

Metal Metal Land

Final Year Project Report

DT228

BSc in Computer Science

**Student: Paul Redmond**

**Supervisor: Mark Foley**

School of Computing

Dublin Institute of Technology

**02/05/2018**



Abstract

“Metal Metal Land” is a procedurally generated, multiplayer action game implemented using the Unity (C#) game engine with the goal of it being as fast paced as possible with as minimal load times as possible[?]. It involves two players selecting characters, each representing different sub-genres of metal (such as Old-School British Metal, Pirate Metal, Thrash Metal etc.) and fighting to the death in a procedurally generated, destructible arena using a variety of weapons, power-ups and environmental objects to do so.

The key goal of this project is to produce a fast paced local multiplayer action game. As technology has improved, the physical interaction between players as they play games with one another has faded away more and more, being replaced with online multiplayer almost entirely. The absence of traditional “couch-based” local multiplayer in modern games has not gone unnoticed and there has been a cry for a return to the local multiplayer games of old from a large section of the gaming community. I am in agreement with this view and while I appreciate the convenience that the modern online-based multiplayer provides, it does leave me wanting for games similar to the ones I grew up with.

When the game boots up, players are presented with the main menu, containing links about how to play the game, game options and the actual play-game button. If players are already familiar with the game and its rules, they should be able to begin playing the game within 10 seconds of arriving on the main menu. Once they begin playing, they will be presented with minimal interruptions so as to not negatively impact the flow of the game. The feedback from initially testing this game showed me that people were very open to, and fans of, the minimal load times and rapid pace of the game.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<Student Name>

<Date>

Acknowledgements

Firstly, I would like to thank my parents who have supported me physically and emotionally over the past few gruelling months. I’d also like to thank the lecturers from DIT who helped with this project, Dr. Susan McKeever, Dr. Bryan Duggan and in particular my own mentor Mark Foley, who helped and supported me to the absolute best of his ability throughout the entirety of the this project.

Additionally, I’d like to thank my classmates; Aaron Renaghan, Eamon Tang, Jay Hynes, Alannah Mullins and Oleg Petcov for all the great times we’ve had over the last 4 years. I’d also like to thank Ally Daly for keeping me plied with burritos, pints and the most adorable dog pictures over the last year and beyond to help keep me motivated.

I’d also like to thank the people who helped me test my project and gave me feedback throughout; Jonathan Bradley, Emily Kremers, Ally Daly, Michael Clonan, Aaron Jennings, Gary Raymond, Liam Phelan, James Hassett, Nicola O’Brien, Max Curtis, Haashim Bari, Ross Murphy, Eamon Tang.

Finally, I’d like to thank all the Metal bands, without whom this wouldn’t be possible, who inspired me to make this game such as “Metallica”, “Alestorm”, “Gloryhammer”, “Black Sabbath”, “GWAR”, “Judas Priest” and the countless other bands that have fuelled my passion for this genre. In particular I’d like to thank “Lagerstein”, who generously allowed me to use their music in the game and most importantly of all, I’d like to thank “Iron Maiden” for introducing me to the genre and for remaining my steadfast favourites over the last 14 years.

Table of Contents

*(These are suggested contents only. You can add or modify if required)*

1. Introduction
   1. Overview of the project and the background behind it.
   2. Project Objectives
   3. Project Challenges
   4. Structure of the document
2. Research
   1. Eg. Research related to identifying the problem that this project solves, research into solution definition
   2. An overview of the technologies evaluated and selected or rejected and the rationale behind the key decisions.
3. Design
   1. Identification of a design methodology including why it was chosen
   2. Design of each of the project components eg: the UI, Network, Project Demonstration, source code layout
   3. Clearly identifying the list of features and use cases supported within the project.
4. Architecture & Development
   1. Overview of the system architecture and a diagram to represent all of the key elements within the architecture.
   2. Details of each component within the project, problems encountered and resolved, challenges overcome or worked around.
   3. Identify key development components;
   4. Identification/explanation of external APIs used versus own code ; List of classes of your code etc .
5. System Validation
   1. Testing
      1. What testing was performed, why it was selected and what are the key use cases within the project.
   2. Demonstration
      1. Identify what features can be demonstrated and show screen shots or reference a video online to show the project demonstration (for audience not at demo)
6. Project Plan
   1. Project Plan analysis and review of how it changed from the initial proposal including explanation of what changed and why, and suggestions on how to address this if the project was repeated.
7. Conclusion
   1. Analysis of the projects key elements identify the key learning obtained from the project and recommendations and suggestions for how the work can be improved on continued into the future.
8. Bibliography
9. Appendix

# 1: Introduction

Local multiplayer games were originally a mainstay in the gaming industry. Multiple games popularity were defined by the depth and fun brought by their multiplayer experiences, this is despite the fact that some of these games were originally designed with single-player being at the forefront of the project. However, as video game consoles, PC’s and the technology that powers them have advanced, local multiplayer began to fade away in favour of entirely single player experiences or in favour of cloud-based online multiplayer games. This however has not been a transition without problems. While cloud-based online multiplayer games introduce an entirely new level of convenience, allowing players to play games with one another regardless of geographical position, it has brought with it a level of disconnect between the players. Some video game developers, independent developers in particular, have begun to notice the lack of traditional local “couch-based” multiplayer games and the nostalgic cry from the gaming community for a return to these simpler games.

A new technology that can be found in modern games to varying degrees of prevalence is Procedural Generation. Procedural Generation generally refers to the process of generating game content automatically, using a variety of different algorithms to do so. In the case of video games, this usually involves dynamic level generation and difficulty. The benefits of introducing procedural generation into a game is that it generally allows for over-all smaller file sizes while dramatically increasing the content available in the game. This allows for a potentially varied experience with each play-through of a game. Procedural generation in games in the last few years has been used to create objects of almost any form. This includes music, art, world design, enemy behaviour and more.

As previously mentioned with regards to local multiplayer and online multiplayer, technology is forever changing and improving. While this does allow for projects of grander scope and scale to be achieved, a consistent problem with some of these games is the introduction of multiple loading screens. These screens generally break the player’s level of immersion in the game and can take them out of the game.

In the following report, I will discuss my research into each of these 3 aspects that I seek to overcome with my project. Additionally, I will also include my own process into designing and developing the game as a whole and what potential future this project could have.

### 1.1 Project Overview

When the players wish to start the game, they will be greeted with a character selection screen. Each of these characters, while uniquely designed, all share the exact same functionality. The designs of the characters come from different sub genres of metal music including; Power Metal, Thrash Metal, Alt Metal, Pirate Metal, Old-School British Metal, Hair Metal, Death Metal and Metal Mascots. When each player has selected their character they will be brought to an options screen allowing them to have a certain degree of control over the levels and rounds they will play in. These options range from terrain type, which will control the general shape the terrain should adhere to (If it should adhere to any shape in particular at all), sudden death mode, round count and more. When these options are set, the players will spawn in a procedurally generated arena which will include everything they need to defeat their opponents. Depending on the player’s option choices, stage hazards such as explosive barrels or spike pits may appear. A number of weapon spawning altars will spawn in with the players, the number of which is determined by the overall amount of land squares that make up the arena. The players will use these weapons in order to try and defeat the other player with each shot generally equalling a kill if hit. When a player dies, a 1.5 second timer is activated. If the surviving player dies in this time, no point is awarded. However, if that player survives, there are awarded a point. The next level at this point is brought in, ideally with no wait time or loading screen as it should have finished generating while the previous round is in progress.

### 1.2 Project Objectives

The main goal of this project is to deliver a high quality, fast paced, local multiplayer action game. Additionally this game should have the ability to create destructible levels that are dynamically generated based on a very abstract player input (E.g. “Standard”, “Valley”. “Random” and so on) with each level being at a reasonable of playability (for example, a player should not spawn inside a set of blocks or on top of a spike pit). All of this should be accomplished at a rapid pace with minimum load times or intrusions which could potentially impact player immersion. In order to accomplish this, an efficient procedural generation algorithm shall be used alongside a-synchronous level generation (wherein the levels are generated in the background, masking potential load times). The game itself will be developed on Windows using the Unity Game Engine, specifically using C# as its core language.

### 1.3 Project Challenges

One of the challenges that I ran into was trying to find an algorithm which can procedurally generate levels of a reasonable level of quality efficiently. As previously mentioned, it is one of the key objectives of this project to generate these levels with no loading times apparent to the user, this includes the game stuttering, lagging or being adversely affected in some other form. While certain procedural generation algorithms are less complex than other ones, they may not offer the level of quality that I would be looking for in terms of level generation. As such, it is important to find the right balance between level generation efficiency and level generation quality.

Another challenge that I ran into while completing this project was animation. While I have in the past completed projects in Unity, I never under took a project of this scope and never designed my own art or animations for a game before. While initially designing the character and environment sprites, I felt that I was going into more detail than was originally needed and that creating roughly 15 sprites for each character would have wasted far more time than was really warranted. I therefore decided to settle on a more minimalist approach with regards to character and environment art which I feel not only looked a lot better, but saved a lot of time in addition to being a lot easier to work with animation wise.

Another challenge, and arguably the hardest to overcome when building a player versus player game like this, that I encountered was

PJR Note – add one more challenge

### 1.4 Structure of Document

# 2. Research

In this section of the report, I will explain the research that I undertook prior to beginning development on this game and what other research impacted the game as I was developing it. This includes what other projects I looked into and what impact I feel they had on the design of my own project. In addition to this I will also discuss what technologies were examined and used as part of this project. This includes game engines and procedural generation algorithms.

### 2.1 Project Background Research

When I was originally trying to come up with the basic design of this project, its theme and general gameplay were quite different from what they ended up being. Originally, the game was going to be called “Beat Rush” and would have 1 character from popular genres such as Blues, Pop and Metal. The characters originally were never meant to kill each other particularly quickly also, instead, they would have to damage each other over time using different weapons and moves before trying to finish their opponent off for the kill. The success rate of finishing off another player was based around the amount of damage that player had endured.

These ideas however were quickly changed when I started to examine other games I liked that would fit into this genre. Rather than rounds or player encounters taking a long time, like in the original version of “Beat Rush”, they would usually have a sort of “Glass Cannon” approach to player versus player reactions, meaning that while players can do a lot of damage to one another, they were also equally susceptible to damage. This glass cannon approach allows for rapid and quick encounters where players are generally able to kill their opponents within a very short space of time. The glass cannon approach allowed for a game to be not only fast paced but more simplistic as well which would allow for people who aren’t usually familiar with the genre to be able to understand and to get used to the general gameplay mechanics quickly.

Additionally, while trying to design characters for “Beat Rush”, I found that I had a lot more fun and a lot more ideas for metal and rock themed characters. I also noticed that when I looked at other games that I enjoyed, the characters generally revolved around one singular narrow theme. For example, in the game “Broforce”, which will be discussed in greater detail later in the report, player characters are based around parodied versions of famous action movie characters such as “John Rambo” and “The Terminator”. Another example of game using a narrow theme for its characters is “Duck Game”, wherein players just play as different coloured ducks.

Before beginning this project, I had only a small bit of previous experience with regards to game development. As part of my Object Oriented Programming module in the 2nd year of my course, I grouped up with two other friends to create a C# based game in the Unity game engine called “Pirates in a Barrel” [1]. Pirates in a Barrel was a tower defence game where players would have to defend a castle from a series of different pirates in different tiers of boats by building towers along the edge of a river. Originally, I struggled with getting certain mechanics to work as it had never something that I had actively attempted to do before. However, as the project continued on, everything began to fall into place and since completing that project, I have tried to familiarise myself further, focusing on the unity game engine in particular.

### 2.2 Alternative Solutions to the Project

- Duck Game

- Towerfall Ascension

- Broforce

- Brutal Legend

- Terrarial

### Technologies needed