|  |
| --- |
|  |
| City Simulator |
| Year 13 Coursework |

|  |
| --- |
| Felix Andrade May |

Contents

[Proposal: 2](#_Toc526937260)

[Stake Holders: 3](#_Toc526937261)

[Researching the Problem: 4](#_Toc526937262)

[Decomposing the Problem: 6](#_Toc526937263)

[Describing the Solution: 7](#_Toc526937264)

[Describing the Solution – Variables: 8](#_Toc526937265)

[Describing the Solution – Features: 9](#_Toc526937266)

[Limitations: 10](#_Toc526937267)

[Describe the Solution – Algorithms: 11](#_Toc526937268)

[Iterative Development – Evidence: 12](#_Toc526937269)

[Iterative Development – Prototypes: 13](#_Toc526937270)

[Testing the Code: 14](#_Toc526937271)

[Usability: 15](#_Toc526937272)

[Robustness: 16](#_Toc526937273)

[Evaluation – Success Criteria: 17](#_Toc526937274)

[Describe the Final Prototype: 18](#_Toc526937275)

[Maintenance and Development: 19](#_Toc526937276)

[Describe the Approach to Testing: 20](#_Toc526937277)

# Proposal:

Starts out in the centre with some small buildings (lowest size), over time the buildings in the middle grow (size increase after a certain time period, assigned when the building is spawned). New buildings spawn in a north south east or west direction from a new building. These also grow like the central buildings. When a number of buildings reach the highest level it becomes locked so no more can reach that level. The building information and the order of appearance will be stored in databases. The buildings may be presented from above by squares with a colour corresponding to the level of the building. There may be “Zones” such as a marsh, where the max building level is lowered, or the time to grow is increased. It may also be necessary to implement a pause/play button and a quit/stop button alongside a ticking date calendar for a better user experience.

Abstraction would be used to reduce the complexity of an entire city so that only the important details are included in my simulation. I would be romeoving things such as traffic, pedestrians, the econnomy and so on, as this is unnecassary for the core of the programme. There will not be very many inputs to the system, only the users’ ability to stop and start the simulation, though outputs will include both the visual representation of the buildings as well as the information being output to the data base of building information. Computational thinking will alow me to go through every key feature of the programme and break each one down into their constituent parts, and so on until every feature is broken down into all of the parts needed for it to be completed. This will also allow me to easily solve the problem by working through it in small stages. By doing this I will also be able to code using sub-procedures and modular code, this may speed up the coding time because I can duplicate neccassary parts of the code if need be. Computational thinking will allow logical decisions to be made, such as in which direction the next buildings will be generated, whether or not the building is able to progress to the next level and so on. I would use porogramming constructs such as itteration to constantly run through the code generating new buildings and updating old ones as long as the user is allowing the programme to run. I can do this in an IDE which will allow me to test the graphical interface of the programme during development as well as pointing out errors and allowing me to debug the code.

## Stake Holders:

### Geography Teacher:

They teach GCSE Geography students of an average ability who need to have their grades boosted. They need a program that accurately demonstrates the life cycle of a city based on the Burgess Model while also entertaining the student’s enough for them to take in the information. They will use it in class either on a screen in front of the class or tell the students to access it on laptops individually/in pairs. They will need it to accurately represent how a city looks so that the students will be better taught. They will also require the program to be different each time to show that the Burgess model is not always accurate and can change in size/shape. It should also be easy to run as well as fast, so that it doesn’t consume more of the lesson time than it should.

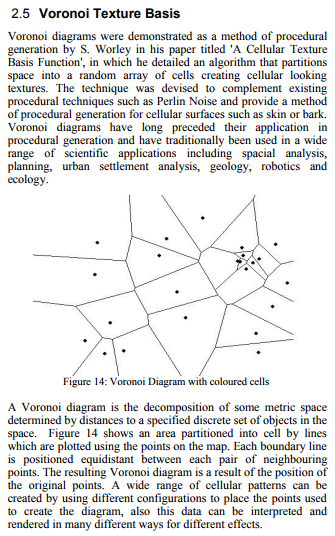
### Geography Students:

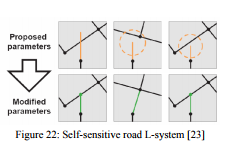
They are taking GCSE geography with grades of E’s, they need a program which will help them learn about Cities, boost their grade and also entertain them enough to make sure they learn. They will use it in class or have it set as homework to go and look at. By using it they will be taught about the Burgess Model and how it would look in a city, this will help them achieve higher grades because it will help them understand the layout of a city, which will help in myriad questions they may be asked in a geography test or quiz in class.

### Student Parents:

They want to learn about their child’s subject, so that if their child needs help the parent will be able to give it. It would also encourage the parents to make sure that their child understands the lesson content if they also know roughly what it is. They will use it in their free time or while their child is using it for homework. By teaching the parent similar content to the student they will be able to help the student if the student is stuck on a piece of work, as well as being able to shame their child for being stupid, as the parent knows stuff the child should know despite not attending GCSE Geography lessons. As this is definitely the most efficient and humane way to educate a child. It will meet their needs by being simple and easy to use and understand because the parents have other things to do that consume more of their time, such as employment and caring for the child as well as household duties.

## Researching the Problem:

On the right is an extract from the paper ***“A Survey of Procedural Techniques for City Generation”*** by George Kelly and Hugh McCabe of ITB, in Dublin. It describes a method of procedural generation which could be used to create roads for my project. I would implement it by randomly creating a spread of points across the map, and then adding these drawn lines, or revealing them as time progresses. I might be able to use this to generate a slightly more natural looking city than simple 4x4 blocks of buildings. Though generating buildings to fit along the roads would be challenging. This can be a good approach to the problem of city generation, as many cities are built on rough grid patterns, though it can create abnormally large grids if parameters aren’t implemented to limit the space between dots. A lot if cities also have curved roads, which this doesn’t account for. Despite this the algorithm isn’t very complicated and can be easy to produce in many programming languages.

L-Systems (also spoken about by George and Hugh in their paper) were used by Parish and Mueller in their *CityEngine* program which was presented in the paper ***“Procedural modelling of Cities”*** to create roads in a real time generation of a city. Originally created to generate plant life and biological objects they can be easily translated across to City design as both roads and branches of a tree share very similar functions and will grow in a similar manner. *CityEngine* uses an advanced form of L-Systems called Self-Sensitive L-Systems, which are “aware” of what is in front of them as the draw themselves, if there is a body of water they will stop (Unless designated a ‘highway’, in which case a bridge will be created to cross the water”, if the meet another road they will form a cross-road, or curve to join a T-Junction to create one. As shown to the right:

They achieved this by creating Global and Local Goals. The Global Goals set out general parameters such as the two different road types, Highways and Streets. Streets are to follow super-imposed geometric patterns and chose the path of least inclination. Local Goals then set out specific requirements for each individual road depending on its own situation. Preferably I would use this to yet more natural looking cities, though the time to learn and program how to create these may be better used on other parts of my project.

Watabou, creator of *Toy Town 3D* and *medieval Fantasy City Gen* used Haxe and OpenFL to create a programme which allows you to randomly generate a medieval style city, with various user controlled parameters, such as the shape of towers, whether walls have towers, bastions, as well as how big the city should be and if there should be \*more roads\*. They used strictly controlled brushes to generate the roads, and a set of “wards” such as castle and farm wards; these are separate files with a small amount of code which generates a polygon, this is able to fit along the “main” roads, while the minor roads, such as streets and alleys, are generated by leaving a gap between the polygons. This works for the generation as it allows a large town to be created with many different wards of different sizes and shapes, while the buildings don’t have to align to the roads as a road in 12th century Europe would not have been of equal width, so if the building overlaps where the road should be or is too far back from the road it is of no consequence and only acts to enhance the realistic look of the plan. The examples to the left show a generated town of a medium size, the lower image has labels which show the different generated wards. The simulation also generates the “landscape” (the waterbodies).

## Decomposing the Problem:

I created a User Interface group as it is primarily separate from the rest of the program, only serving to enhance the user experience. Under this heading exists 2 modules, a play/pause button and a clock/timer.

### User Interface:

The Play/Pause buttons will allow the user to play and pause the simulation, this will stop the timer from counting and appear to stop the simulation, as building generation and progression will be tied into this timer.

The Clock/Calendar is a representation of the timer, it might be represented as a dd/mm/yyyy format, where the days are seconds (a month would be 30/31 seconds). This does have a major influence on the rest of the simulation as the building generation and progression is tied into it, however it is more related to the play/pause button and UI than the buildings (which could run off an invisible timer)

### Buildings:

I separated this category into 3 further modules, the building’s type, the building’s location, building’s direction. All of these will need to read and write to a database containing information on every building (which the user interface doesn’t need to access, however the appearance is integrated into the buildings quite heavily, though I separated them for ease of use, as buildings can still be created without the visual element, though the visual element can’t exist without the building data in the database.)

Building Types is the level of the building (level 1 through 4/5). When a new building is created it will write to the database the level of the building and how long it should take to progress (all buildings start at level 1). Buildings which already exist will be upgraded after the generated time, which will require the module to write to the appropriate building’s row and update the building level.

Building Location will determine exactly where on the map the building is placed, writing pixel coordinates to the database, it will have to check if there is already a building there however. There may be a chance of a building above level 1 generating, and if this building’s first coordinates are on top of a building of a lower level, it will replace the building.

It could be possible to categorise this under Building Direction, however I felt that Building Direction had components of it which are too dissimilar from Building Location to group them together.

Building Direction will decide in which cardinal direction the next building will appear (north/up, south/down, east/right, west/left). It will chose this by reading where the last building was generate.

All of these combined will generate a building and place it on the screen.

### Appearance:

This is separate from the others as it only ties into the data base to read the building level so that it can generate the sprite for the building whenever necessary. Otherwise it operates separately, providing a front end for the user, but not influencing the rest of the programme.

# Describing the Solution:

## Variables:

## Features:

### Backend:

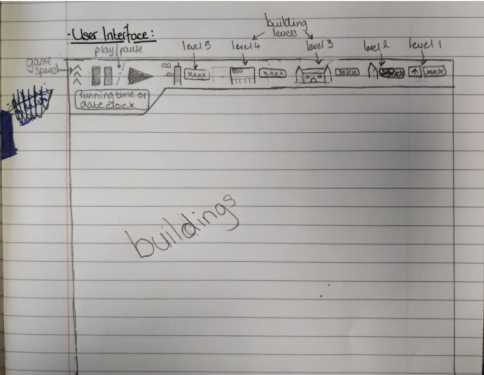
Random Generation:  
 The program should be able to randomly generate buildings in a North, East, South, West direction. As well as randomly generating an integer that would act as the tick time for the building to level up. The random direction would allow for the city to grow in a slightly more natural way, as buildings are often built in a concentric pattern around the city centre of the city/town, providing there are no natural obstacles. The random tick time integer would allow for buildings to grow at different rates, this would also cause the city to grow more naturally, as the growth of a building is often up to its owner, and everybody earns enough money to regenerate their building differently.

### **Interface**:

#### Play/Pause Button:

The program should have at least some user interface, so that the user can pause the simulation and look around it to see how each building has progressed. This would allow for them to build emotional attachments to some buildings, causing them to pay more attention to them and better follow their levelling up progress compared to the building’s neighbours, this would probably mean the also pay more attention to the simulation as a whole and there for learn about city progression better than they otherwise would. This would be used either with the mouse to click on buttons, or assign the space bar to the action, the spacebar would be used as this is a the standard button for pausing something across many applications, such as YouTube, and other games which involve a pause function, and also because the space bar is the largest button on a keyboard, so it is easy to press without much notification as there is little chance of missing it.

#### Labels:

 I may also use labels which appear when the mouse is left to hover over a building, these labels would simply state the level of the building, further eliminating confusion between different levelled buildings in the simulation, as somebody who is new to the simulation is probably not going to know of the differences between the buildings straight of the bat. The labels may also show some more general information about the simulation, such as the number of buildings of each level (perhaps with the total possible number of buildings for that level), though this may be put alongside the play pause button so that the user can see general statistics of the simulation as it is running. The user interface may look like the image to the right.

### Visuals:

#### Isometric Viewpoint:

The simulation would best be presented through an isometric view point, as this allows for different textures and, more importantly, sized sprites to represent the different building levels, this would make it far easier for the user to distinguish the different building levels and learn about the city progression levels. This would be more convenient than a bird’s eye view, as distinguishing building heights from above with no depth is very hard and would be detrimental to the simulations usability as a whole.

#### Sprites:

I would probably use sprites as the building models; they would have different heights to show the different building levels, as well as various textures for each level of building, as to introduce an element of realism to the simulation, as 100 identical looking buildings next to each other would look rather boring and would most likely disengage the user from participating in the simulation, hampering their ability to teach about it or learn from it.

## Limitations:

The presentation through an isometric view will be a limited factor because the time spent creating/finding proper textures will most likely be better spent creating other aspects of the programme, while learning how to code the isometric camera will also be take a lot of time.

Another limitation will be that the city isn’t “realistic”. Though I may be able to create a more organic road layout using L-Systems or Voronoi Diagrams, they will take large amounts of time and will also make the creation of proper buildings more difficult, as they will have to have their shapes fit within the borders of the roads, which take yet more time. This will be of the detrement to the stakeholders, as if they are using the programme to educate they would prefer the programme to be as accurate as possible; though this level of abstraction may be acceptable.

## Algorithms:

# **Iterative Development**

## Evidence:

## Prototypes:

## Approach to Testing:

## Testing the Code:

# Evaluation:

## Usability:

## Robustness:

## Success Criteria:

## Final Prototype:

## Maintenance and Development: