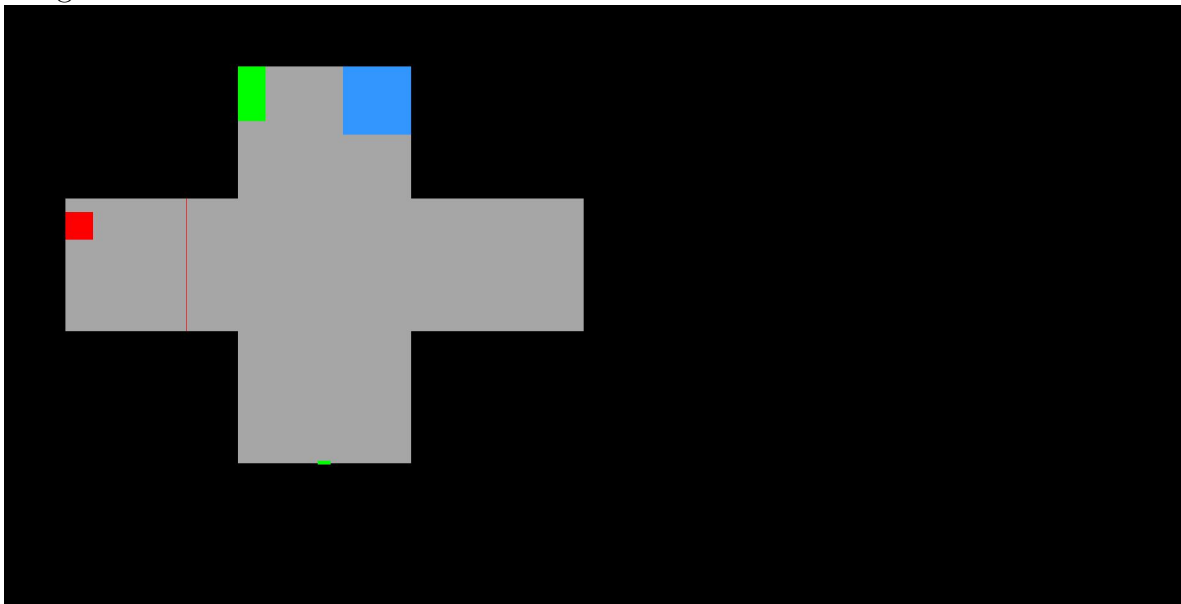


Image 9

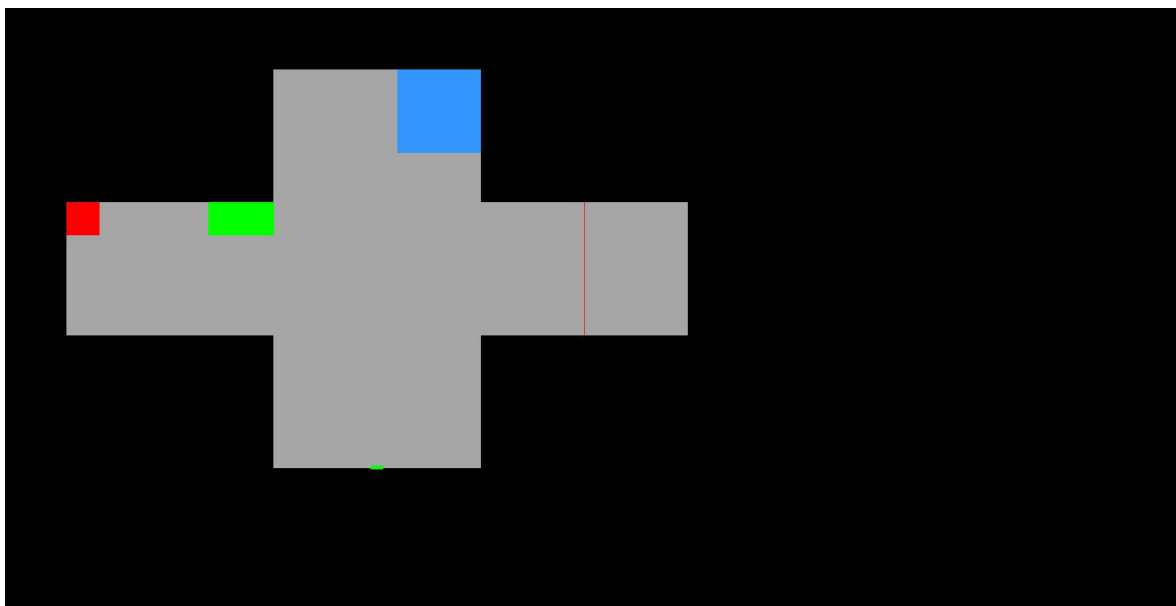


Image 10



Image 11

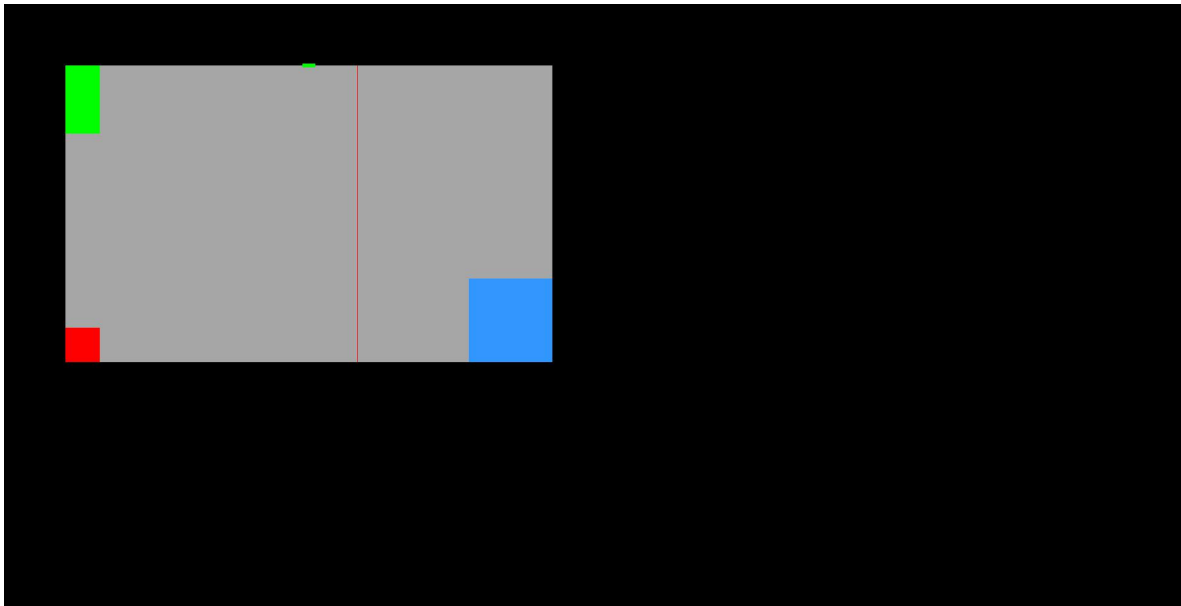


Image 12

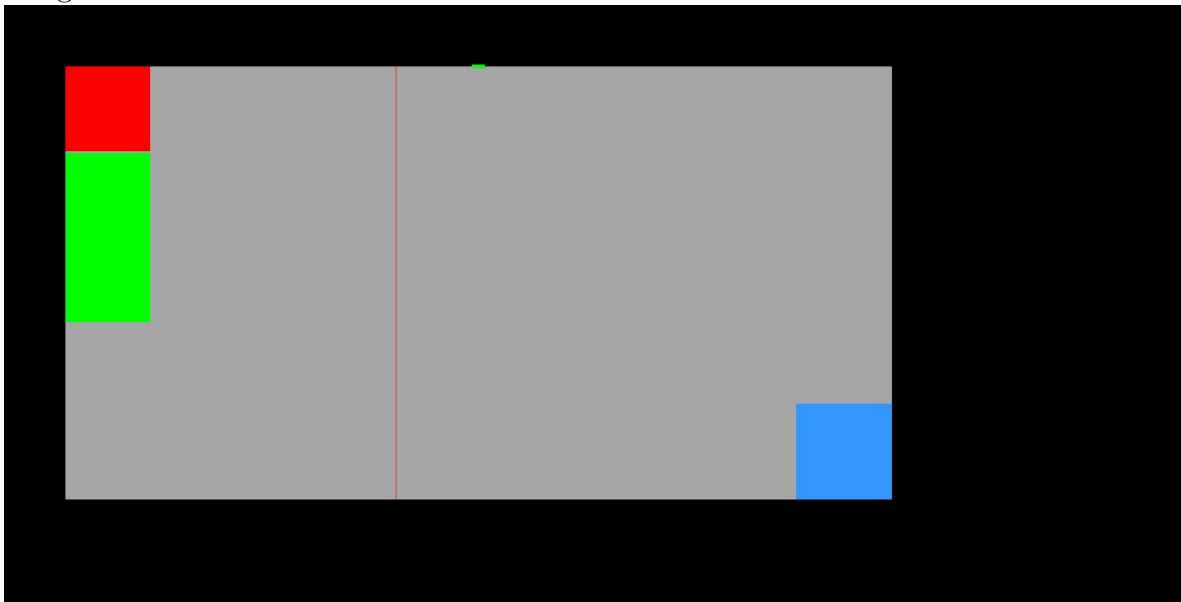


Image 13



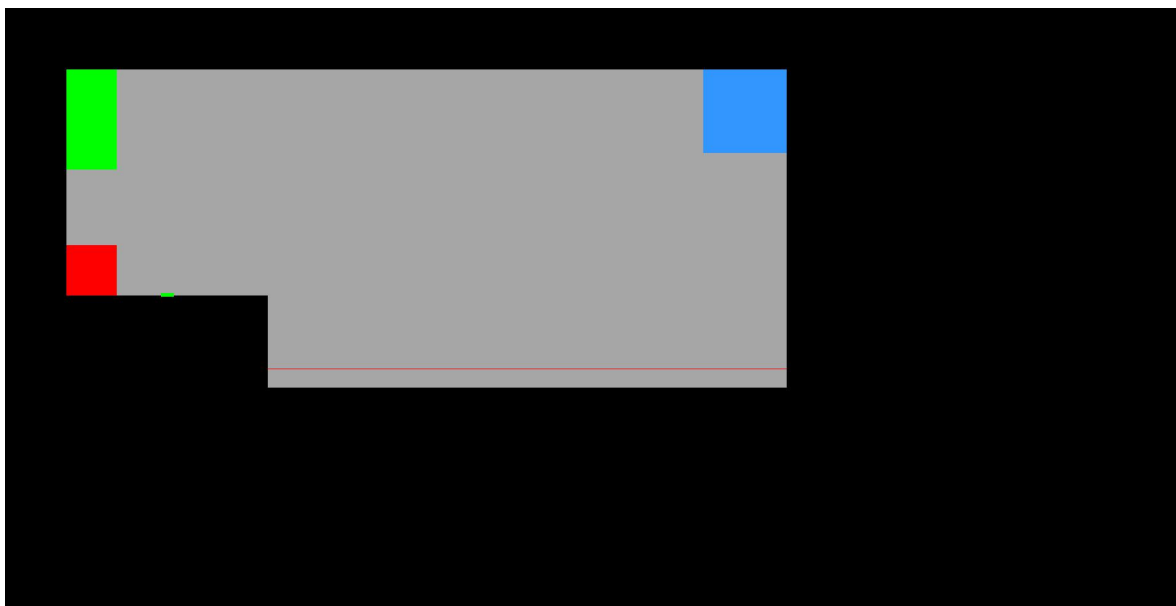


Image 14

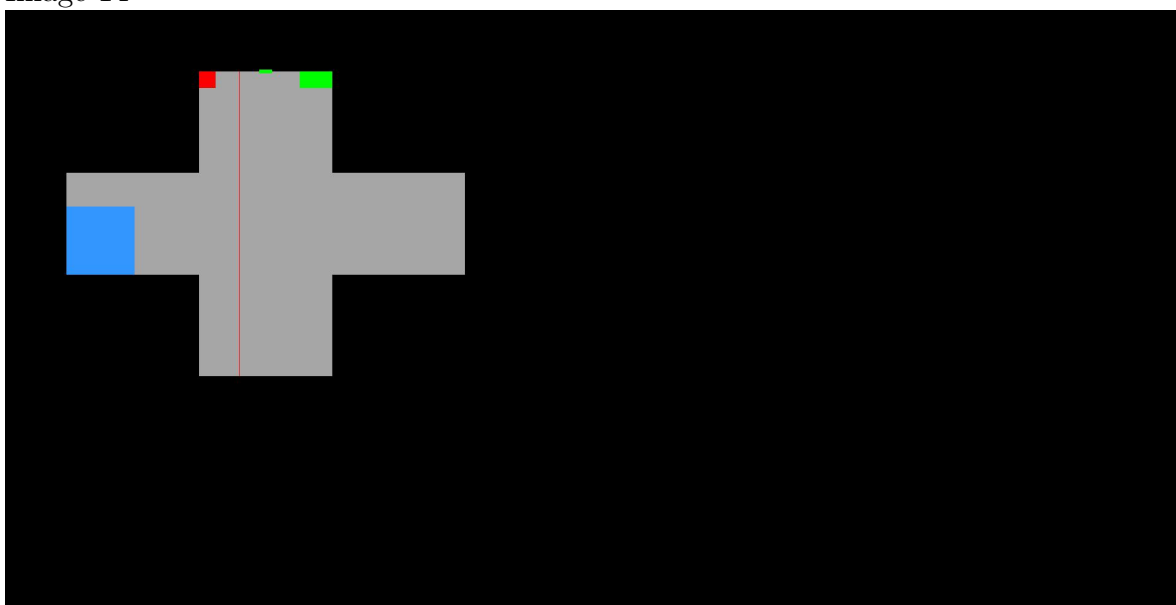


Image 15 - OMITTED

Image 16

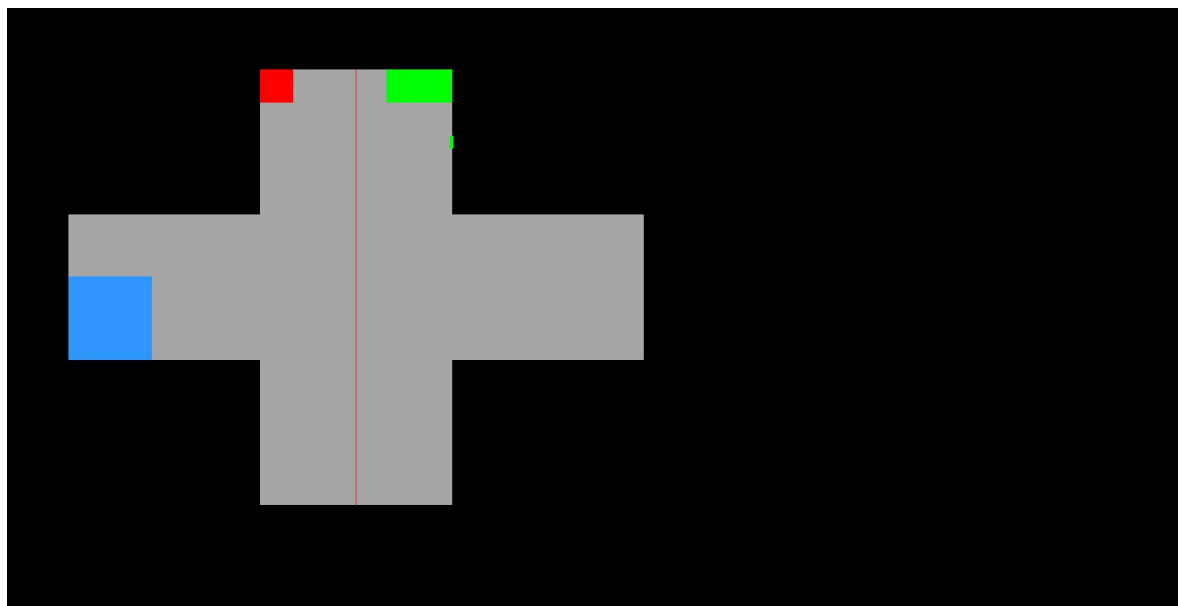


Image 17

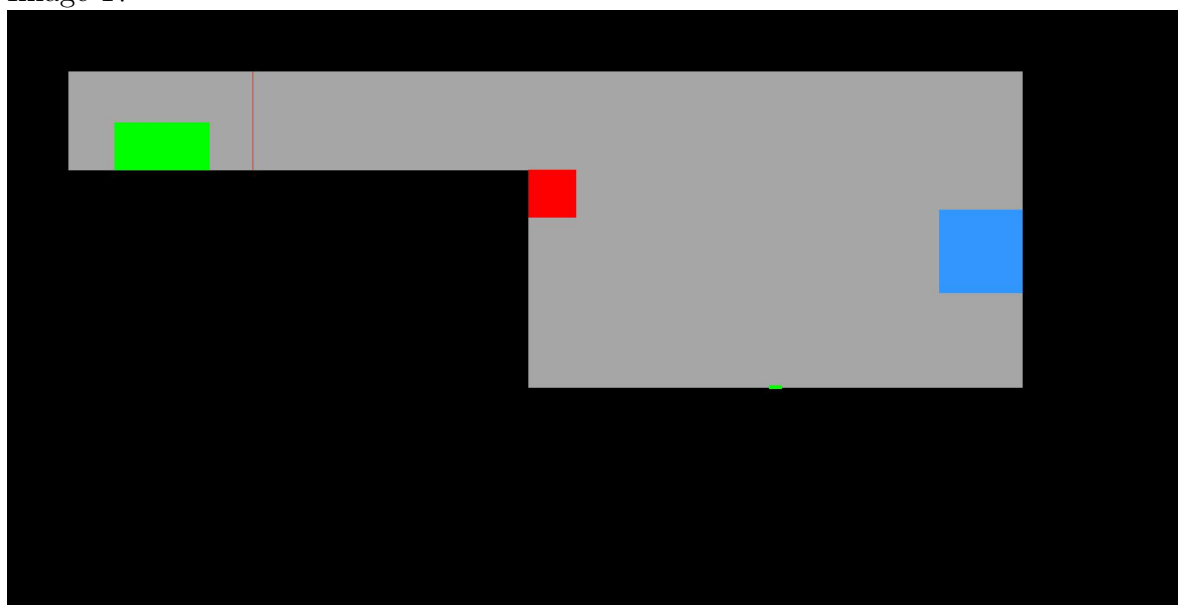


Image 18

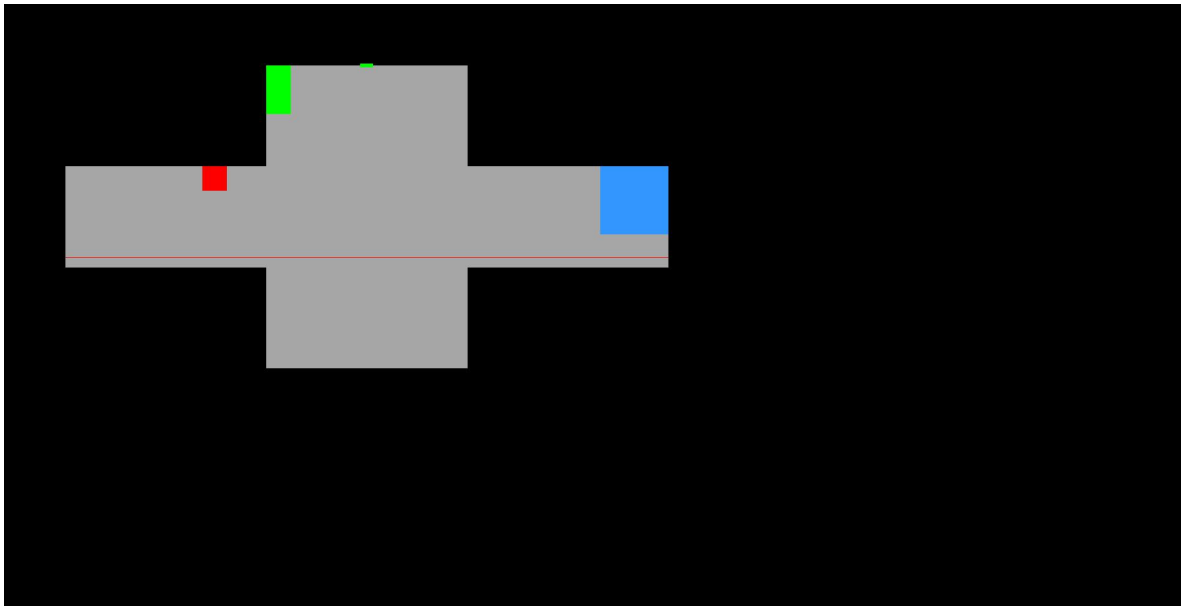


Image 19

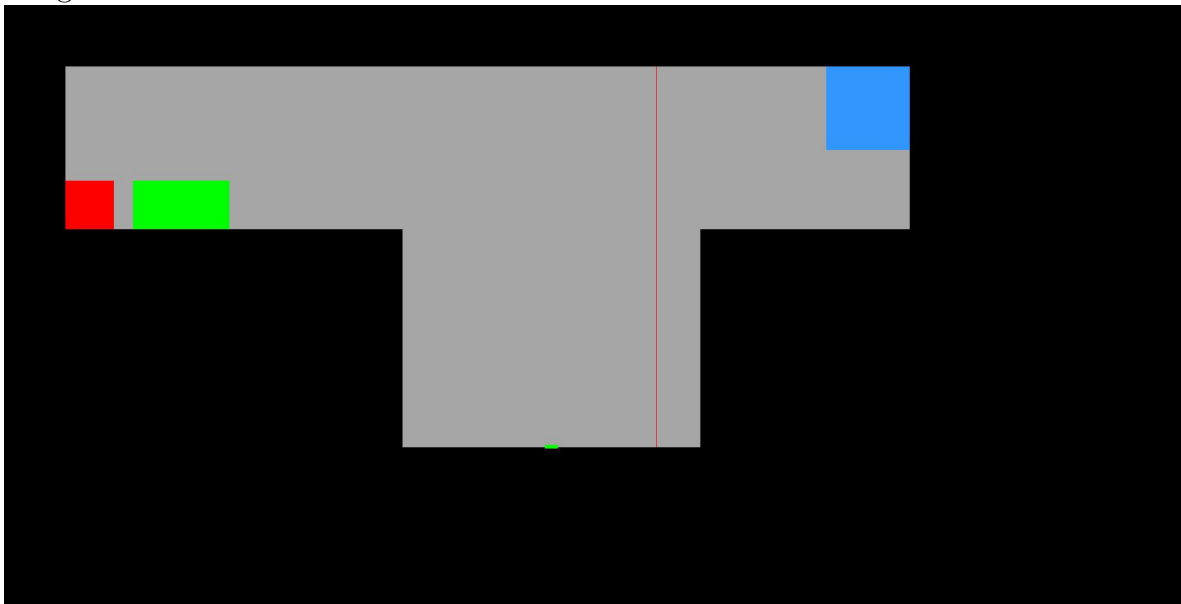


Image 20

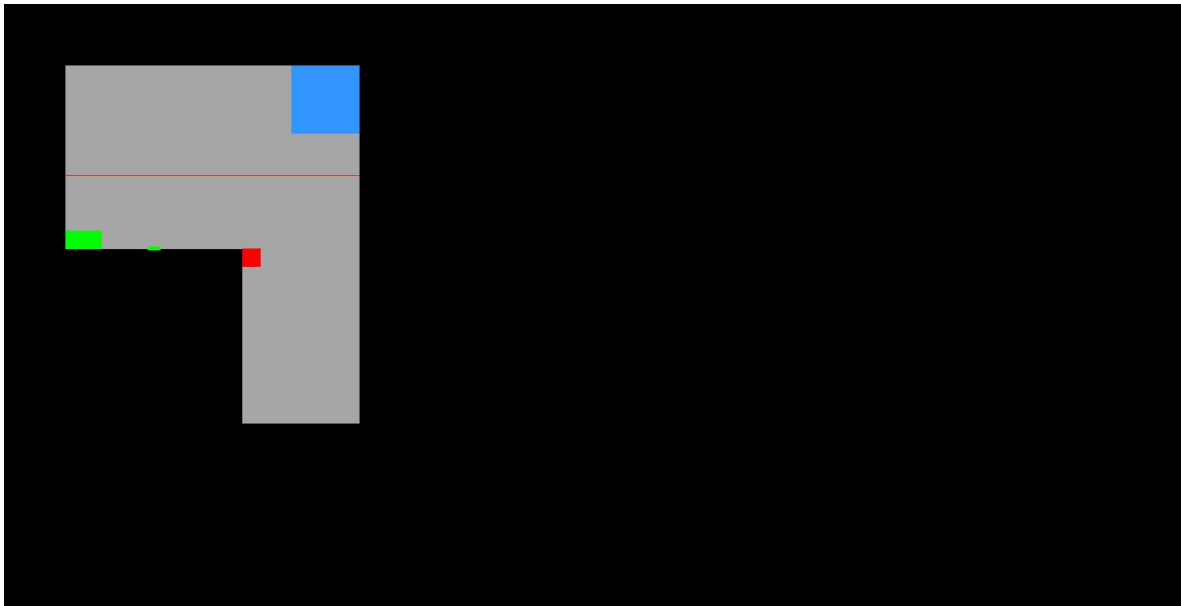


Image 21

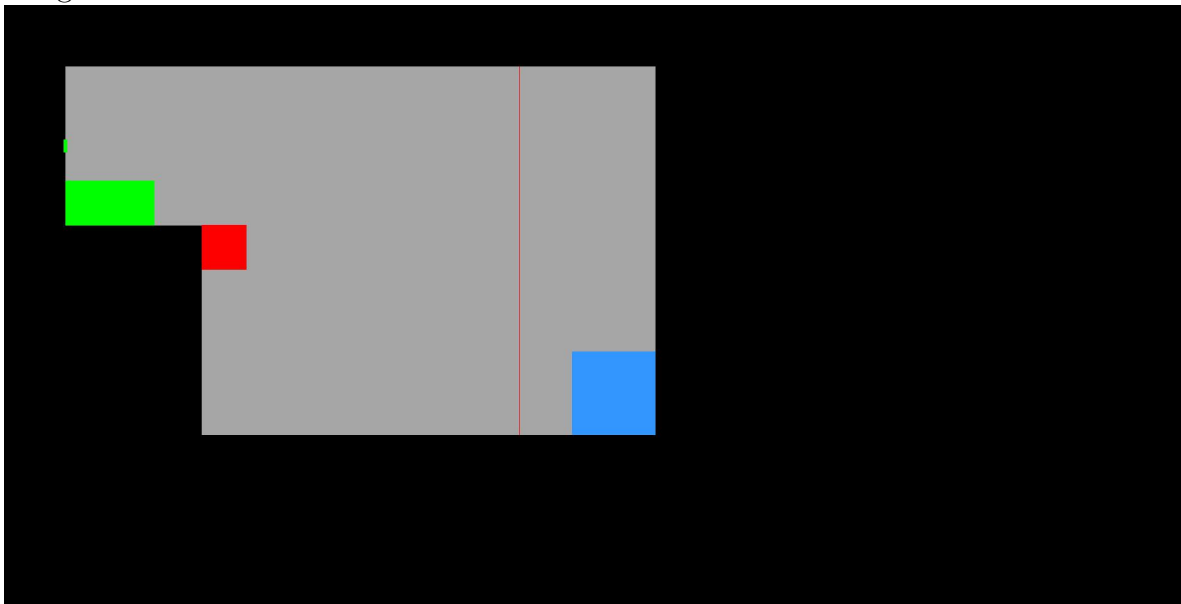


Image 22

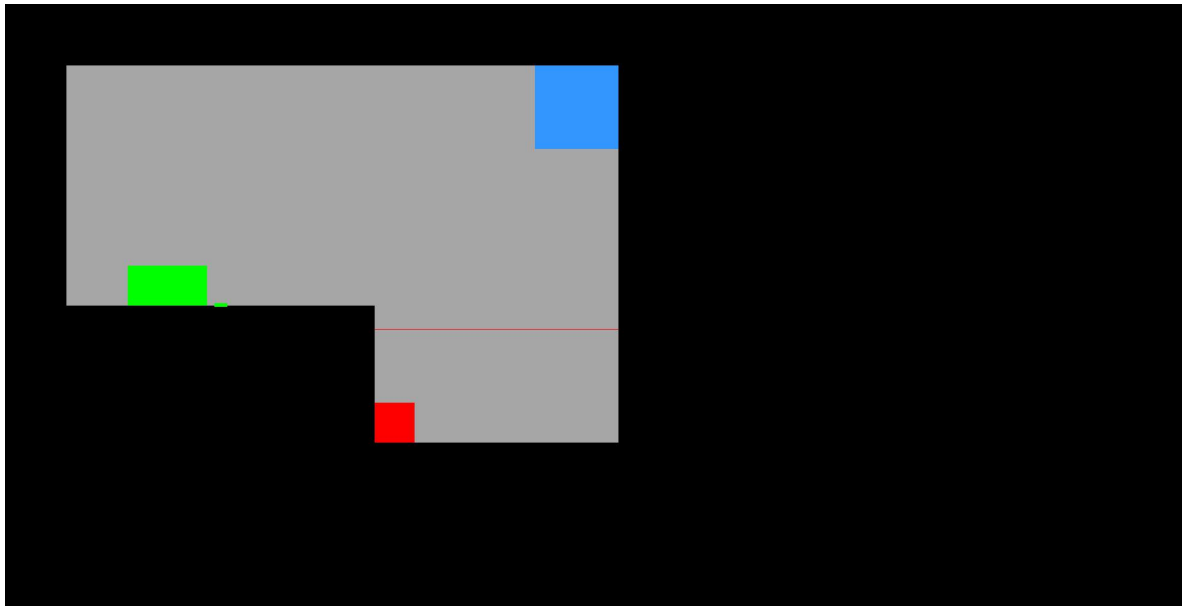


Image 23

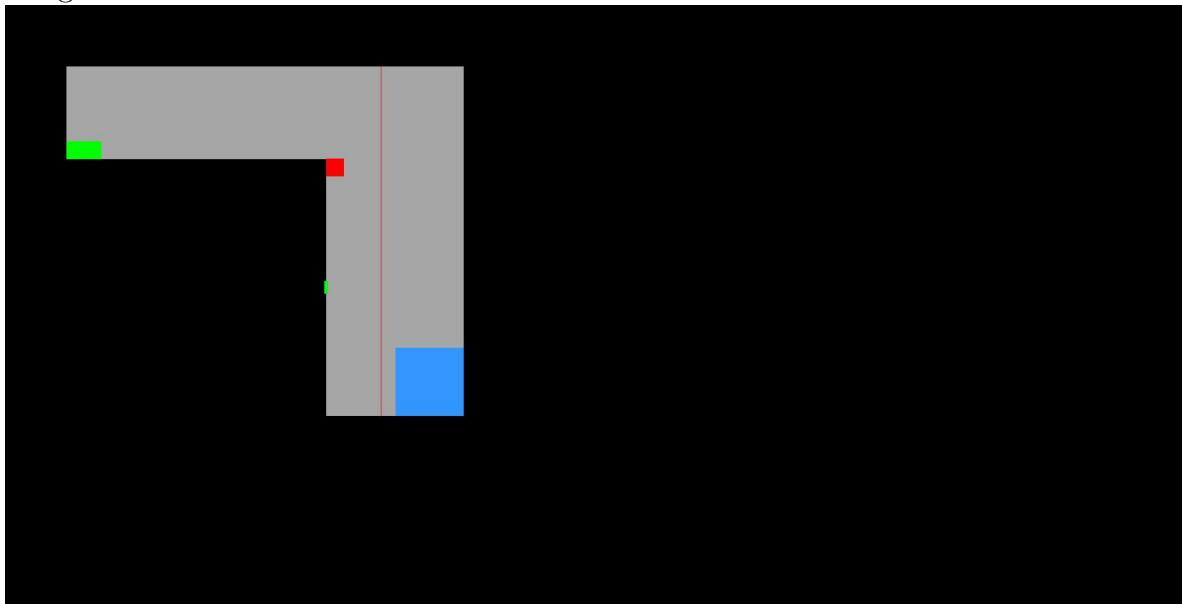


Image 24

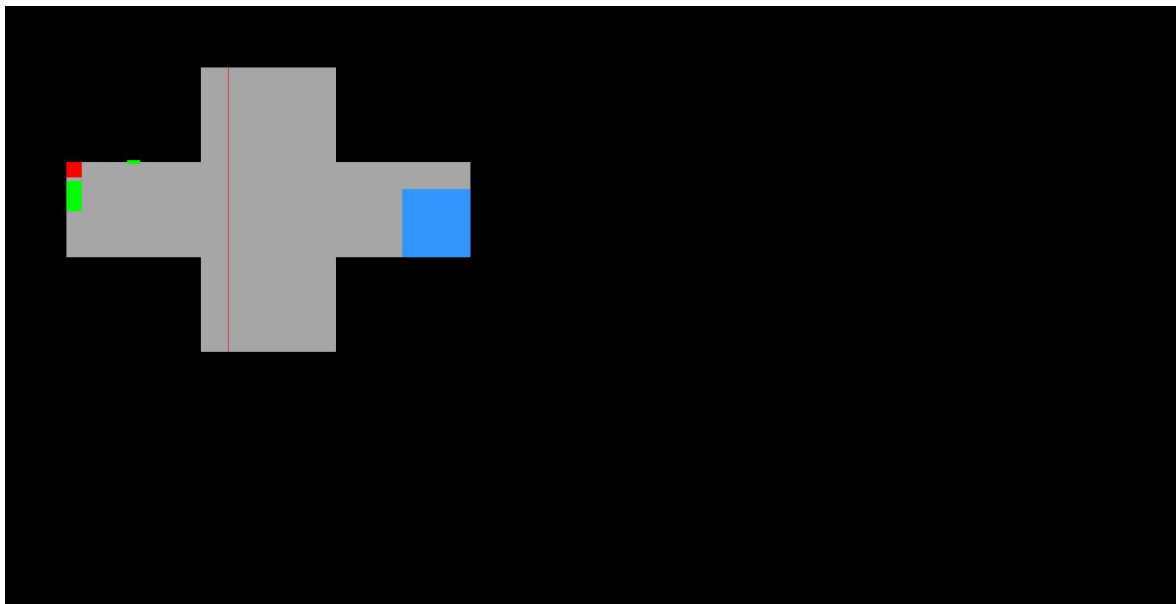


Image 25

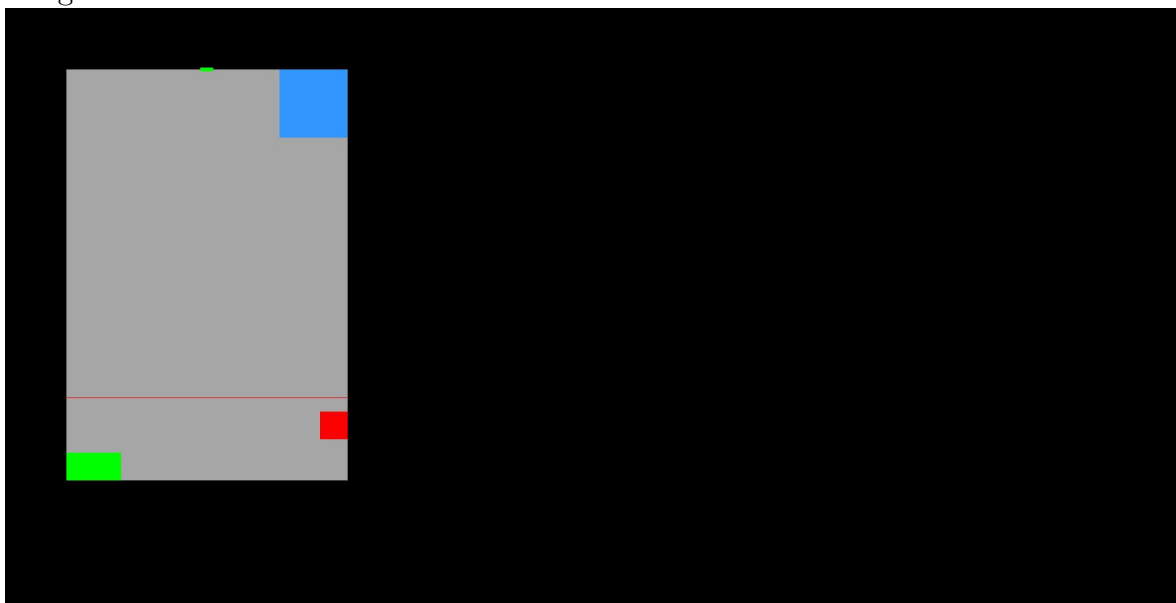


Image 26

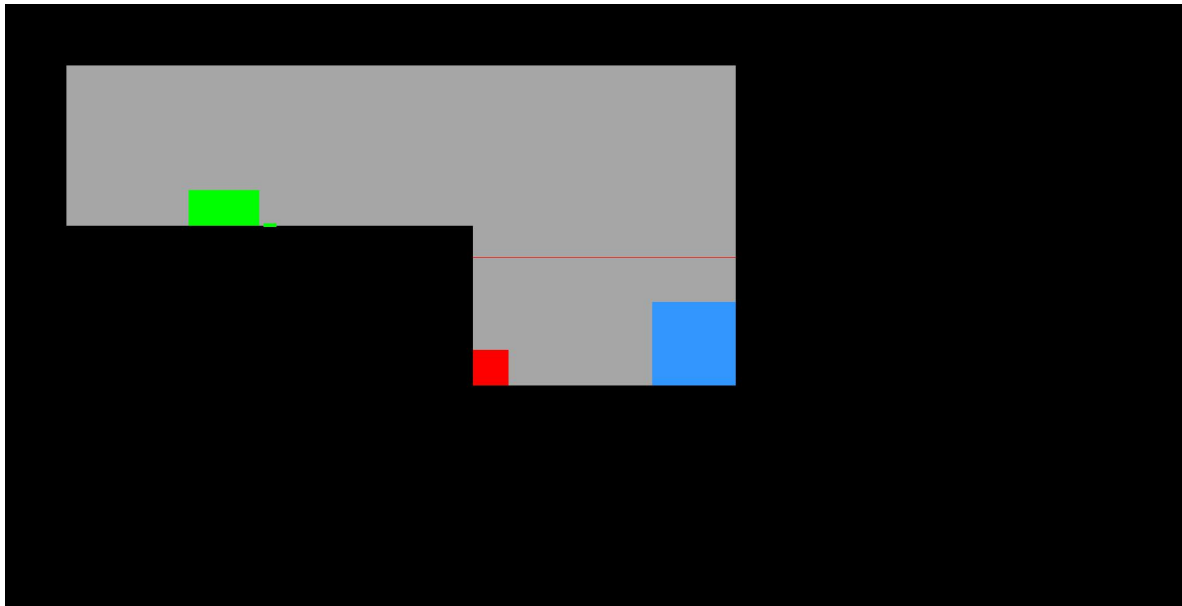


Image 27

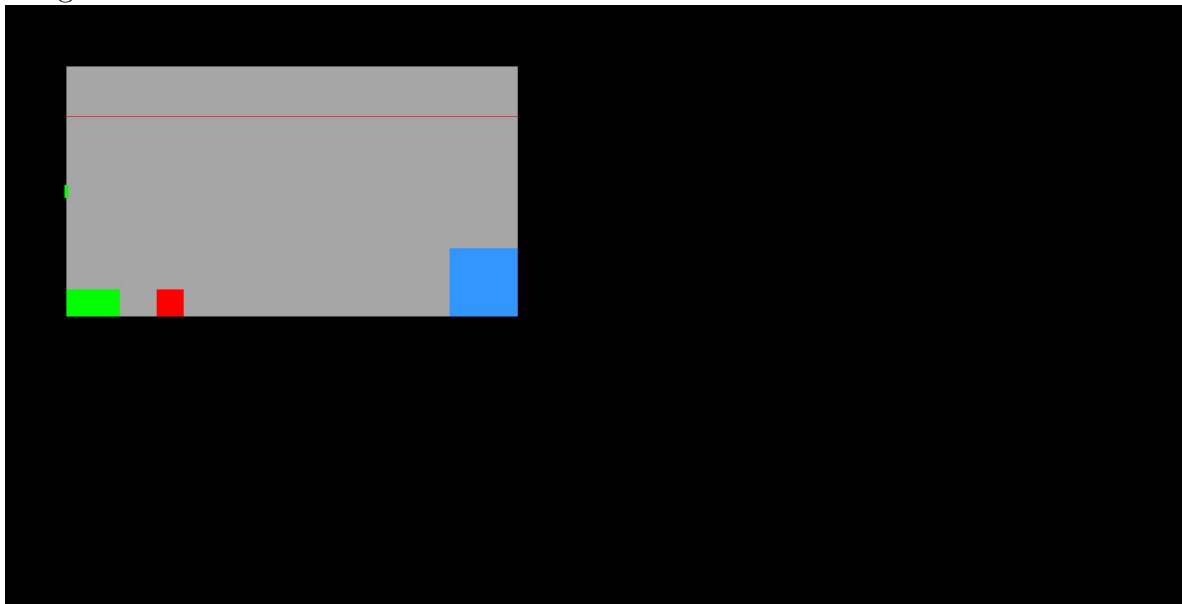


Image 28

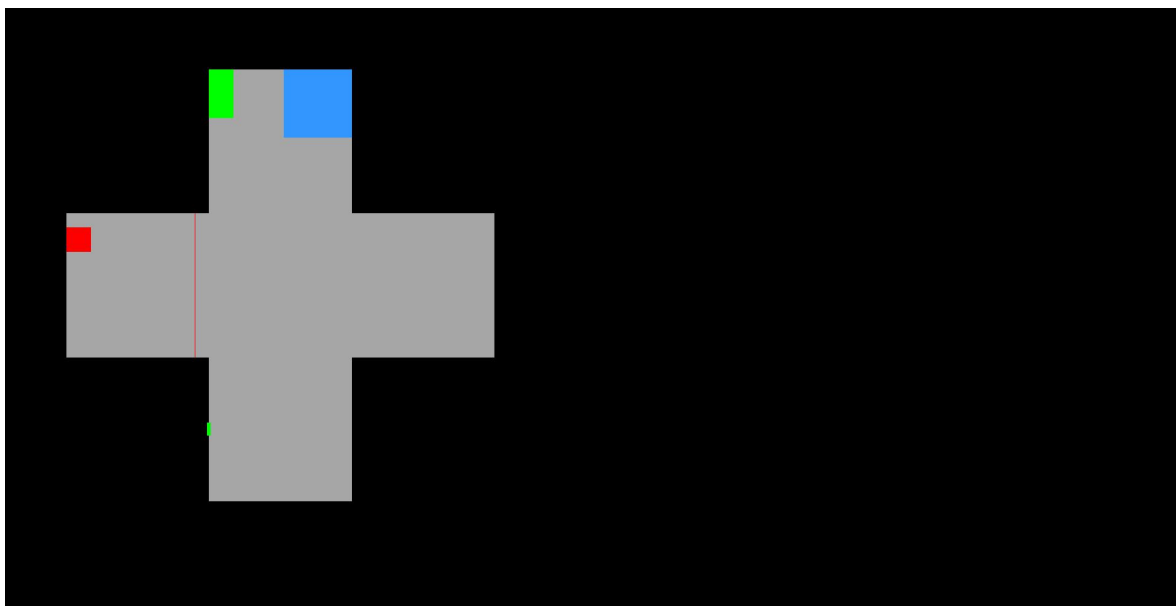


Image 29

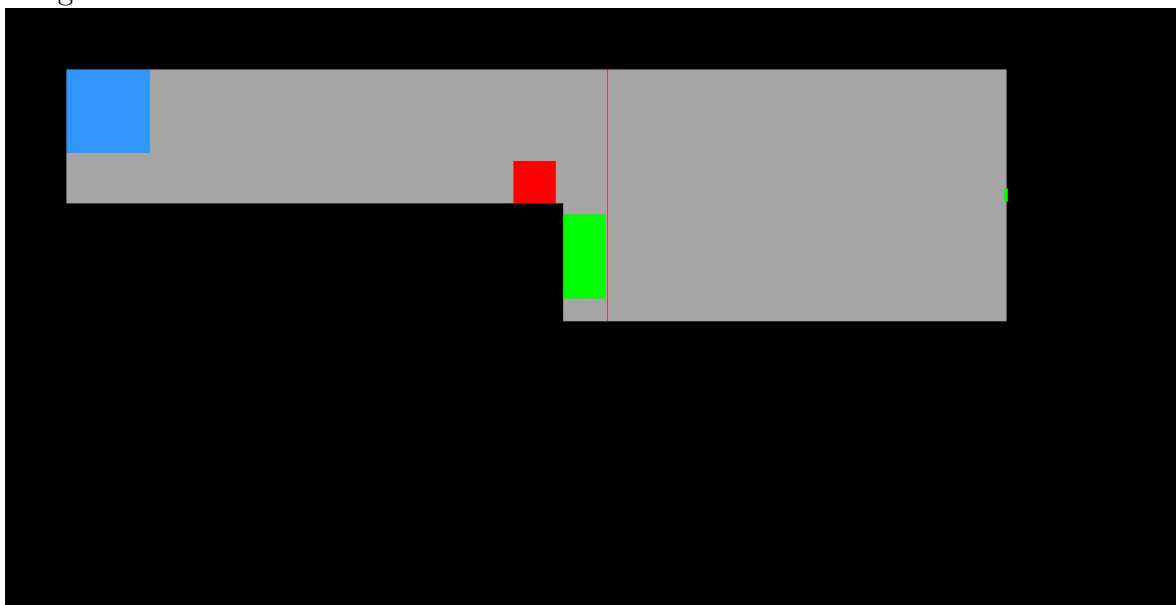


Image 30 – OMITTED