UNIVERSITAT
JAUME•I

UniPurge

Implementation of Landscape Generation Algorithms to Create or Modify Video Game Scenery in order to Simulate Parallel Universes

Ignacio Rodríguez Gozalbo

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

June 30, 2022

Supervised by: Miguel Chover Selles

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, Miguel Chover Selles, for his involvement in the project and his guidance when it comes to the procedural generation; as well as his perseverance when it comes to learning new Game Engines.

I would also like to thank my parents for always being close to me sometimes handing a helping hand in certain elements outside my comfort zone.

I also would like to thank Sergio Barrachina Mir and José Vte. Martí Avilés for their inspiring LaTeX template for writing the Final Degree Work report, which I have used as a starting point in writing this report.

One last aknowledgment to Alec Markarian, Benjamin Stanley and Brandon Fedie for creating the template upon which the Game Design Document was created.

Abstract

This is the memory of the final degree project in Video Game Design and Development bachelor's degree at the Jaume I University.
In this document there will be presented all aspects related to the development of "UniPurge", a Third Person Stealth Game set in a procedurally generated city in the contemporary age where the objective is to catch and remove a multiversal threat from the timeline.
The city is generated everytime a new playthrough takes place, randomizing all elements of the structure of the environment, simulating a parallel universe different to the last one. Since the main character is able to manipulate the universe fabric, some elements may vary while remaining quite familiar each time an alteration is generated.
In order to accomplish their objective, the player has access to various tools to alter and modify other character behaviours by tricking them.

Contents

Contents **v**

1 Introduction **1**
1.1 Work Motivation 1
1.2 Objectives 2
1.3 Environment and Initial State 2

2 Planning and resources evaluation **3**
2.1 Planning 3
2.2 Gantt 4
2.3 Resource Evaluation 4

3 Game Design Document **7**
3.1 Overview 7
3.2 What sets this project apart? 9
3.3 Core Gameplay Mechanics 10
3.4 Story, World and Gameplay 11
3.5 User Interface 15
3.6 Art 23
3.7 Sound and Music 25

4 Functional and technical specification **27**
4.1 System Design 27
4.2 Game Mechanics 31
4.3 Level creation 33

5 Results **39**
5.1 Work Development 39
5.2 Planning Control 40
5.3 Changes, Difficulties and Solutions 40
5.4 Milestones 41
5.5 Results 41

6 Conclusions and Future Work **45**

6.1 Conclusions 45
6.2 Future work 46

Bibliography **47**

A Source code **49**
A.1 Procedural generation Algorithms 49

Chapter 1

Introduction

Contents

1.1 Work Motivation **1**
1.2 Objectives **2**
1.3 Environment and Initial State **2**

This stealth game origin comes from the developer's mind after watching for way too long at the tiles on the ground. They imagined some ninjas jumping around really tall buildings. In the following years More elements were implemented into that ninjas and buildings, up until coming up with a stealth based game which can be seen here.

Since almost no one seemed to be interested in the themes presented, the developer decided to keep the idea under lock up until a situation presented itself; which did on this Work. Then the idea of ninjas jumping between buildings was intermingled with some ideas about procedural generation with some stealth mechanics sprinkled on top, which ended up becoming a quite interesting concept.

The stealth genre was chosen due not only to the ninjas, but also to emphasize exploration of the world and fast thinking over other elements like mechanical skill or focusing.

1.1 Work Motivation

The main motivation behind this work is to understand the inner workings of the procedural generation of environments; its difficulties and its possibilities, as well as the actual hardware limitations such systems may enforce and the optimizations required to make them work properly. Alongside such motivation, there is also an interest in expanding knowledge on game development by implementing the game on a previously not dominated game engine with very different systems and workflows to the previously known game engine.

1.2 Objectives

This proposal englobes the objectives cited before on work motivation:
The main objective is to learn various methods on how procedural generation works in games and how to implement those systems alongside with ensuring the generation of a big enough environment while keeping the time and resources needed to do such a thing as low as possible and keeping a realistic distribution of the world elements.

Other objectives include learning how the game engine Unreal Engine (UE) [4], specifically the 4th iteration works and the proper ways to implement elements in such environment.

Due to the engine choice, another objective includes the programming of the procedural generation systems with the c++ language.

Some other objectives includes the modelling of various 3D elements which can be combined in a modular fashion so that the resultant environment from the generation is believable enough; the planning and implementation of game mechanics which modify the environment and the implementation of sounds accordingly to the setting.

1.3 Environment and Initial State

UniPurge has been developed by a team of a single person by using only either custom resources made by them or by using free Unreal Games elements developed by Epic Games.

The environment on top of which the game is built upon is the game engine UE [4], being more precise, version 4.27.2, which is the second bugfix iteration of the version 4.27, released august 17, 2021.

Initially the project uses the “Starter Content” and “Mannequin” content libraries from the Unreal Engine default ones; all licensed to free use with Unreal Engine-based products.

Other tools used for the required work are Blender [3], a free Open source 3D computer graphics software toolset; Audacity [1], a free Open Source digital audio editor and recording application software and Krita [2]; a free open source raster graphics editor.

Chapter 2

Planning and resources evaluation

Contents

2.1 Planning **3**

2.2 Gantt **4**

2.3 Resource Evaluation **4**

2.1 Planning

All the required work in the project pretty much had a linear design apart from the artistic elements: modelling, texturing, making materials, animating and creating the sounds; having each one of them different requirements. But due to the project development being handled by only one developer, it was all developed with one parts after another; reducing the strength of parallel development due to the linear procedure and all elements being developed one after another in a linear fashion.

A timetable was estimated before starting the work on the project, separating the 300 hours dedicated to the project into various parts. The times indicated are not final, but not far from the truth; since there were not many deviations.

The planning method implemented in the project is a soft version of the Scrumm method; where each week or group of weeks were dedicated to a certain topic of collection of topics, like programming, modelling or making the NPCs.

Section	Task	Estimated Time	Total Estimated time
Narrative	Story	10 Hours	20 Hours
	World Structure (Lore)	10 Hours	
World Design	Planning the environment rules	8 Hours	20 Hours
	Progression	3 Hours	
	Designing modules	9 Hours	
Mechanics Design	Movement	6 Hours	30 Hours
	Stealth	10 Hours	
	Progression	7 Hours	
	Conflict Resolution	7 Hours	
Programming	Basic mechanics (Movement, progression and combat)	10 Hours	120 Hours
	Stealth mechanics	20 Hours	
	“dimensional shift”	30 Hours	
	Environment generation and modification	30 Hours	
	AI (Movement through procedural terrain and decision making)	30 Hours	
Art	Character Design	10 Hours	40 Hours
	Animation	10 Hours	
	Modules	10 Hours	
	Visual Effects	10 Hours	
	GUI	5 Hours	
	HUD	5 Hours	
Audio	Sound Effects	10 Hours	10 Hours
Academic	Memory	40 Hours	50 Hours

Table 2.1: Table with the original planning

2.2 Gantt

2.3 Resource Evaluation

Since this is a Final Degree Work, there was no economical compensation related with the type of work done on each moment; but assuming it existed a compensation; the price would be mostly separated in three parts according to the most important aspects in the project: designing, programming and art.

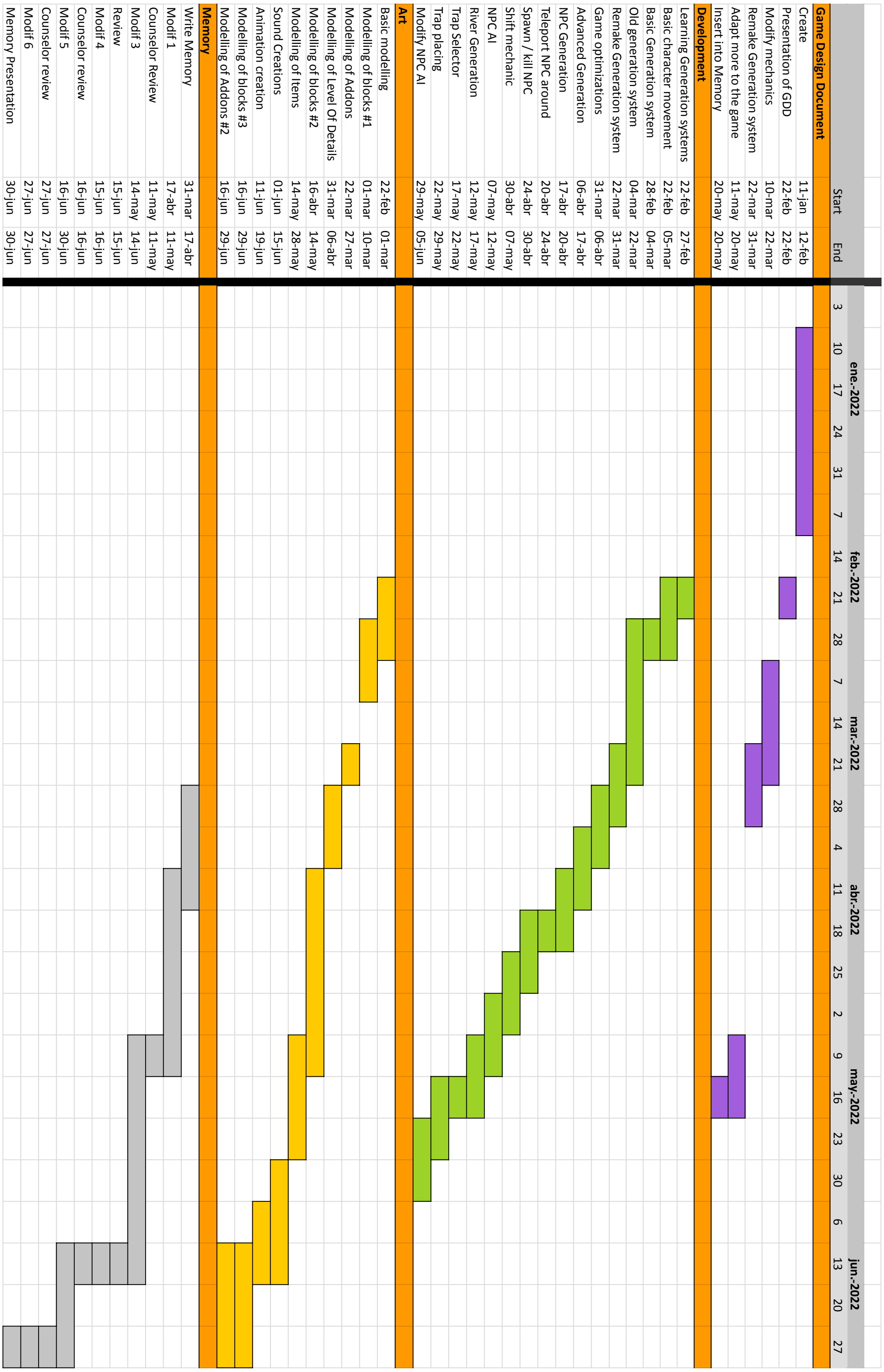
The average retribution for an entry level game designer job in Spain is of 27.000€ annually, which comes to be around 9,38€ an hour; which, relating back to the timetable and grouping World design and Mechanics Design as "game design" with a duration of 50 hours; would be worth 468.75€.

The average retribution for an entry level game programmer job in Spain is of 8,75€ an hour which, relating back to the timetable and the programming section with a duration of 120 hours, would be worth 1050€.

The average retribution for an entry level game artist job in Spain is of 30.910€ annually, which comes to be around 10,73€ an hour; which, relating back to the timetable and grouping Art and Audio as "artistic work" with a duration of 50 hours; would be worth 536,63€.

In the hardware department; the game was developed with a "gaming" laptop with a cost of 3000€ at the time of purchase and a mouse for the price of 60€.

Due to all software used in the project being either Open Source (Blender, Krita) or free (Unreal Engine) and the small amount of plugins and other elements used being also free, there were no cost associated with software at all.



	Start	End
Game Design Document		
Create	11-jan	12-feb
Presentation of GDD	22-feb	22-feb
Modify mechanics	10-mar	22-mar
Remake Generation system	22-mar	31-mar
Adapt more to the game	11-may	20-may
Insert into Memory	20-may	20-may
Development		
Learning Generation systems	22-feb	27-feb
Basic character movement	22-feb	05-mar
Basic Generation system	28-feb	04-mar
Old generation system	04-mar	22-mar
Remake Generation system	22-mar	31-mar
Game optimizations	31-mar	06-abr
Advanced Generation	06-abr	17-abr
NPC Generation	17-abr	20-abr
Teleport NPC around	20-abr	24-abr
Spawn / kill NPC	24-abr	30-abr
Shift mechanic	30-abr	07-may
NPC AI	07-may	12-may
River Generation	12-may	17-may
Trap Selector	17-may	22-may
Trap placing	22-may	29-may
Modify NPC AI	29-may	05-jun
Art		
Basic modelling	22-feb	01-mar
Modelling of blocks #1	01-mar	10-mar
Modelling of Addons	22-mar	27-mar
Modelling of Level Of Details	31-mar	06-abr
Modelling of blocks #2	16-abr	14-may
Modelling of Items	14-may	28-may
Sound Creations	01-jun	15-jun
Animation creation	11-jun	19-jun
Modelling of blocks #3	16-jun	29-jun
Modelling of Addons #2	16-jun	29-jun
Memory		
Write Memory	31-mar	17-abr
Modif 1	17-abr	11-may
Counselor Review	11-may	11-may
Modif 3	14-may	14-jun
Review	15-jun	15-jun
Modif 4	15-jun	16-jun
Counselor review	16-jun	16-jun
Modif 5	16-jun	30-jun
Counselor review	27-jun	27-jun
Modif 6	27-jun	27-jun
Memory Presentation	30-jun	30-jun

Figure 2.1: Gantt graphic

Chapter 3

Game Design Document

Contents

3.1 Overview 7
3.2 What sets this project apart? 9
3.3 Core Gameplay Mechanics 10
3.4 Story, World and Gameplay 11
3.5 User Interface 15
3.6 Art 23
3.7 Sound and Music 25

3.1 Overview

3.1.1 Theme / Setting / Genre

The game takes place in a contemporary like city filled to the brim with winding roads sprinkled with tall buildings with some parks in-between and some rivers separating the city into various parts; with many pedestrians filling the streets, differentiating it from the real world.

Due to the world not being fixed and it being generated every time a game is started, it is difficult to establish a concrete setting to place the game within other than the contemporary world influence in the various elements of the game.

The game belongs to the stealth game genre; a genre which revolves around maximizing the time the player stays hidden from all the characters view while completing their objectives; it usually entails mechanics allowing the player to stay hidden when confronted with other NPCs, as well as having some advanced AI centered around searching the character, following them and such. An example of the genre is Hitman 3 (figure 3.1).

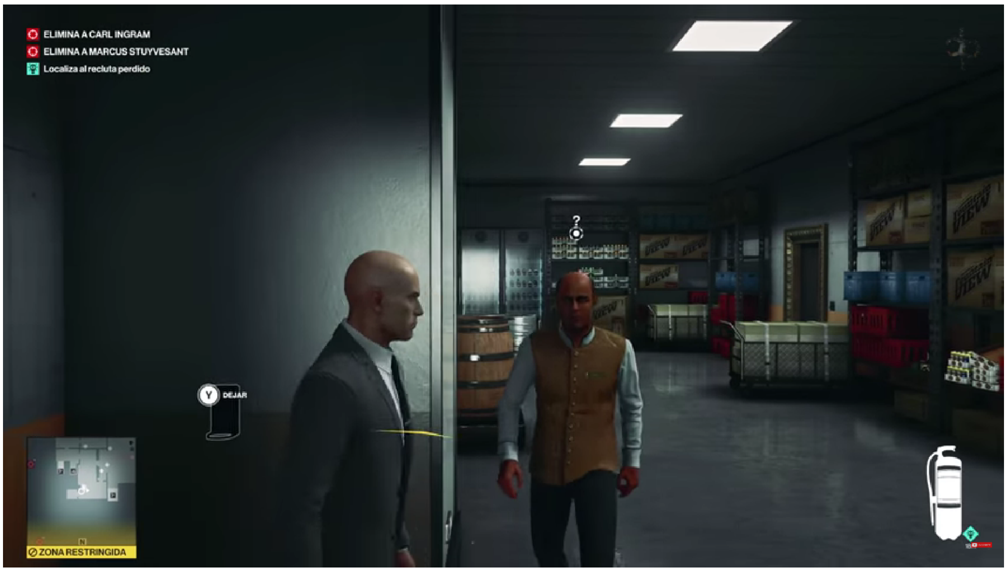


Figure 3.1: Hitman 3 Gameplay

The game also contains a certain amount of roguelite influence; mainly due to the randomness when generating the world and the sort of permadeath element in the game as well as the beginning from zero in each playthrough.

3.1.2 Targeted platforms

The game is targeted towards a Windows 10, Windows 11 PC x86_64 Installation with Xbox One / Series X/S and PS4/5 being also possible platforms to package the game for.

3.1.3 The Elevator Pitch

UniPurge is a game about ninjas who have to save the multiverse from a very dangerous man who threatens to destroy the fabric of reality.

3.1.4 Project Description

UniPurge is a stealth based game where the objective is to reach an enemy and defeat them. It is accomplished by traversing a procedurally generated city created with modular elements which combine in order to create a semi-realistic environment filled with various types of NPCs doing their daily routine.

The main character has all the basic controls of a 3D third person game: walking, running, jumping, crouching and camera rotation; with some additions like wall-climbing, wall running or wall jumping. All this extra movement options allow the character to avoid any confrontation, since they are really weak.

Alongside those movement mechanics, the player also has some stealth mechanics which can be utilized, like distracting characters by placing either sound or visual distractions in the environment.

When it comes to confrontations and problem solving, the player has another trick up its sleeve. A controller that allows them to jump into a sub dimension which, in turn, allows the character to place traps in the environment that will activate once they return to the original universe; but this controller needs to be recharged between uses, forcing the player to make choices about when and where to place those traps. Alongside moving the character from dimension to another, the shift has a secondary effect in that it changes things from the real environment. There are no living beings originals from the sub dimensions, making it so that only the player exist in the area; but other characters can be seen is their current position, allowing the player to more easily choose where to place the traps.

About the NPCs; they all have some procedural movement, like going from a house to another or walking in the park.

Most NPCs will stop their routines when they hear the player moving close to them and will start chasing them if they cross their view. If the player manages to avoid them, either by running faster or by hiding in a difficult to reach area, they will lose interest and return to their traditional routines. If instead of getting away, the player gets caught; this will fail the ninja's mission, making it so that the universe has to be destroyed (a gameover is triggered).

The objective is a special NPC which will always run away from the player (Runs slower than the main character). The objective is to "kill" them.

3.1.5 Target Audience

The main target audience for this game is teenagers and young adults, even though the game is playable for anyone capable of comprehending basic mechanics.

3.2 What sets this project apart?

This project has various features that set it apart from other games of the genre:

- Every game takes place in a different, procedurally generated, city than the previous one, varying many elements of the environment; from roads to the distribution of windows.

- This city can be further modified when interacting with the multiversal elements of the game; the "Dimensional Shift", which changes certain elements of the world every time its executed.

- The gameplay is based on stealth, but; instead of the traditional approach to stealth, which usually entails hiding behind certain elements, it is based upon deceiving opponents to move inadvertently.

- The game takes advantage of its procedural generation to offer infinite replayability to the player; since no two worlds are the same and the objective can be attained in various ways.

3.3 Core Gameplay Mechanics

3.3.1 Movement

Details: Semi-advanced movement mechanics; 2D motion while on the floor alongside jumping, border climbing, crouching and swimming; wall-related movement; like wall running, wall climbing and wall jumping and 3D motion while falling.

How it works: The movement type is decided according to what items are around the player character. If there is something solid under the character, it is walking/running movement; if there is none, but there is a solid item on one side, the movement would be wall-related; if a water block is detected, it will start the swimming motion and if no solid objects are detected the player will start falling. Anytime the player isn't touching solid ground or holding to a wall is affected by gravity. All movement is made with the same controls, being keyboard keys or controller buttons / axis with the typical controls design; WASD or left stick for 2D movement, Spacebar or the lower button on controller to jump and control or the right button on controller to crouch.

3.3.2 Stealth

Details: All NPCs in the world tend to be somewhat aware of their surroundings and may try to catch the character if they enter their field of view. If the player can avoid their glance for some time, they will forget them and return to their activities. They can also be fooled by placed traps, which will peak the NPCs curiosity as well as distracting them.

How it works: All NPCs in the world have two sensors, a cone in their face which acts like their sight and a sphere centered on them which acts like their hearing. Once a sensor detects any abnormal element, a check is done. If the detection was done by the "sound" sensor; the character rotates to face the sound origin; if it was made by the sight sensor, the NPC starts chasing the detected being.

3.3.3 Dimension Shift

Details: The player has a “Dimension controller” which allows for creating and traveling to a sub-universe, allowing the character to overcome problems in their way.

This “Dimension shift” has certain effects: it firstly generates a minor universe which is a variation of the original universe with some modifications on the area; then, once the character reactivates it again, it transports them to the place where they activated the shift in the first place, but, due to multiversal mechanics, some other elements from the minor universe are transported to the original universe. While inside the sub-universe the character is able to place traps to distract or slow the NPCs. The controller has a battery which is required to "shift" and place traps and which needs to be recharged in order to do so. The battery charges passively while in the real world, punishing constant use of the mechanic by reducing the available amount of energy the player has available.

How it works: The player has a button dedicated to “Dimension shifting” which activates the controller. Then a 5×5 area of blocks around the character are “taken” and regenerated with some variations. The player is then free to roam around the area and place traps anywhere they feel like it. Afterwards, the player can reactivate the controller to go back to the original universe. After activating the controller, some of the modifications from the sub-universe are chosen and applied to all the blocks in the world and all the placed traps start working again.

Traps

Details: As an interconected mechanic between Dimension Shift and Stealth; traps are a system which can be interacted with while Dimension Shifting and are used to distract NPCs. Each trap has a different effect on the NPCs that come in contact with them; depending on the trap it only distracts them by making them walk to the location of the trap or it blocks the character movements, alongside other effects.

How it works: Once entering Dimension Shift the placement of traps begins. A trap is previewed where it would be placed upon pressing the "place trap" button; this preview takes into consideration both gravity and the trap’s hitbox, allowing the player to place it even without clear vision of the ground. Each trap has a cost of Energy to be placed, and they also have varying lifespans. Once the player ends the Dimension Shift; all traps activate at once and do their functions along all their lifespan. Once their lifespan is expired; the trap is destroyed.

3.4 Story, World and Gameplay

3.4.1 Story

There is a group of entities whose job is to ensure the multiverse’s stability and it’s longevity by any means necessary. Those entities have a team of enforcers who are entrusted with the difficult missions of fixing those problems that threaten the stability or to completely terminate the universe if the damage is irreversible. Inside this group of enforcers there is the ninja core; diverse ninjas whose job is to fix punctual problems in a universe, like a threatening life form or a machine, without causing much of a

commotion.
The game is all about the adventures this group endures in order to terminate a man who is a danger to the multiverse due to the latent powers it has; so it has to be assassinated before those powers can develop and destroy everything.
In order to help the ninjas in their mission; they have limited access to some multiversal controller, allowing them to create bubble dimensions in order to help them achieve their goal. If they fail in their mission, the whole universe is destroyed; since the damage done by the ninja will have been irreparable.
In the case the ninja is successful, they are send to a different universe in order to repeat their objective; such universe being similar and different from the previous one.

3.4.2 World Design

The World Itself

The game takes place in one of the various parallel universes which are endangered by the objective's existence.
All gameplay is encased inside a very big city which is procedurally generated, with various basic elements, like roads, houses, rivers, parks, etc. which connect seamlessly thanks to the modular system of the elements and their decorations.

In this city there are people living, going about their daily routines, like going to work or shopping; mainly shown by showing them walking from building to building.

World Generation

Modular Items The world generation is managed with certain modular items with square form which are modified after generating in the land, making it so that they can create more complex scenery.

There are different types of modules, each related to different elements.

- **Roads:** The roads modules are comprised of blocks which have a pavement and sidewalks alongside the borders. In them other additional items can be built; like street lamps, dumpsters or litter. They connect only with themselves or with bridges; leaving the sidewalks to connect to any other block

- **Parks:** The park module is very basic; since it only contains a grass floor; but its additions are very varied: trees, tables, fountains and other elements serve to give the module its livelier state.

- **Houses:** The house module is quite complex. The base is formed with four pillars and a ceiling, which encompasses the common elements in any building it has to be built. The addons of the module are quite varied. It has floors and interior elements like tables or ladders, walls of various types: simple walls, walls with single windows, double, balconies and such and doors with their own variations of the element. The addons are placed according to the surroundings of the building, allowing for buildings in custom shapes thanks to the "fusion" of various house modules by just removing the wall in-between.

- **Rivers:** The river module is really similar to the road module, but using water instead of road; and having two variations with bridges in them which allows for the road to cross through the river. The module is by far the most detailed one from the get-go; but it still has some addons in itself; mainly drainage, but also some boats and such.

Generation system

The world generation is created by layering various systems of procedural generation on top of each other.
Firstly the rivers are generated in the environment; then, utilizing Wave Function Collapse, the roads are generated in a connected way and serve to separate the world in various areas to be filled with other elements in posterior steps. Afterwards a "globular" system searches all adjacent tiles to roads and generates groups with their own adjacent tiles to create either buildings or parks. A decorator system that navigates all tiles in the world and generates random elements in them according to their adjacent tiles so that buildings feel completed. An NPC data generator that decides whether a tile contains an NPC and their destinations in their routines.

3.4.3 Gameplay

Once starting a game, the world gets created according to the factors that were previously specified in the Options menu and the selected seed (a seed is chosen at random if left empty); if no factors were specified in the menu, some default ones are used instead.

Once the generation is finished, a ninja appears in the game world, being the main character and giving the player complete control over its movement. Afterwards, the objective is told: to reach a certain person and “remove” it from the universe.

The player has many options at this moment, all allowing for multiple ways to challenge problems.

There is the typical ground movement, where the ninja will move towards the indicated position, not after turning to face it. It is possible to run holding the corresponding key and to jump with its own key.

The camera usually orbits the player and is controlled by the mouse or right stick.

Once the ninja is no longer grounded and touches a wall, it will start climbing and the movement controls will change. The jump will “propulse” the character opposite to the wall with some extra vertical velocity (Normal vector + some up force). In this position moving forward or backwards moves the character up or down along the wall, while the other positions make the player move alongside the wall slowly, allowing them to turn corner; if the sprint key is held while moving, a wall run is initiated, making the movement much faster but disabling corner turning.

If the ninja reaches the top of a certain wall and there is no ceiling, they will be able to vault over it.

If the ninja is no longer touching a wall or a floor, it will start falling, where they will be able to influence some of their momentum in the air. When it falls into the floor, a falling animation plays. If it instead comes into contact with a wall, they'll start scraping it until they come to a halt.

Finally in liquid environments, the ninja starts swimming alongside the surface.

The AI in the game is divided into two groups: The NPCs and the Objective. Both having sight and hearing capabilities.

The first group consists of almost all the NPCs and they tend to follow a routine by walking in the streets between some points. Once they cross paths with another character, they will check it and act accordingly. If they think is the player, they will run towards them and try to catch them.

The Objective has a more complicated Schedule, traversing great distances. Once it detects the player, they will become startled will run far from the direction the player was for some time. Once they lose sight of the player,they will return to their routine.

The most important tool in the ninja's arsenal is the dimensional controller, which allows them to jump into a sub-universe where there are no sentient beings and certain elements are different. After returning, some of those differences may be brought to the main universe.

The game can end in two states:

- **Success:** If the target is reached, in which the ninja will kill the objective and the mission will be considered accomplished, teleporting the ninja away from there.
- **Failure:** If the ninja dies or is caught in the mission a universal deletion will be triggered by destroying all elements in it and the living beings inside them.

3.4.4 Artificial Intelligence (AI)

PathFinding

UniPurge utilizes the A* algorithm with pathfinding with a procedural Navigation Mesh generated at runtime, allowing for modification inside itself.

Non Playable Characters (NPCs)

- **Civilian:** The civilian AI has certain modules defined as waypoints, one of them defined as their home, and will walk a route between them. Whenever their sensors detect another character, they make a check to detect if it is a "known" entity (another Civilian); if it is not, they will either turn towards the position of the element, if the sensor was sound, or start chasing down the element if it was detected by their sight. If the player leaves their sight or they discover the detected element is not the player, they will look around for a moment and then either continue chasing the player, if found, or return to their routines.

- **Objective:** The objective has a very similar AI to the civilian but it is much more scared of the player; running away from them at the slightest hint of their location.

3.5 User Interface

3.5.1 Flowchart

The game contains various menus both in-game and out of it, those being as follows:

- **Main Menu:** The starting menu for the game, containing the title, some buttons to start a new game, go to options or exit the game and a text input to input a desired seed or leave it empty to select a random one.
- **Options Menu:** The menu for adjusting the options. It has some sliders to control volume and to control the render distance of both blocks and the amount of NPCs observable at a time; as well as the absolute world size. And a button to go back to the Main Menu.
- **Game scene:** The scene refers to where all the game takes place. A small menu with buttons can be opened by pressing ESC and allows for easy access to the options menu, go back to the game or exit to the Main Menu and also contains an image of the targeted character.
- **Credits:** A single screen detailing the participants of the project and all the credit to the creators of certain tools.

Since the Main Menu and the Pause menu share some things but not everything, a different flowchart is done for each one. (Figure 3.2) (Figure3.3)

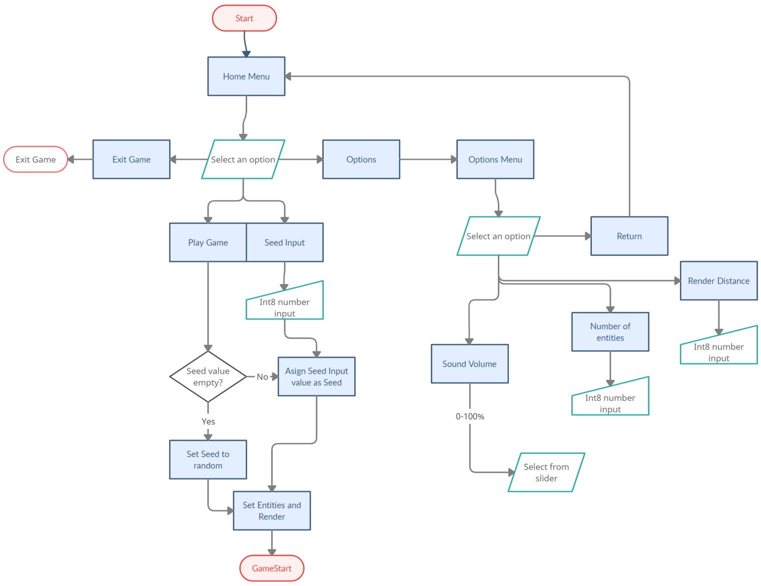


Figure 3.2: Flowchart image of the Home Menu (made with Creately)

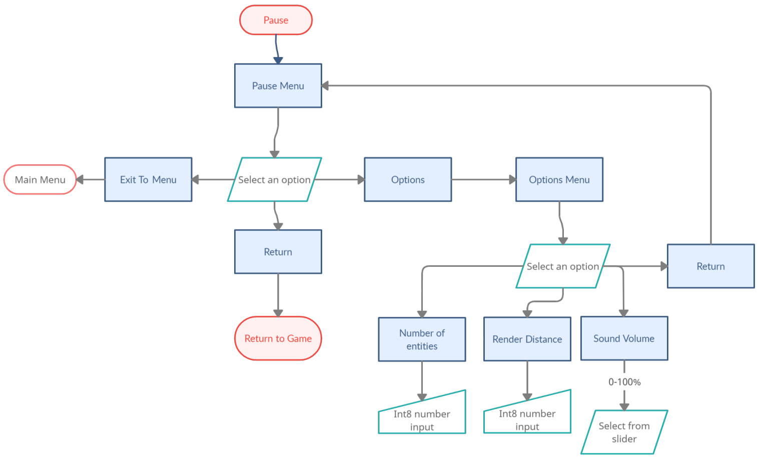


Figure 3.3: Flowchart image of the In-Game Menu (made with Creately)

3.5.2 Functional Requirements

The GUI is not very deep; and as such, only requires basic requirements

Main Menu

The main menu functionality boils down to its buttons and what each one of them does on release (Table 3.1), (Table 3.2), (Table 3.3) and the Seed input area (Table 3.4).

Input:	Button: On release
Output:	The game changes world to the generated world
The game switches worlds from the "Menu World" to the "Generated World", where the gameplay takes place. In this switch certain elements are stored so that they can be retrieved in the other world. Those being the seed to use and (in case no value was specified in options), the default values for World Size, NPC render and Module Render.	

Table 3.1: Functional requirement «Start The game»

Input:	Button: On release
Output:	The game menu from the current one to the Options one
The game switches the open menu to the Options menu. This interaction is further modified depending on the menu from which this change originates. In case of the Main Menu, all elements from the Options menu appear: Sound and Music volume; World size and Renders for modules and NPCs. If the menu is instead open from the Pause menu, the option for changing World size is disabled, since it is already generated.	

Table 3.2: Functional requirement «Open the options menu»

Input:	Button: On release
Output:	The game closes
The game closes once one "Exit Game" button is pressed.	

Table 3.3: Functional requirement «Closes the game»

Options Menu

The options menu is the other menu with some complexity; since it sets up almost all the elements that are then fed into the Game Manager to create the experience. (Table

Input:	Input string (keyboard)
Output:	A seed is selected
The player can write in the seed space, allowing them to control which seed is generated; helping with the sharing of worlds and to control the generation. It uses a custom parser which has encoded 64 characters, each with a different value and uses any other character as a default value.	

Table 3.4: Functional requirement «Seed input»

3.5), (Table 3.6), (Table 3.7), (Table 3.8), (Table 3.9) and the button that stores all those variables (Table 3.10).

Input:	Slider
Output:	Changes the volume of the sounds
The sounds volume changes according to the value indicated in the slider.	

Table 3.5: Functional requirement «Sound Slider»

Input:	Slider
Output:	Changes the volume of the music
The volume of the music changes according to the value indicated in the slider.	

Table 3.6: Functional requirement «Music Slider»

Input:	Spin Box (value)
Output:	Changes the size of the world
The value of the Spin Box for the World Size determines the size of the world that will be generated once the "Start The Game" button is pressed (Table 3.1). It has a minimum size of 20x20 modules and a maximum size of 200x200 modules.	

Table 3.7: Functional requirement «Determines the world size»

In-Game Menu

The In-Game menu, or pause menu, functions very similarly to the Main Menu; but changing the "Exit Game" button for a "Back to menu" button which changes the world from "Generated World" into "Menu World".

Input:	Spin Box (value)
Output:	Changes the render radius of modules in the game
The value of the Spin Box for the Render Radius determines the radius of modules that can be seen at a time, all around the player position. This value can be changed even in-game, allowing for modification if computing power is compromised. It has a minimum size of 10000 units and a maximum size of 1000000 units.	

Table 3.8: Functional requirement «Render Radius for Modules»

Input:	Spin Box (value)
Output:	Changes the render radius of NPCs in the game
The value of the Spin Box for the Render Radius for NPCs determines the radius of non playable characters that can be seen at a time, all around the player position. This value can be changed even in-game, allowing for modification if computing power is compromised. It has a minimum size of 2 units and a maximum size of 20 units.	

Table 3.9: Functional requirement «Render Radius for NPCs»

Input:	Button: On release
Output:	Goes back to the precious menu
The button is used for two functions. On one hand, it changes the current menu back to the previous one, either the Main Menu or the In-Game Menu; but, on the other hand, it stores all the data specified inside the menu into a file so that it can be loaded into the Game Manager and used properly afterwards.	

Table 3.10: Functional requirement «Exit Options Menu»

3.5.3 Non-functional Requirements

In order to make the Graphic User Interface as approchable as possible, it has to remain simplistic and concise, but represent in a proper way all the required elements and retain a high amount of personality.

The menus need to have quite big buttons and sliders, allowing players to easily press them without many problems.

The HUD needs to show the current remaining charge of the dimensional controller and the costs of each trap before placing them.

3.5.4 Mockups

Main menu Mockup (Figure 3.4)

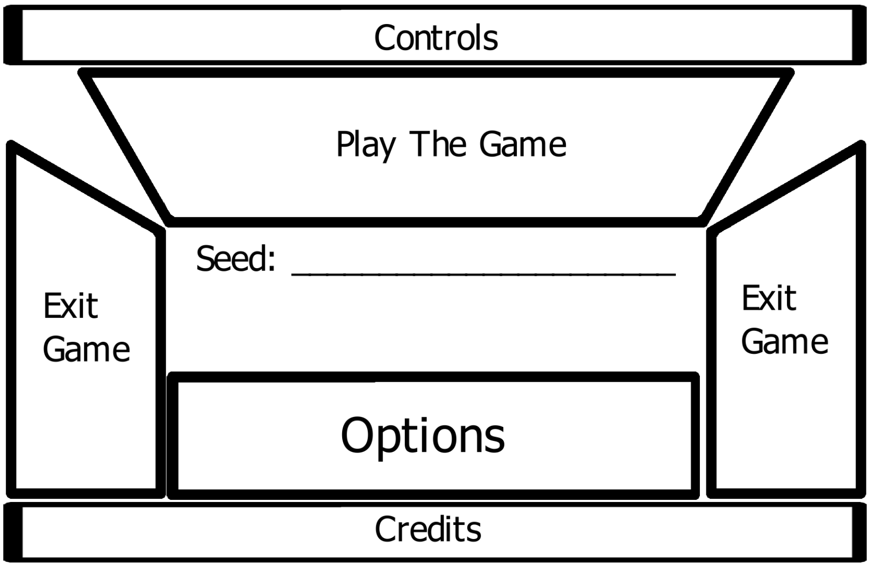


Figure 3.4: Mockup of the main menu.

In-Game Mockup (Figure 3.5)

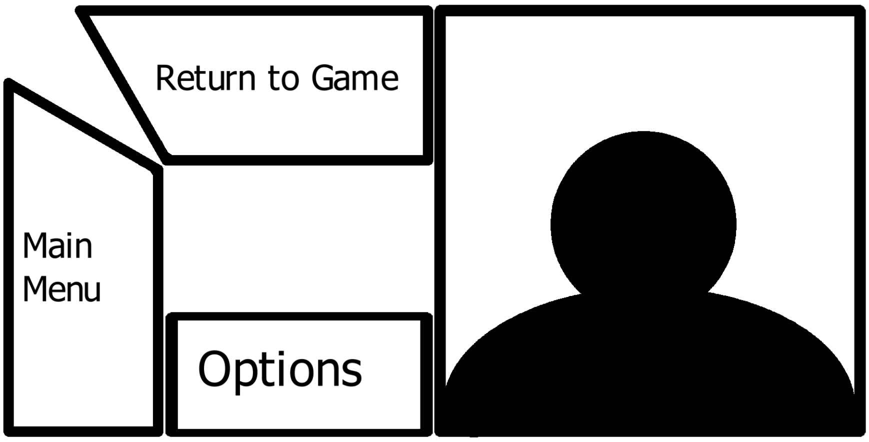


Figure 3.5: Mockup of the In-Game Menu

Options Menu Mockup (Figure 3.6)

HUD Mockup (Figure 3.7) The In-Game HUD is almost empty; and that is on

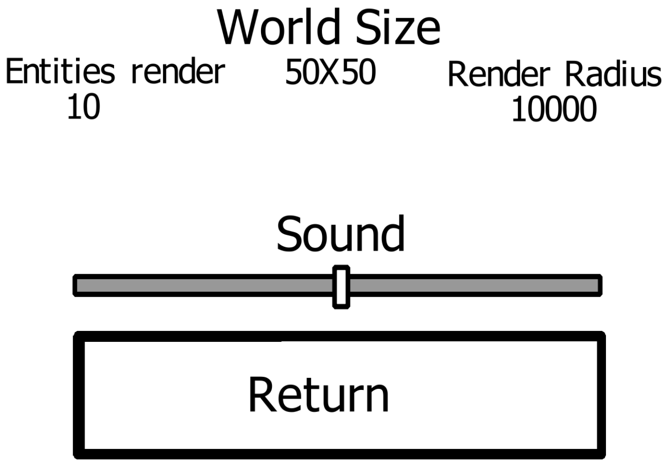


Figure 3.6: Mockup of the Options menu

Figure 3.7: Mockup of the HUD

purpose; since the character is defeated by a single touch and NPCs are always dangerous. So then the only thing that needs tracking is the remaining charge in the Dimensional Controller; and that's the use of the bar. It'll fill or empty to show the remaining battery. Whenever a trap is going to be placed, a region of the current charge is highlighted, showing the cost of the trap.

3.5.5 Default Controls

Gameplay

- **Movement:** WASD || Left Stick
- **Move the camera:** Mouse Delta || Right Stick
- **Jumping:** Space || Bottom button of a controller
- **Sprint:** Shift || Left Button || Right Button
- **Crouch:** Control || Right button of a controller
- **Start Dimensional Shift:** Left Click ||Right Trigger
- **End Dimensional Shift:** Right Click || Left Trigger
- **Switch Placeable Trap:** Q / E || Left / Right on the D-Pad
- **Place Trap:** F || Left Button of a controller
- **Menu:** ESC || Start

Menu

- **Navigation:** WASD ||Left Stick || Control Pad Can also be made by placing the mouse on top of an option
- **Selection:** Left Click || Bottom button of a controller
- **Back:** ESC || Right button of a controller

This controls can be represented in the image 3.8 as follows: Green indicates the controls used in movement. Light green represents Mouse Delta, used for the camera movement Yellow represent the keys dedicated to the placement of traps. Blue represent the buttons dedicated to the Dimensional Shift. Finally red represent the button used to bring up the menu

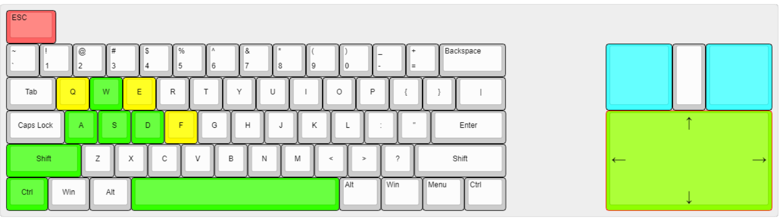

ESC
~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { } |
Caps Lock A S D F G H J K L : " Enter
Shift Z X C V B N M < > ? Shift
Ctrl Win Alt Alt Win Menu Ctrl
↑
←
→
↓

Figure 3.8: Layout of the keys used in Mouse and Keyboard inputs Generated on http://www.keyboard-layout-editor.com/

3.6 Art

3.6.1 Overall goals

The overall goals of the art direction of the game is to portray a contemporaneous city using 3D models with a low-poly aesthetic by using semi-detailed models and materials.

Due to the increased importance of the 3D modelling over the texturing, 3D models are heavily detailed, but always keeping the details not too small or unnoticeable.

The materials used are more of an interpretation of the actual material rather than realistic depiction; this helps to give the game an enduring art style, making it so that the graphics do not feel dragged down by less powerful machinery.

A lighting system with hard edges is chosen over one with soft edges so that the low-poly art style is maintained while keeping the modelling as pure as it is desired.

The models are designed modularly so that they can fit together without creating any holes in the final city. This modularity approach allows for the implementation of extra variations of almost all elements of the game; allowing to expand the variation of the world pretty easily.

3.6.2 modelling

Most of the models used in the game were hand made using the goals cited above. Those models can be separated in blocks, addons and decorations.

Blocks

The blocks collect a collection of models used to generate the base of the world. Each one of them has either two or three level of details, each one of them doing certain actions to decrease the cost of the render. (fig: 3.9) Reduced detail models (fig: 3.10a) (fig: 3.10b)

Addons

Addons are models that are necessary to complete a block but can be interchanged among themselves. (fig: 3.11a) (fig: 3.11b)

Decorations

Decorations are all the other models that are not necessary for the structure of the game but give an atmosphere, as well as including other models like traps. (Figure 3.12a) (Figure 3.12b)

3.6.3 Animation

The game contains two custom animations for the main character: a climbing animation and a swimming one, both animations run at 30 frames per second and last two seconds on normal speed. They were hand created on Unreal's native animation tool by

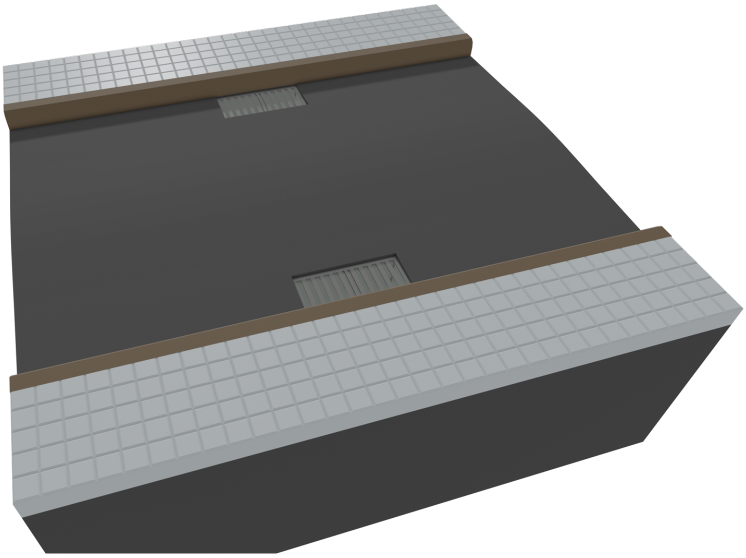
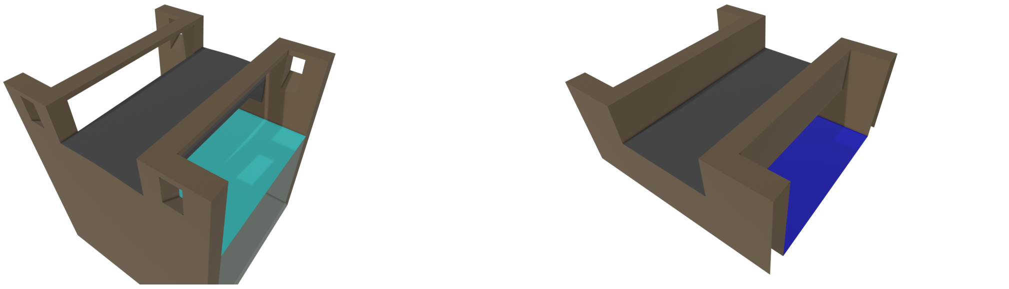


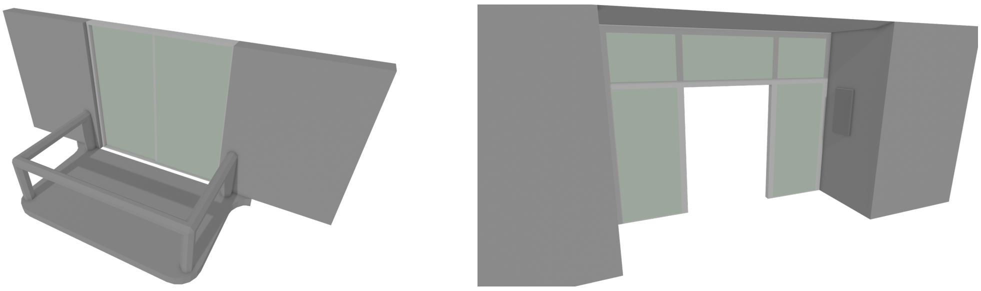
Figure 3.9: Most detailed model of a straight road



(a) LOD 1 of a straight river crossing

(b) LOD 3 of a straight river crossing

Figure 3.10: Level Of Detail Examples



(a) Most detailed model of a balcony

(b) Most detailed model of a door

Figure 3.11: Addons Examples

hand using the keyframe system since the skeleton that the basic Unreal mannequin is built upon is completely different that the usual skeleton used in other products.

(a) Most detailed model of a speaker

(b) Most detailed model of a tree

Figure 3.12: Decorations Examples

3.7 Sound and Music

3.7.1 Overall goals

The overall goals of the sound design is to enrich the experience the player have while playing the game.

3.7.2 Sound FX

All sound effects attempt to represent the actions done by each character in the game. The player has sound-effects for stepping, jumping and swimming, while the NPCs only have sounds of walking and jumping. There is also sounds for the traps. All sound effects are affected by 3D distance; turning quieter the further the player gets from them.

All the sounds used are not made in house and are instead obtained from files under the Creative Commons License. Some of them are under the CC 0 license (Citar zero), but others fall under other Creative Commons licensing: The sound used for the swimming animation[9] or the slips of music used on the speaker [8]

Chapter 4

Functional and technical specification

Contents

4.1 System Design 27
4.2 Game Mechanics 31
4.3 Level creation 33

4.1 System Design

4.1.1 Functional Requirements

The proposed system of interaction with the world has some environment dependant actions that change depending on the relation the actor has with the world that surrounds them, like moving (table 4.1), jumping (table 4.2), sprinting (table 4.3) or crouching (table 4.4).

The camera is controlled by the player and orbits the player (table 4.5).

The remaining controls are all related to the Dimensional Shift (table 4.6) (table 4.7), the selection of towers (table 4.8) and the placement of them (table 4.9).

4.1.2 Non-functional Requirements

The main requirements of the gameplay design is to open doors to the player, allowing them to go to most places and to explore, since the main restrictions in the game come from the level design and the amount of NPCs walking all over the place. In order to

<table>
<tr><td>Input:</td><td>WASD | Left Stick of a controller</td></tr>
<tr><td>Output:</td><td>Character moves along a determined plane</td></tr>
<tr><td colspan="2">When the player presses any button from the group (W,A,S,D) or moves the left stick of a controller, the character moves with the appropriate vector of movement, which is modified by a conversion matrix depending on the surface the movement takes place, ensuring a movement parallel to the surface of the collision:
The Forward / Backward movement (W,S) or Y-Axis moves the character in the forward vector with various magnitudes.
The Right / Left Movement (A,D) or the X-axis moves the character in the right vector with various magnitudes; this movement is also limited when moving while climbing a wall, simulating the character moving mainly by short movements of their hands. But this allows for corner turning alongside the walls, either outer corners and inner ones.</td></tr>
</table>

Table 4.1: Functional requirement «Movement on 2D plane»

<table>
<tr><td>Input:</td><td>Space Bar | "South" button of a controller</td></tr>
<tr><td>Output:</td><td>The character jumps</td></tr>
<tr><td colspan="2">When the player presses the jump button, the character will attempt to jump. If they are on top of the ground, the jump will result in a mostly vertical force applied to the body, but if the character is climbing a wall; it instead applies a force opposite to the wall (normal) with a small impulse upwards. In the case the character is swimming; the jump fails.</td></tr>
</table>

Table 4.2: Functional requirement «Jump»

<table>
<tr><td>Input:</td><td>Hold Shift | shoulder buttons of a controller</td></tr>
<tr><td>Output:</td><td>The character sprint</td></tr>
<tr><td colspan="2">While the player holds the sprinting button, the movement system magnitudes are modified to simulate the sprinting behaviour. This modification is mostly an static scalar multiplier; but when the character is climbing it modifies the mechanics of the Right / Left movement: Making it much faster but at the cost of losing outside corner turning, making the character leave the wall and drop to the ground instead.
Additionally, sprinting increases the sound generated by the character.</td></tr>
</table>

Table 4.3: Functional requirement «Sprinting»

<table>
<tr><td>Input:</td><td>Hold Control | "East" button of a controller</td></tr>
<tr><td>Output:</td><td>The character crouches</td></tr>
<tr><td colspan="2">Kinda similar to the sprinting input; crouching modifies the speed of the movement of the character while holding its button, decreasing the character speed. It also modifies the sound generated by the character, almost muting the sounds it makes.</td></tr>
</table>

Table 4.4: Functional requirement «Crouching»

<table>
<tr><td>Input:</td><td>Mouse Delta | Right Stick of a controller</td></tr>
<tr><td>Output:</td><td>The camera orbits around the character in the direction indicated</td></tr>
<tr><td colspan="2">The game's camera is controlled by the player so that they can look in any direction they desire.
When realizing any input, the camera moves alongside an orbital path around the player following the desired direction.
In the case there is an obstacle between the camera and the player; the camera moves closer to them so that minimal visual impediment is generated.</td></tr>
</table>

Table 4.5: Functional requirement «Camera controls»

<table>
<tr><td>Input:</td><td>Left click of the mouse | Left trigger of a controller AND Energy > required Energy</td></tr>
<tr><td>Output:</td><td>Enter Dimensional Shift</td></tr>
<tr><td colspan="2">This input initiates the dimensional shift mechanic if the player has enough energy:
It stops all NPCs in place and disables their collision with the player.
It limits the area the player can move around to a square area of 5 squares of side.
It changes some decorations of the world to similar ones.
Consumes a part of the energy of the Dimensional controller
Enables the selection and placement of traps.</td></tr>
</table>

Table 4.6: Functional requirement «Enter Dimensional Shift»

<table>
<tr><td>Input:</td><td>Right click of the mouse | Right trigger of a controller</td></tr>
<tr><td>Output:</td><td>Exit Dimensional Shift</td></tr>
<tr><td colspan="2">This input finalizes the dimensional shift mechanic:
It restores the behaviours of all NPCs and re enables their collision with the player.
It removes the area of movement limitation.
It teleports the player to the location they activates the Dimensional Shift mechanic.
All the placed traps get activated
Disables the selection and placement of traps.</td></tr>
</table>

Table 4.7: Functional requirement «Exit Dimensional Shift»

<table>
<tr><td>Input:</td><td>Q/E | Left / Right in the pad of a controller</td></tr>
<tr><td>Output:</td><td>Changes between all the types of elements the character can build</td></tr>
<tr><td colspan="2">This input changes the selected tower to build between all of them in a linear fashion. With a button (Q | Left) to change to the one "before" the selected at the moment and another (E | Right) to change to the one "after" the current selected.</td></tr>
</table>

Table 4.8: Functional requirement «Change Tower Selection»

<table>
<tr><td>Input:</td><td>F | "West" button of a controller AND Energy > required Energy for the tower</td></tr>
<tr><td>Output:</td><td>Place the selected tower in the world</td></tr>
<tr><td colspan="2">If the player has enough energy in the controller, it places the selected trap in the world where the camera is pointing towards (A basic preview is shown before placing).</td></tr>
</table>

Table 4.9: Functional requirement «Place selected tower»

accomplish such requirements; the gameplay is built around freedom of movement and a bit of flow to make the movement a bit more enjoyable. The main character can surpass almost any obstacle by themselves without any problem; but the important aspect is if they are seen doing it.

4.2 Game Mechanics

4.2.1 Movement

The character movement mechanics are dependant on their surroundings, changing on their functionality but keeping the simple controls for all of them. It can be easily categorized in two types dependant on the state of the character in relation to walls.

Ground Movement

When the player has their feet on the ground, the movement is based upon the basic 3D third person movement, with both possibilities for mouse and keyboard and controller; each one with their own assignations.

All movement is done parallel to the ground with the WASD keys or by using the left stick in a controller. The basic movement speed is higher than all the movement speeds of the NPCs. That movement speed can be modified by using the sprint button, Shift or Right button or R1, depending on the controller used; or by crouching with the control key, or right button in a controller; each one having different effects in the surrounding NPCs; like sprinting being louder and notifying the characters from a longer distance or crouching reducing the sounds made.

The character can also jump by pressing the Space key or the bottom button in a controller; the jump is controllable and always reaches the same height, independent on the time pressing the button. Once the maximum height is reached, the character starts falling, this movement state allows the character to start the second type of movement: Wall related movement.

Wall Movement

Whenever the main character collides with a wall while falling; they change their movement to wall-related movement. The movement controls become dependant on the wall the character is latched onto; changing the movement to be on a plane parallel to the wall. Some basic movement is also modified: while vertical movement (W,S or vertical axis of the left stick on a controller) stays very similar to it's grounded version; horizontal movement is highly reduced, mainly to resemble a more realistic movement where a person needs to make sure their grip is secure before moving.

This movement also allows the character to move around corners while climbing. The sprint key is also modified depending on the movement direction: it only increases speed when climbing up and down, but it changes completely the horizontal movement; it makes horizontal much faster than the base horizontal wall movement, but it doesn't allow the protagonist to rotate around walls when finding an outward corner, instead leaving the wall with their accumulated speed. Jumping is also modified when jumping of a wall: It propulses the player, not only away from the wall, but also a bit upwards, generating quite a bit of momentum.

The player can leave this movement state by leaving the wall by jumping from it, which will leave them falling with quite a hefty amount of momentum; reaching the top of the wall or climbing down until it touches the ground, at which point they'll transition to ground movement.

4.2.2 Stealth

Since UniPurge is a stealth game, it has quite a bit of stealth mechanics at the main character's disposal:

The game has a sound system; where each movement the protagonist does makes a sound that can be heard by NPCs. This sound will make any NPC turn to face the origin of the sound, trying to locate its creator. The sound can be modified by sprinting or crouching; sprinting increases the strength of such sound, allowing for more NPCs to hear it, while crouching decreases the sound strength and, as such, decreases the number of NPCs that may hear the character.

If an NPC finds the character inside their point of view, they start chasing them by going to the latest known position, which updates everytime the player is seen. If the player manages to hide from the NPC and they reach the last known position; they start looking around for a time, after which they will either find the player or return to their daily routine.

There's an NPC that acts different from the rest: the main objective. Once they know the player is coming to find them; they start trying to get away from them; increasing the speed once they hear them or see them, only calming after some time without hearing from the player whereabouts.

4.2.3 Dimension Shift

Dimension Shift is a special mechanic activated by pressing the left mouse click or the right trigger in a controller and deactivated by pressing the right mouse click or left trigger in a controller.

This mechanic freezes the universe and separates a part of it around the player. It then allows the player to place certain traps inside the area, which will all activate once the shift is finalized.

It requires a certain "energy" to work, which is consumed when entering and when traps are placed in the world.

The traps that can be placed allow for controlling the NPCs all around the place. Those traps include a bear trap that stops any NPC that walks on top for a certain duration, a speaker that plays sounds to distract any close NPCs to watch at it; a cardboard cutout of the ninja, making any NPC that sees it to go closer and examine it isn't the real one, a "tar" pit that slows down any civilians that come in contact, etc.

When the player deactivates the Dimension Shift mechanic, time resumes and all NPCs continue doing their daily life; and, at the same time, all traps activate: The bear trap is open, the speaker starts making sounds, etc.

Most of the time, using this power will result in modifying something minor in the world; like the window state of the buildings or the colour of certain elements all around the universe; this is due to the universe trying to solve the problems brought by those manipulations.

4.3 Level creation

4.3.1 Level Diagram

Due to the roguelite inspirations in the game development; there are no defined levels, instead opting for a single level to play each “run” which offers a randomly generated challenge.

4.3.2 Level Design / World Generation

Unipurge takes place inside a procedurally generated city, which impedes the classical level design system, using instead a set of processes and restrictions to generate the city.

Such city is completely generated at the loading screen (just at the start of the world), by using a combination of generation algorithms via layering them on top of each other and combining their results.

The generation algorithm begins by placing rivers in the city area; it then follows up by layering the roads and generating them, it then continues by creating groups of either buildings or parks, finally all blocks generate extra elements, like walls, stairs, lamps or doors.

The procedure begins by generating a proportional amount of rivers in the city, which is then followed by the generation of roads with WFC (Wave Function Collapse) [6]; the remaining spots are then separated with an “inflation” algorithm which creates blocks of buildings.

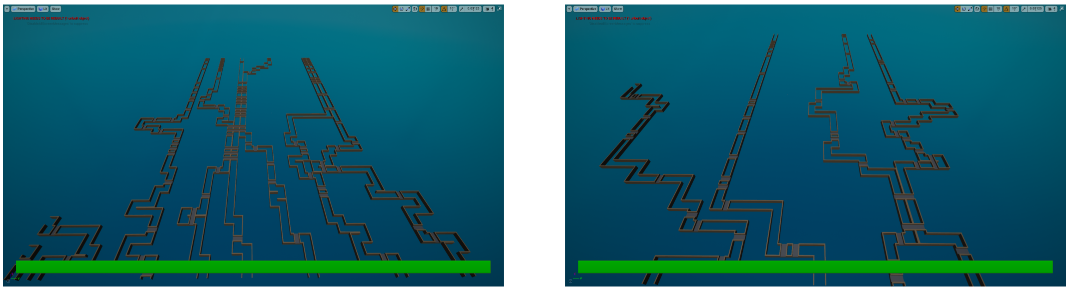
Lastly, such buildings are generated and select additional elements like walls are added according to the information of their surroundings, additionally the roads and parks also generate their decorations.

Due to some modifications done to the algorithms the generation ended up being deterministic, allowing for the sharing of seeds.

The first step of the generation comprises the creation of some rivers alongside the whole range of the city. Their generation uses a linear pattern where there is a desired direction for the river to flow and the algorithm tries to follow it, all while including some amount of turns in its path, only to increase the organic feeling of the rivers.

This rivers also generate bridges alongside their straight paths which can even combine with adjacent bridges to create a road that does not end abruptly in the middle of nowhere.

This algorithm allows for a more twisty river than simply using a pathfinding algorithm to accomplish a similar objective, since it includes a random factor of deviation from the original route. (Figure 4.1). Since the rivers can be part of the road ecosystem; they must be able to generate bridges across any amount of rivers, and, as shown in the figure 4.1a, the algorithm is capable of doing so.



(a) A boosted generation of rivers

(b) A normal generation of rivers

Figure 4.1: River generation Examples

The road layering step uses, as indicated previously, a modification of the Wave Function Collapse algorithm, which is, in turn, a modification of the Model Synthesis [7] algorithm for generation.

Model Synthesis, as described in the paper from 2009 from Paul C. Merrell, is a generation algorithm based on C++ that functions in 3D environments with a grid base. It works by selecting a cell at random from the grid and selecting one of the possible options to fill it with; marking all other possibilities as impossible and propagating the consequences of such action with the AC4 algorithm. This procedure is then repeated up until all the tiles in the grid are filled.

The propagation of consequences works recursively by modifying the available options of adjacent tiles according to the current options of a tile. It then keeps executing up until no options are modified, at which time the propagation stops in that tile. This reduction in posibilities is done by using constraints which delimit which tiles can be adjacent to each other and, in turn, when an option does not accomplish any constraint that has linked, it gets removed from the list since it is no longer capable of being placed without generating contradictions.

The algorithm works most of the time, but it has some major problems, the major of which being the existence of contradictions which occur when no possibilities are present and end up breaking the whole algorithm.

WFC [6] is a modification of Model Synthesis developed by Maxim Gumin created specially to take Model Synthesis' random generation into 2D environments. It uses the colour of certain spots of the tiles as the constraints and utilizes a Least Entropy heuristic to select which cell to fill next.

The heuristic used in the algorithm uses the amount of options available to each cell to select the next cell to be filled; decreasing the chances of generating a contradiction when options are chosen at random and end up making a tile unfillable due to the lack of options.

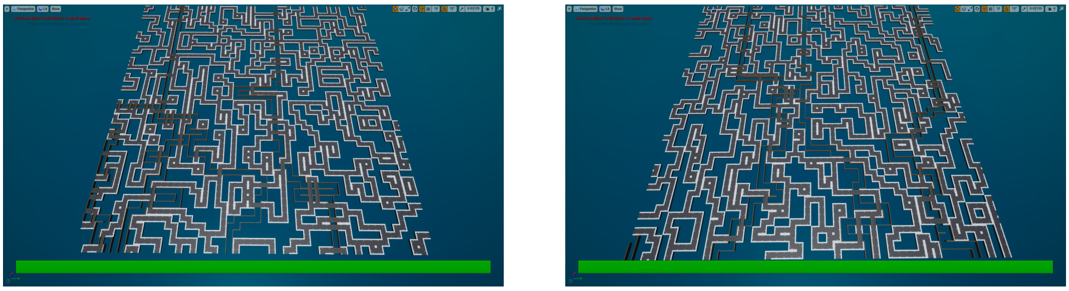
Even though the system attempts to fix the contradiction problem, it is not fault proof and, to fix possible contradictions, the algorithm allows for certain modifications. Such fixes requiring to encompass many tiles into a group where the algorithm is applied; since those contradictions are much less probable inside smaller environments. Due to the possible inconexion with the adjacent groups; this procedure requires to include the edges of previously generated groups to be in the both groups; or to implement backtracking into the algorithm, which keeps track of previous random elections and backtracks to the last one to remove the previously selected option since it ended up making a contradiction. This both approaches can be combined to help even more with the error mitigation in the algorithm.

The modifications done to the WFC algorithm done in this project contains an introduction of pseudo 3D environments which are 3D world where the algorithm only works in two dimensions; the grouping of various blocks in groups according to their conditions, allowing for a reduction of verifications required when removing possible options in open tiles; a biasing whenever a random selection of tile is required to fill the block, a modification to make such generation deterministic and based upon a seed; and a separated part made only to take care of any contradictions by using a secondary calculation with all blanc filling options.

Of all the restrictions used in the implemented version of the algorithm includes having to connect all roads, trying to not leave any road without exit; forcing empty spaces in the adjacent tiles to roads that aren't connected to other roads and prioritising roads with low amount of exits. The last two restrictions were implemented after all the generation ended up being a net of roads without many empty spaces to put buildings into.

Those changes made the city too few roads and forced to bias all the random placements to incentivize more roads into the environment.

Some results of the final road generation algorithm can be seen in the following (Figure 4.2a) (Figure 4.2b)



(a) An example of road generation

(b) Another example of road generation

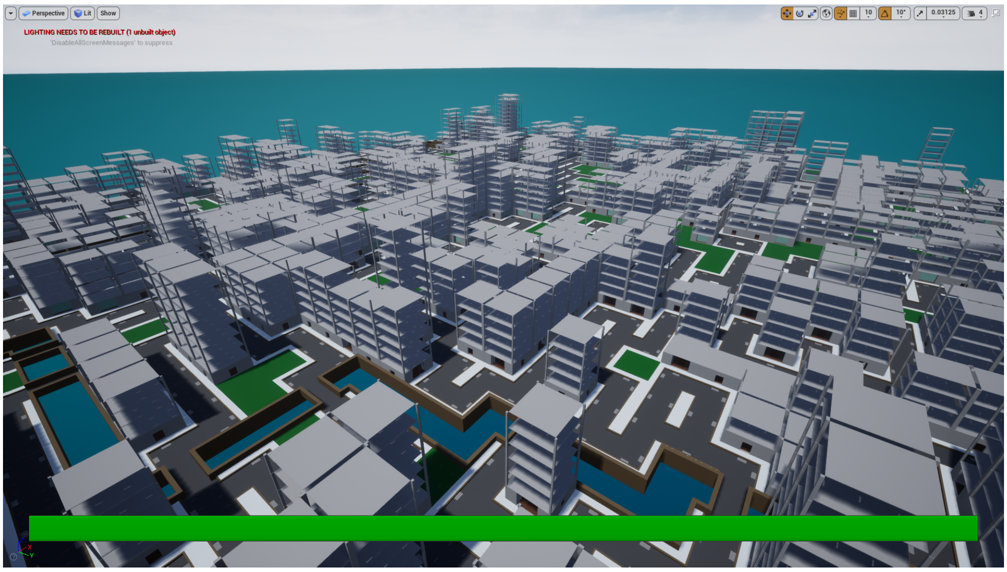
Figure 4.2: Road Generation examples

The second main step in the generation system is to create groups with the empties left by the road and river generations. To accomplish its goal it first searches for any road in all the grid.

Once a road is chosen, its neighbours are analysed to know if they are currently empty; whenever a positive comes up, a new group is generated and gets assigned its type, building or park; its height and its area, each value selected randomly but within certain limited range; allowing for extra control of probabilities.

After the selection, a recursive analysis of all adjacent blocks to the initial group block is initialized, selecting any empty tile and adding them to the possibilities of connection. From those possibilities one is chosen and their adjacencies are also added. This procedure then repeats itself up until there are no more empty tiles left or the size matches the desired one.

This procedure is chosen over generating everything with the WFC algorithm done previously so that groups of multiple items can be created, instead of 1 tile modules that the Wave Function Collapse would have made; and to avoid repetition since, as it is stated by Brian Bucklew at the Game Developers Conference [5], whenever a Wave Function Collapse algorithm uses many blocks to generate, the generations it comes up with tends to be very homogeneous and rarely differ much from the original pattern, thus making non interesting results in the world generation. (Figure 4.3)

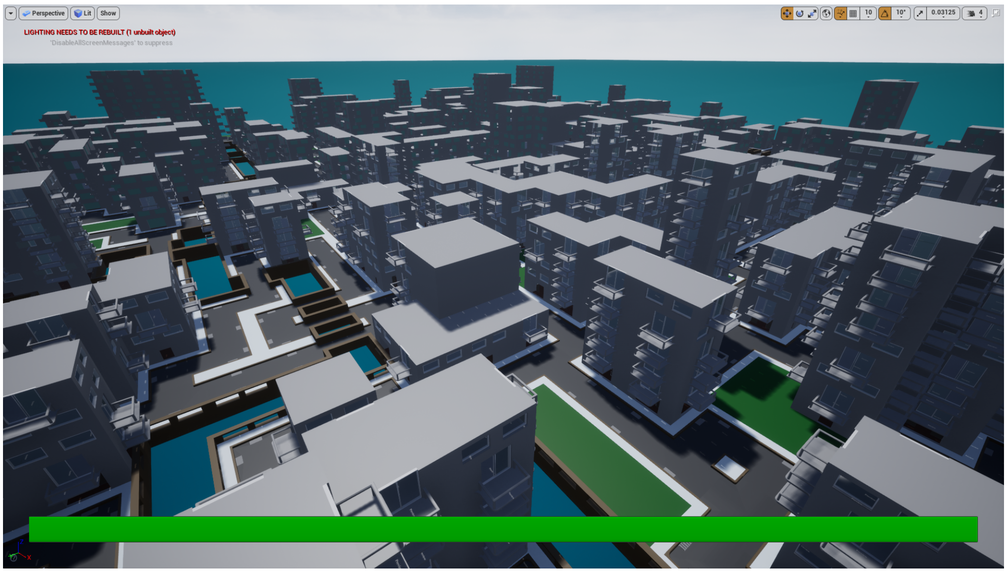


LIGHTING NEEDS TO BE REBUILT (1 unbuilt object)
'DisableAllScreenMessages' to suppress

Figure 4.3: An example of generation after the grouping step

Lastly, one last pass trough all the tiles gives each other information from its neighbours so that they can generate, not only additional items like lamps or trees; but also allowing the buildings to decide which walls to place on each of their four sides and allowing to build upwards. (Figure 4.4)

The generation process, while not being very optimized, it works properly and generates a believable city without those basic straight line roads, but with winding and



LIGHTING NEEDS TO BE REBUILT (1 unbuilt object)
'DisableAllScreenMessages' to suppress

Figure 4.4: An example of the final basic generation in the game

turning roads. This encourages the player to use the climbing mechanic instead of simply running towards the objective directly.

In between the generation steps, another generator is used to create the data required to manage all the NPCs from the city; each one receiving a simple patrol between diverse points and a spawning point.

Since the NPCs are almost always moving around in the environment, recalculating routes and they each require an AI controller; there cannot be an infinite amount of characters in the world, even when disabling them by distance. So that, in order to fix the very big hit on performance they have, only the NPCs around the player are spawned and move around so that a populated environment can be simulated; taking their spawning location and patrol from the tile they are assigned to and being destroyed once the tile gets too far from the player. Sadly, this creates popping when the main character moves very fast or the render is quite low; but this effect can be mitigated in more powerful hardware by increasing the NPC render distance so that the range is bigger.

4.3.3 Updating the world

Since the game operates on multiverse theory and universal modification, the Dimensional Shift mechanic changes the whole world every time it is activated; changing minor elements like wall models or trees, to name a few.

This modifications were implemented originally with a linear system where everything would be changed upon entering the pocket universe; but it ended up stopping the gameplay for a good 5 to 6 seconds each time, thus hindering the experience and the fluidity of the game, as well as making it harder to detect the variations in the environment.

In order to fix such problems, an asynchronous system was adopted. Once the player enters Dimension Shift it triggers a function that starts updating a certain amount of elements from the world at a time, helping to reduce the amount of delay from one frame to an other. The amount is controlled by the world size, making it so that in the case the world is much bigger, the generation also takes more tiles per frame; taking advantages from the better hardware that allows for bigger worlds.

4.3.4 Optimizations

Due to the modularity of the project and their expandability; many optimization was required to get the game running properly in most computers.

Some small memory optimizations are centered around specifically the grid and how elements are stored in it. Firstly the grid itself is stored as a simple array with some functions to read and write with 2D coordinates instead of 1D, then all elements in the grid is stored in only one array which contains a pointer to an Enum Tile, mainly filled with Enums or pointers; thus reducing the need of keeping copies of elements.

The next main element of optimization is Distance Culling. A precalculations effort that, with a single check, removes most of the elements from being even calculate for rendering, improving the performance by a massive amount. This distance can even be increased in the options menu so that, in better machinery, the amount of area seen at a time is higher.

Following that, there is Frustum culling, a basic optimization system that, utilizing the frustum of the camera, what it can see, it removes all elements that never crosses the view; thus reducing the load upon the graphics card.

After that, a Dynamic Occlusion is applied to the remaining elements by utilizing Hierarchical Z-Buffer which improves over the traditional Z-buffer Occlusion by collapsing distances into smaller resolutions and iterating over those, allowing for early rejection compared to the traditional.

A Dynamic Occlussion method is utilized since it is impossible to precalculate the occlusion in a procedurally generated environment that can shift over time, thus a dynamic answer has to be chosen; but, since regenerating the buffer each frame is so costly for the game, many previous reductions are required.

The final optimization done for the environment is Level of Detail. This optimization does not do much in this game optimizations since they are already quite simple and the game is not that taxing of the GPU; but they are implemented mainly to hide the popping from the Distance Culling by mainly opaquing the windows seen from a high distance.

Chapter 5

Results

Contents

5.1 Work Development 39
5.2 Planning Control 40
5.3 Changes, Difficulties and Solutions 40
5.4 Milestones 41
5.5 Results 41

5.1 Work Development

The game was too ambitious for the allocated time and the ability of the developer was not on par with the required one to properly bring the procedural city to life. Despite such inconveniences, the final project ended up being a more than serviceable version of the original idea, as well as fixing many of the problems with the design of both the generation algorithm and the gameplay loop.

The gameplay fix came within the design phase, where there was an important problem when it came to mechanic design, their possible entertainment and their relation to the other parts of the game; which mainly occurred due to the lack of experience with stealth games and their gameplay loop and its way to entertain players.

The original idea for the generation algorithm was to create a custom algorithm which would supply most of the requisites of the city, but it turned out to be very problematic and not understandable.

Since this is a project all around procedural generation, the major milestones were all related with the generation of the world and its modifications, as well as progressing through the various stages of such.

5.2 Planning Control

Despite the original planning being quite organized; it could not be properly carried out as desired. There were two main deviations from the original planning.

The first one occurred due to the major problem of the algorithm used to generate the city. The original algorithm was intended to be an all-in-one solution to the generation problem; but ended up being too complicated and hard to implement and explain; and did not manage to accomplish any of their objectives without breaking other ones. This system had to be completely overhauled and, in turn, forced changes in almost all programming related to the system; as well as a deviation from AI and Stealth mechanics towards more time invested into procedural generation. The delay was also highly impacted by the lack of awareness about the inner workings of the Unreal Engine platform and the differences with Unity, as well as its programming language, C++.

The second deviation was quite later in the timeline, and it encompasses the stealth mechanics and the NPCs of the world.

Initially there were supposed to be two types of NPCs (Civilians and Law-Enforcers), each one with their own quirks and actions. The Civilians were to walk around the world like how NPCs do already, but also interact with each other. Create groups and accompany each-other; as well as turning scared if they found the player or another civilian "scared" them; calling the Law-Enforcers. The Law-Enforcers were to be NPCs that would tend to stick close to a defined building and would only move in the case a scared civilian called them. They would be the only NPCs capable of originating a Game-Over.

This system had to be completely removed and substituted by the basic NPCs the game has , which combines elements from both. This was due to the lack of computing power to keep so many NPCs at the same time.

5.3 Changes, Difficulties and Solutions

The project had many problems in its development; some about the main mechanics of the game and others about the environment generation.

The base was always conceived as a stealth game, but the specifics changed along development. In the beginning it was all about avoiding conflict by using the stealth mechanics; but it did not seemed very fun or enjoyable to play around; so the focus of the game changed to a more of a “trickster” style, where the Dimensional Shift instead of allowing the protagonist to avoid everything, it forced them to return to the same spot but allowing for the creation of traps to confuse the NPCs in the world; making the tool more of a tool and less of a “get out of jail free” card.

The last modification to the design came when finding out about the limitations of the engine when it comes to manipulating multiple AI at the same time; since each AI required a AIController and made its own calculations, having differences between civilians and law-enforcers would have limited the amount of NPCs at the same time if the performance was to be maintained. To solve such a hurdle, it was changed to an approach that any NPC can hear, see and catch the character if they manage to reach them.

When it comes to the changes in world generation; those are really big, since originally an original algorithm was supposed to be used; but it ended up generating a lot of problems; particularly when it came to generating realistic roads and building clusters in the world, as well as being very cumbersome to explain and understand.

This difficulty was solved by changing to the current world generation system by utilizing layers and various algorithms in order to generate a more or less believable city landscape.

5.4 Milestones

There were many milestones in the game, but the most important ones were related to the world generation, since it is the main focus of the video game.

The first milestone came when the old generation system was able to generate a road system on top of nothing; utilizing only a point in space and its own algorithm, and which changed every time the game was started.

The next milestone of the project was the substitution of the flawed algorithm with the final one, which managed to do a better result while being easily understandable.

Following that milestone; there was the grouping of empty spaces into groups and the assignation of certain common elements, like type of walls and height, which also got randomized every time the world was generated, keeping everything procedural while giving it a more realistic system.

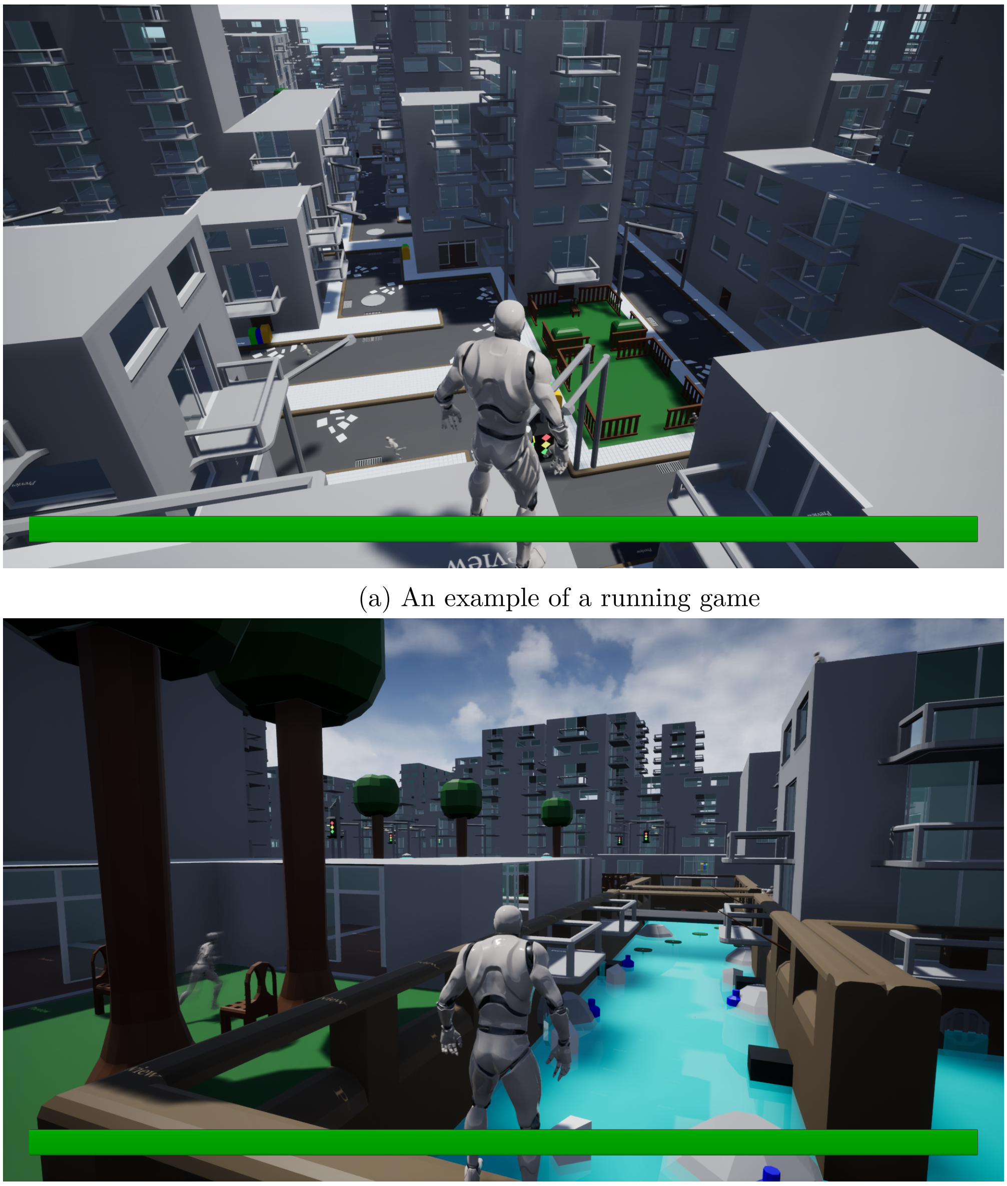
After such milestone came a proper NPC generation system, which was the third iteration on how to accomplish such event, which managed to improve drastically on performance or and gave more control over individual elements.

The last milestone in the project was the implementation of the modification of the secondary elements of the environment every time the player goes through dimension shift and the effect it generates.

5.5 Results

Repository: https://github.com/Ma1pa/UniPurge
Build: UniPurge V1.0.0

The final game ended up being a bit more underwhelming than the envisioned product; but since it was so ambitious, it still accomplished most of the important beats it tried to reach (figures 5.1a, 5.1b), only falling short in the more advanced elements like more advanced AI or a real 3D generation system, instead of the 2.5D one used at the moment. Even though the game is not as good as it may be, it is an acceptable product with enough defining elements, as well as a great canvas for any future modifications or improvements to bring it up to par with the original idea of the project.



(a) An example of a running game

(b) Another example of a running game

Figure 5.1: Final game examples

The final generation algorithm ended up being much less original than the initial designs; making it much less innovative since it is only a layering system of algorithms on top of other, each one being different; linear algorithms, a modification of the original WFC [6], an algorithm based upon an inflation system and other iterables. This system is, as opposed to the original one, much more robust and easy to explain, since it takes place in separated steps; allowing for the activation or deactivation of steps in order to show certain changes or to help troubleshoot.

When it comes to the game engine used in the project, Unreal Engine; Unipurge has helped much in the understanding of the basis of the engine, how it loads elements, how it is compiled and the tools it provides to modify the games developed on the engine.

The project ended up with two big results. The first one is the new understanding about how a system of procedural generation works and how it can be properly implemented, alongside the steps it undergoes along to generate. With the designing of the system, a basic understanding on algorithms and their design was attained; how to design a proper algorithm, the fact that iterating on top of previously established systems does not detract from the original work and how to explain such algorithms to other people.

The second big result of the project is related with the optimization systems and their impact on the final product. Not only the basic oclussion culling or frustrum culling; but also with the render distances and asynchronous functions. All of them being very impactful in allowing the generation of bigger environments while keeping a low amount of stress on the CPU or GPU.

Chapter 6

Conclusions and Future Work

Contents

6.1 Conclusions **45**
6.2 Future work **46**

In this chapter, the conclusions of the work, as well as its future extensions are shown.

6.1 Conclusions

UniPurge has been quite an arduous experience to create; mainly due to the effort needed to learn how an unknown game engine as Unreal Engine works and learning how to implement elements in it, specially when it came to programming in c++ since the documentation is quite lacking in its section. But it was well worth the effort since it helped develop knowledge on different workflows and programming languages, as well as their own elements like the blueprint system.

When it comes to the generation, it was also quite a big task, mainly due to the incorrect approach I had when starting to develop an algorithm to generate everything; it had too much deviation from already known techniques which ended up wasting many hours of the project in a fruitless endeavour, since I ended up scraping most of the elements and ended up settling with the current "layered cake" approach.

The game design itself was not very big due to the randomness of the environment and the simplistic design of the game itself; but it ended up being even smaller due to technical restrictions mainly due to the AI and its behaviours in the engine.

Having to recheck everything and ensure good functionality with a variable, but very high, amount of actors in the world, each one with their own actions and reactions was a very interesting aspect of the project; mainly due to knowing and learning the optimizations systems and implementing most of them by, not only engine tools, but also by code sometimes.

6.2 Future work

Several elements in the game can be either expanded or improved further to improve performance of the game, variability or even the expansion of it. It is planned to improve the game in the following months until a satisfactory result is reached.

- The first change that would add much more to the world would be to implement WFC to the decorations of buildings, allowing for the groups of tiles to generate more realistic walls and windows distributions, as well as better stairs placement.
- There could always have more models of elements included in order to increase variability in the world design.
- Utilizing the groups of houses or parks to add variation within groups, like the addition of brick layered houses or child parks to name a few.
- The expandability of the game can be improved by changing the blocks spawning system for the environment to work closer to the NPCs spawning system so that the maximum size of the world can be expanded truthfully. It would be made by only generating the amount of elements visible at a time and killing them whenever they exit the visible zone.
- Another quite big change that could be implemented into the project would be height variation roads and elevated footbridges connecting higher elevation floors, thus making the environment much more varied and increasing the importance of elevation in the generated worlds.

Bibliography

[1] Audacity. https://audacity.es/. Version Accessed: 2022-06-30.

[2] Krita Foundation. Krita. https://krita.org/es/. Version Accessed: 2022-06-30.

[3] The Blender Foundation. Blender. https://www.blender.org. Version Accessed: 2022-06-30.

[4] Epic Games. Unreal engine. https://www.unrealengine.com. Version Accessed: 2022-06-30.

[5] FreeHold Games. Tile-based map generation using wavefunctioncollapse in *Caves Of Qud*. March 2019.

[6] Maxim Gumin. Wave function collapse algorithm. https://github.com/mxgmn/WaveFunctionCollapse. Version Accessed: 2021-10-21.

[7] Paul C. Merrell. *Model Synthesis*. Dissertation, University of North Carolina at Chapel Hill, 2009. https://paulmerrell.org/wp-content/uploads/2021/06/thesis.pdf.

[8] MrSmith. Remember. https://freemusicarchive.org/music/mr-smith/fredson-drive/remember-1/.

[9] Robinhood76. 05913 swimming loop.wav. https://freesound.org/people/Robinhood76/sounds/317067/. The file was reduced and pitch modified before adding it to the project.

Appendix A

Source code

Since this project is very centered around code, especially in the generation system; I decided to insert both the code for the generation of the Roads and the generation of Houses into the document to make more concrete how they are handled. This two examples were picked since they are some of the more complex systems inside the generation procedure; other systems like the river generation or the rest of the Wave Function Algorithm are also quite complex; but in order not to overfill this section, they were left out.

A.1 Procedural generation Algorithms

Roads Generation

This generation iterates over all tiles in the grid by requesting the tile with the least entropy to the "generator" class, which is used to store the information of the grid. After receiving the tile; an option is chosen between the available ones, giving higher weights to the ones with fewer exits associated. When selecting a block, we then set the block in the "generator" and they update the data on the tile and recursively act on adjacent tiles to remove options according to the constraints. This same function also adds to the options the single exit roads in case the options are empty so that no problems can be derived.

```
void AGameMaster::GenerateRoads()
{
        for (int i = 0; i < Side * Side; i++)
        {
        //We get the Tile with the least entropy and which is yet to be filled
```

```
                std::pair<int, int> Point = Generator->GetLessEntropy();
                //We chose if we put empty or not. Weighted towards road due to the rules
                std::discrete_distribution<int> var({ 1,0.7 });
                int option = var(generator);
                //Check if an empty is possible
                if (option == 1 && Generator->CanEmpty(Point.first, Point.second))
                        Generator->CollapseOptions(Point.first, Point.second, 1);
                else
                {
            //Check if there are no options left
                        if (Generator->GetPosibilities(Point.first, Point.second) <= 0)
                                Generator->CollapseList(Point.first, Point.second, -1);
                        else
                        {
                                std::vector<int> dist;
                                std::vector<int> op = Generator->GetOptions(Point.first, Point.second);
                                for (int j = 0; j < Generator->GetPosibilities(Point.first, Point.second); j++)
                                {
                                        //We assign weights according to the exits and their distribution, favoring straight lines
                                        if (op[j] == 6 || op[j] == 11)  dist.push_back(10);
                                        else if (op[j] < 12)    dist.push_back(5);
                                        else if (op[j] < 16)    dist.push_back(2);
                                        else if (op[j] == 16)   dist.push_back(0.1f);
                                        else if (op[j] == 17 || op[j] == 23)    dist.push_back(1);
                                        else
                                             dist.push_back(0.2f);
                                }
                                std::discrete_distribution<int> distribution(std::begin(dist), std::end(dist));
                                option = distribution(generator);
                                Generator->CollapseList(Point.first, Point.second, option);
                        }
                }

        //We generate the NPC information
                if (Generator->GetBlock(Point.first, Point.second) != Block::EMPTY)
                        GenerarActor(Generator->GetBlock(Point.first, Point.second), Point.first, Point.second);

                //Generate places to place NPCs and their waypoints
                int X = (int)(i / Side) * GridToCoordMult;
                int Y = (int)(i % Side) * GridToCoordMult;
                Generator->StorePosition(i, FVector{ StaticCast<float>(X), StaticCast<float>(Y), 250.0f });
                FVector puntos[4] = {
                                        GenerateNPCPath(i, 0),
                                        GenerateNPCPath(i, 1),
                                        GenerateNPCPath(i, 2),
                                        GenerateNPCPath(i, 3)};
                Generator->SetWaypoints((int)(i / Side), i % Side, puntos);
        }
```

```
}

//Function used to spawn a Block actor or to update an already placed one
void AGameMaster::GenerarActor(Block ChosenRoad, int XPosition, int YPosition)
{
        const FVector Location = {StaticCast<float>(XPosition * GridToCoordMult),
             StaticCast<float>(YPosition * GridToCoordMult), 10.0};
        const FRotator Rotation = GetActorRotation();
        ABaseBlock* actor;
        if (Generator->GetActor(XPosition, YPosition) == nullptr)
        {
                actor = GetWorld()->SpawnActor<ABaseBlock>(ActorToSpawn, Location, Rotation);
                std::discrete_distribution<int> alt({ 1,1.5 });
                Generator->AddAgent(XPosition, YPosition, actor, ((ChosenRoad < Block::RIVER_N_S
                     || ChosenRoad >= Block::BUILDING) && alt(generator) != 0));
        }
        else
                actor = Generator->GetActor(XPosition, YPosition);

        if(ChosenRoad == Block::BUILDING)
                Actualizar.push(actor);
        else
                actor->SetStats(ChosenRoad, Generator->GetHeight(XPosition, YPosition));
}
```

Houses Generation

The house generation portion is mostly done in this code. First the code checks for all tiles in the grid and when it finds a road it tries to find an empty adjacent tile from witch to launch the function "Group Houses". This function assigns a group number, random height and class of tile (if it is a building or a park); and then checks the adjacent tiles of the selected tile in search for empties. Once one is found, the values are stored and the process is repeated. This works until there are no more empties left to fill or the desired size has been reached.

The decoration generation in houses consist in adding each building actor into a queue and, once all empties have been filled, they start getting called once after the other with the "ActualizarActor" function which gives the actor the information of adjacent blocks so that they can decide where to place a door, wall or window.

```
void AGameMaster::GenerateHouses()
{
        int group = 0;
    //Search all the city for any road or river
        for (int i = 0; i < Side; i++)
        {
                for (int j = 0; j < Side; j++)
                        //Check for a road or river
                        if (Generator->GetBlock(i, j) > Block::EMPTY && Generator->GetBlock(i, j)
                             < Block::BUILDING)
                        {
                                std::discrete_distribution<int> ProbParque({ 2,0.5 });
                                //We search the adjacent blocks. And if any of them is empty, generate a building and start generating a group
                                if (i < Side - 1 && Generator->GetBlock(i + 1, j) == Block::EMPTY && Generator->GetGroup(i + 1, j) == -1)       GroupHouses(i + 1, j, group++, ProbParque(generator) == 1);
                                if (i > 0 && Generator->GetBlock(i - 1, j) == Block::EMPTY && Generator->GetGroup(i - 1, j) == -1)      GroupHouses(i - 1, j, group++, ProbParque(generator) == 1);
                                if (j < Side - 1 && Generator->GetBlock(i, j + 1) == Block::EMPTY && Generator->GetGroup(i, j + 1) == -1)       GroupHouses(i, j + 1, group++, ProbParque(generator) == 1);
                                if (j > 0 && Generator->GetBlock(i, j - 1) == Block::EMPTY && Generator->GetGroup(i, j - 1) == -1)      GroupHouses(i, j - 1, group++, ProbParque(generator) == 1);
                        }
        }
        //We operate on all the houses to build the sides
        while (!Actualizar.empty())
        {
                ABaseBlock* actor = Actualizar.front();
                Actualizar.pop();
                ActualizarActor(actor, actor->GetActorTransform().GetLocation().X / GridToCoordMult, actor->GetActorTransform().GetLocation().Y / GridToCoordMult);
        }
}

//Function to group tiles into groups
void AGameMaster::GroupHouses(int X, int Y, int group, bool park)
{
    //Decide the amount of tiles in the group
        std::discrete_distribution<int> var({ 0.5,1,2,1,0.5 });
        int modifier = var(generator) - 2;
        //The amount of blocks we want in the group
        int groupSize = AverageGroup + modifier;
        std::stack<std::pair<int,int>> posibilidades;
        int iterator = 0;
    //The distribution of heights
        std::discrete_distribution<int> alt({ 1.25,2.5,5,10,20,10,5,2.5,1.5,1,0.8,0.6,0.4,0.2,0.1 });
        int altura = alt(generator);
        posibilidades.push(std::pair<int,int>{ X,Y });
        //Iterative recursion to check all neighbours
        while (iterator++ < groupSize && !posibilidades.empty())
        {
                std::pair<int, int> actual = posibilidades.top();
                posibilidades.pop();
                Generator->CreateHoses(actual.first, actual.second, group, altura, park);
                GenerarActor(Generator->GetBlock(actual.first, actual.second), actual.first, actual.second);
                //Check the sides
                if (actual.first < Side -1      && Generator->GetBlock(actual.first + 1,
                     actual.second) == Block::EMPTY)        posibilidades.push(std::pair<int,
                     int>{ actual.first + 1, actual.second });
                if (actual.first > 0           && Generator->GetBlock(actual.first - 1,
                     actual.second) == Block::EMPTY)        posibilidades.push(std::pair<int,
                     int>{ actual.first - 1, actual.second });
                if (actual.second < Side -1 && Generator->GetBlock(actual.first, actual.second +
                     1) == Block::EMPTY)    posibilidades.push(std::pair<int, int>{ actual.first,
                     actual.second + 1 });
                if (actual.second > 0           && Generator->GetBlock(actual.first, actual.second
                     - 1) == Block::EMPTY)        posibilidades.push(std::pair<int, int>{
                     actual.first, actual.second - 1 });

        }
}

//Function used to update each building to put the proper addons
void AGameMaster::ActualizarActor(ABaseBlock* actor, int X, int Y)
{
        int group = Generator->GetGroup(X, Y);
        actor->toggleFloor();
    //The group of both adjacent blocks is compared and a value is asigned accordingly
        actor->SetNewExits(     Generator->CompareGroup(X + 1, Y, group),
                                                Generator->CompareGroup(X, Y + 1, group),
                                                Generator->CompareGroup(X - 1, Y, group),
                                                Generator->CompareGroup(X, Y - 1, group));
    //The addons are saved accordingly; even to the blocks upwards
        actor->SetStats(Generator->GetBlock(X, Y), Generator->GetHeight(X, Y));
        //The building is updated with the new info
        actor->UpdateBuilding();
}
```