REAL-TIME DYNAMIC DESTRUCTION IN COMPUTER GAMES

Reuben Miller

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Games Programming

University of Northampton

May 2024

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my dissertation supervisor, Dr. Anastasios G. Bakaoukas. His guidance, encouragement, and expertise were instrumental in the completion of this dissertation. His willingness to dedicate his time to discuss ideas, provide feedback on drafts, and offer support throughout the research process has been invaluable.

I am also grateful to my family and friends for their unwavering support and encouragement throughout this journey. Their belief in me has been a source of strength and motivation.

Abstract

This dissertation explores the integration of Real-Time Dynamic Destruction in computer games, with the primary focus on investigating, designing, and implementing a system capable of real-time fracturing of complex rigid bodies and applying stress propagation. The produced system contains 2 different modes, pre-fracture in which the object can be split up before runtime and the connections calculated, and a run time fracture where the object will be destroyed, and stress propagation applied. Results demonstrate that the developed real-time dynamic destruction system achieves a minimum frame rate of 30 FPS with low latency. As such this study successfully advances the implementation and knowledge around real-time dynamic destruction, providing a robust framework for future development in gaming and virtual simulations, potentially transforming interactive experiences.

Contents

Abbreviations

Acronym	Definition
FPS	Frames Per Second
FEM	Finite Element Method
UI	User Interface
RTDD	Real-Time Dynamic Destruction
RTDDS	Real-Time Dynamic Destruction System
VFX	Visual Effect
SPH	Smoothed Particle Hydrodynamics
MCA	Moveable Cellular Automata
HACD	Hierarchical Approximate Convex Decomposition

Tables and Figures

1. Introduction

In Modern game development, the integration of Real-Time Dynamic Destruction (RTDD) represents a pivotal milestone (Cameron, 2009). With modern games no longer being confined to static environments, developers now possess the tools required to craft environments that replicate the chaos of the real world (Unreal Engine, 2022). In this dissertation, the complexities of RTDD will be unraveled, exploring their technological foundations, implementations, and optimization methods.

By addressing gaps in the current methods, this research aims to offer a clear guide for developers to streamline the integration of RTDD. By collating the information around dynamic destruction, this research contributes to the advancement of realism and performance within RTDD, this will not only elevate games but will also open doors to new possibilities, such as virtual simulations and training environments (George, 2020). As such this study is not solely about refining a gaming feature; but about pioneering a shift in how we interact with virtual environments, Laying the foundation for the next generation of immersive and realistic virtual experiences.

The scope of this research is RTDD in the context of game development, so this research will be limited to solutions that adhere to the real-time condition meaning that they run above 30 Frame Per Second (30FPS) (Butler, 2014), and techniques limited to brittle fracture.

2. Background

Destruction in games often falls short of expectations, either the destruction is predictable and gets boring or destruction is just used as massive set pieces. such implementations feel bland and instantly remove the immersion from the player (Buday Baranowski and Thompson, 2012). For the few games that get destruction right, they often use in-house solutions (Appendix A) leaving the rest of the game development community in the dark. Resolving these issues is vital, as games thrive on immersing players in captivating virtual worlds (Steele, 2023), If the destruction inside of these worlds lacks authenticity, it removes immersion and disrupts the gameplay loop. This research aims to resolve that issue by contributing to the standardization of dynamic destruction in games.

Destruction has been a core element in video games since their creation, offering a cathartic release like no other (Simply Put Psych, 2023). The earliest example of destruction in video games is 1986’s Rampage (Bally Midway, 1986), an arcade classic that let players rampage through cities as giant monsters, eating civilians and flattening buildings, whilst the destruction featured in this game was nothing compared to that of the modern era, upon its release it was one of the first video games to feature destruction. It wasn't till 1994 when Iron Soldier (Eclipse Software Design, 1994) was released, finally allowing gamers to get their hands on another destruction-focused game.

Looking further in the future, Red Faction (Volition, 2001) first released in 2001, started a new wave of destruction with the innovative GeoMod engine (Volition, 2001), allowing players to dynamically destroy walls, tunnel through terrain, and collapse entire buildings. The release of Red Faction Guerrilla (Volition, 2009) in 2009 implementing the new GeoMod 2.0 offered even more destruction than the first version by implementing a stress-based system to accurately destroy buildings, massively increasing the realism of the destruction.

In 2010 Battlefield: Bad Company 2 was released (DICE, 2010) being the second game to be made with the newly created Frostbite engine (DICE, 2008), introducing a new level of destruction to the Battlefield franchise, this came to a head in Battlefield 3 (DICE, 2011) where buildings could crumble, walls breached and entire landscapes could be reshaped, further entries in the series built upon these mechanics further, setting the Battlefield series as a pioneer for interactive and immersive destructive gameplay.

In 2011 Breach (Atomic Games, 2011) was released where destruction took a more strategical approach where players could break through walls and floors to gain an advantage, adding new depth to the shooter genre. Building upon this in 2015 Tom Clancy's Rainbow Six Siege (Ubisoft Montreal, 2015) released, where like Breach walls can be broken to create new pathways and gain a tactical advantage.

2019's Crackdown 3 (Sumo Digital, 2019), implemented a fully dynamic destruction, system much like 2009's Red Faction Guerrilla, where objects and materials affect structural integrity, but Crackdown 3's physics calculations were so complex that they were implemented on cloud servers (Corden, 2018) to ensure that the game would run smoothly whilst still having highly detailed destruction simulations running.

Moving to 2020, Medieval Engineers (Keen Software House, 2020) released, a sandbox survival game that lets players build elaborate castles and machines, and then destroy them with catapults, trebuchets, and explosions, this game also implements a fully dynamic destruction system but unlike Crackdown 3 this game did not implement server-side simulation.

One of the most influential games to feature fully dynamic destructible environments is Teardown (Tuxedo Labs, 2022), a game whose entire focus is on destruction on an unparalleled scale. Teardown is a voxel-based game that lets the player do whatever they can think of, but unlike some of the other dynamic destruction simulations, Teardown does not feature a stress simulation or anything of the sort meaning that buildings can be supported by 1 small voxel-taking away from the realism.

Finally, in 2023 The Finals (Embark Studios, 2023) released, a battle royale shooter that incorporates a fully destructible environment as one of the core gameplay features, allowing players to strategically break walls or knock down entire buildings.

From Rampage in 1986, to modern-day games offering dynamic destruction on an immense scale. The gaming industry has seen a transformation in destructive capabilities. Titles like Red Faction, Battlefield, and Teardown have showcased advancements in graphics and physics, providing players with diverse ways to engage with the environment. Games such as Detonate, Breach, and Rainbow Six Siege showed that destruction can be implemented strategically to enhance the gameplay experience. Releases like Crackdown 3 further expand on this showing the possibility to offload complex physics calculations for seamless gameplay. Recent releases like Instruments of Destruction (Radian Games, 2022), ABRISS (Randwerk, 2023), and The Finals continue to embrace destruction as a core gameplay feature, showcasing its enduring appeal and innovation in the gaming world.

3. Project Specification

3.1. Aims

The following aims of this project will ensure that thorough research is carried out, looking at all the elements of RTDD and that a detailed project design is developed that will act as a guide when producing the destruction system. These ensure that a robust system is produced, that will allow for the dynamic destruction of objects in real time. this will be done with vigorous testing and user feedback to ensure that the final produced system adheres to the set goals and requirements:

- Investigate the trade-offs of current methods for fracturing complex rigid bodies in real-time and applying stress propagation.
- Design a system that allows for the real-time fracturing of complex Rigidbodies and utilizes stress propagation to make it dynamic.

- Produce a system that will allow for the real-time fracturing of complex rigid bodies in real-time, that also utilizes stress propagation to determine the structural integrity of the system.
- Evaluate the produced system by performing extensive benchmarking as well as analyzing user feedback.

3.2. Objectives

The following objectives will ensure that all the Aims of the project are achieved, to a high standard. These specific objectives were chosen to break down the core elements of each of the aims of the project into more manageable tasks that can be completed allowing for better tracking of the completion of the research.

- Review existing literature and methodologies for real-time fracturing.
- Identify common principles and best practices for stress propagation simulations.
- Develop a conceptual design for a system that enables real-time fracturing of complex rigid bodies, integrating stress propagation.
- Implement the designed RTDD system.
- Optimize the system for performance and efficiency, considering computational resources and real-time constraints.
- Conduct a comparative analysis of the produced systems against existing methods in terms of performance, accuracy, and computational efficiency.
- Collect user feedback through surveys to assess the usability and user satisfaction of the systems.
- Analyze and interpret results to identify areas for improvement and refinement in the developed systems.

3.3. Requirements

3.3.1. Functional

The following are the functional requirements of the planned RTDD system, that will ensure that the final produced system will adhere to the goals of the project and function as intended.

- rigid body fracture.
- Propagate stress through structures.
- user interaction, allows the user to trigger destruction events.
- particle effects, showing smoke and debris.

- Optimize the system to adhere to the real-time constraint.
- customizable parameters, to allow for custom experiences.

3.3.2. Non-Functional

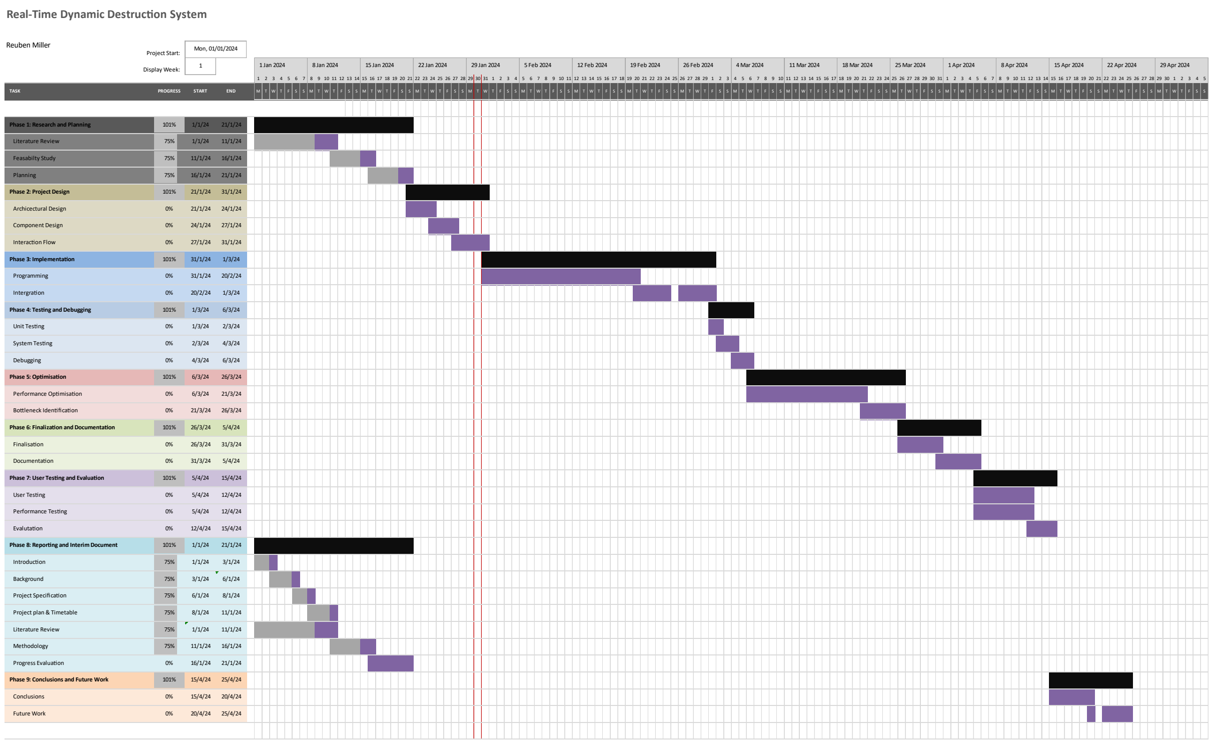
The following nonfunctional requirements of the system will ensure that the produced project will adhere to the industry standards for RTDD, and computer games. These requirements will also ensure that the produced system will adhere to the industry standards for codebases, allowing for smooth integration with other systems.

- achieve a minimum frame rate of 30 FPS.
- The system can handle varying levels of complexity and scale.
- maintain low latency, aiming for a maximum delay of 50ms between trigger and visible destruction.
- provide comprehensive documentation through code comments.
- design the system with modularity in mind, allowing easy integration with other systems.
- Robust error handling, to prevent crashing.
- use version control to track changes and maintain a stable codebase.

Whilst this research aims to make a substantial contribution to the understanding and implementation of RTDD, there are certain limitations of the research. The first limitation is that this research will not address all hardware configurations, limiting the universality of the proposed system, as well as this the proposed system will be implemented with the Unity game engine, further limiting the universality of the system.

With the extreme pace of technological advancements in the gaming industry (Christofferson et, al., 2022), certain findings might become outdated over time, so it is important to mention this research is conducted in the context of existing technology and methodologies. New and emerging technologies may present new opportunities or challenges not covered in this research. Despite having these limitations, this research aims to provide valuable insights and advancements within the defined scope, contributing to the ongoing discourse in and around RTDD.

4. Project Plan and Timetable


Real-Time Dynamic Destruction System

Reuben Miller

Project Start: Mon, 01/01/2024

Display Week: 1

TASK	PROGRESS	START	END
Phase 1: Research and Planning	101%	1/1/24	21/1/24
Literature Review	75%	1/1/24	11/1/24
Feasabilty Study	75%	11/1/24	16/1/24
Planning	75%	16/1/24	21/1/24
Phase 2: Project Design	101%	21/1/24	31/1/24
Archicectural Design	0%	21/1/24	24/1/24
Component Design	0%	24/1/24	27/1/24
Interaction Flow	0%	27/1/24	31/1/24
Phase 3: Implementation	101%	31/1/24	1/3/24
Programming	0%	31/1/24	20/2/24
Intergration	0%	20/2/24	1/3/24
Phase 4: Testing and Debugging	101%	1/3/24	6/3/24
Unit Testing	0%	1/3/24	2/3/24
System Testing	0%	2/3/24	4/3/24
Debugging	0%	4/3/24	6/3/24
Phase 5: Optimisation	101%	6/3/24	26/3/24
Performance Optimisation	0%	6/3/24	21/3/24
Bottleneck Identification	0%	21/3/24	26/3/24
Phase 6: Finalization and Documentation	101%	26/3/24	5/4/24
Finalisation	0%	26/3/24	31/3/24
Documentation	0%	31/3/24	5/4/24
Phase 7: User Testing and Evaluation	101%	5/4/24	15/4/24
User Testing	0%	5/4/24	12/4/24
Performance Testing	0%	5/4/24	12/4/24
Evalutation	0%	12/4/24	15/4/24
Phase 8: Reporting and Interim Document	101%	1/1/24	21/1/24
Introduction	75%	1/1/24	3/1/24
Background	75%	3/1/24	6/1/24
Project Specification	75%	6/1/24	8/1/24
Project plan & Timetable	75%	8/1/24	11/1/24
Literature Review	75%	1/1/24	11/1/24
Methodology	75%	11/1/24	16/1/24
Progress Evaluation	0%	16/1/24	21/1/24
Phase 9: Conclusions and Future Work	101%	15/4/24	25/4/24
Conclusions	0%	15/4/24	20/4/24
Future Work	0%	20/4/24	25/4/24

Figure 1. Project Plan

4.1. Research and Planning

The first phase is Research and planning the foundations of the project will be made. This phase will have an in-depth literature review which will be used to inform methodology and design decisions. The feasibility of the chosen methods, technologies, and tools will also be analyzed to ensure that they are suitable. Finally, a detailed project plan broken into milestones containing breakdowns of all tasks will be produced.

4.2. Project Design

The design phase will translate research findings into a development plan that can be followed to create the RTDDS. This stage will involve developing a detailed overview of the architectural design of the system acting as an outline of the high-level structure of the RTDDS. This will then be broken down into individual roles, functionalities, and interactions. Finally, the flow of each of the components will be shown, to clearly show how each element will communicate with each other. This solid design framework will guide the implementation phase to ensure precision and accuracy.

4.3. Implementation

In this phase the design specifications, and detailed designs for each of the components of the RTDDS will be turned into executable code integrated into the Unity editor, ensuring seamless interaction between individual components. During this development version control will be implemented to manage and track changes efficiently. This phase acts as a key marking in the transition from design and research to execution, taking the project one step closer to a functional program.

4.4. Testing and Debugging

This phase involves thorough unit testing to evaluate the functionality of each of the individual components, ensuring that each of them functions as expected in isolation. extensive system testing will also be conducted to ensure that all components of the RTDDS work together in unison. Finally, all bugs and issues found will be fixed to ensure a stable and performant system.

4.5. Optimization

The main objective of this phase is to enhance the performance of the produced system. This involves implementing strategies to optimize the speed, responsiveness, and efficiency of the RTDDS. any bottlenecks in the program will also be identified and fixed in this phase to ensure a performant and reliable system.

4.6. Finalization and Documentation

This phase ensures that all components of the system are integrated seamlessly, and the system functions as intended. All relevant documentation will also be completed, via in-depth code comments. Following these phases' completion, the RTDDS will be ready for use, with documentation to ensure that all users can understand, implement, and maintain the RTDDS.

4.7. User Testing and Evaluation

The primary goal of this phase is to assess the produced system in a real-world context. User testing will be conducted, during which feedback will be gathered on the functionality, interface, and overall experience of implemented RTDDS. Following this, an analysis of the collected data and results from performance testing will be evaluated against predefined criteria, including realism, visual fidelity, and user satisfaction. This will ensure that the produced system aligns with user expectations and project requirements.

4.8. Reporting and Interim Document

This section will be completed in the first half of the project development and will contain the written sections from introduction to methodology, there will also be an evaluation of the current progress of the project, this phase acts as a critical checkpoint allowing for a review of the project's status.

4.9. Conclusions and Future Work

This phase involves summarizing the project's outcomes and the lessons learned whilst providing insights into the challenges faced during the research and development of the project. This phase also aims to discuss avenues for future research and improvements that could be made. This will make sure that the RTDDS project not only contributes to current research but that it also serves as a foundation for ongoing advancements in the field.

5. Literature Review

The demand for more immersive and realistic games has led to a surge in technological innovations in the games industry (George, 2020). RTDD stands out as a key force in pushing the boundaries of what can be done in real time. This literature review will analyze the current methods of dynamically fracturing Rigidbodies, and applying stress propagation, evaluating the successes and limitations of the current techniques. By focusing on the gaming perspective, this review aims to explore the complexities of RTDD whilst contributing valuable insights to game developers, researchers, and enthusiasts alike.

The development of an RTDD can rely on a variety of different frameworks and methodologies. PhysX by NVIDIA (NVIDIA, 2019) is a widely used physics engine that provides a robust framework for real-time simulation. It includes support for Rigid body and soft body simulations, enabling developers to simulate realistic interactions between objects. PhysX is also the physics engine of choice for many AAA titles, offering a wide range of tools that offer vast customizability for the game designer. However, PhysX is resource-intensive, and requires optimization for reliable performance, Unlike Havok (Havok, 2023), an industry-standard physics engine known for its versatility and efficiency, it facilitates the interaction of realistic physics interactions. Havok is also used in many high-profile games attesting to its reliability.

Unreal Engine (Epic Games, 2023) integrates NVIDIA PhysX and Apex Destruction tools, providing a powerful combination for creating dynamic destruction. Apex Destruction tools allow for easy implementation of destruction physics which have been optimized to work in real time. However, these tool's functionality is hidden inside of the engine, meaning that to use such tools developers are stuck using Unreal Engine.

When it comes to methods for creating an RTDD several methods can be used to model the fracturing of objects. Muguercia et al. (2014, pp. 86-100) differentiate between three main methods. Physically based methods, Geometry-based methods, and example-based methods.

5.1. Physically based methods

Physically based methods aim to replicate the real-world behavior of materials under stress, strain, and impact, these methods use numerical approximations to solve equations for the material and can be further broken down into mass-spring models, Finite Element Method (FEM), and Meshless methods, there are also other methods which do not fit into these three categories but still adhere to the physical principles for their simulation of the fracture process.

5.1.1. Finite Element Methods (FEM)

FEM breaks complex structures down into smaller, simpler elements, and then models the behavior of each element based on the physical properties of the material.

Parker et al. (2009) suggests a tetrahedral finite element method (FEM). Optimized for leveraging parallelization for performance gains. The fracture algorithm is based upon (O'Brien, 1999) which involves decomposing forces into tensile and compressive components and then using the separation tensor to identify potential fracture locations and locally re-meshing the area around the fracture to maintain mesh consistency, this method allows for the dynamic destruction of the object whilst being performant, however, there are some issues with memory coherence, and the system cannot handle a large number of objects whilst still adhering to the real-time constraint.

Another variation on the FEM method was suggested by Muller (2001) who suggested a hybrid approach where the object is represented by a 3-dimensional tetrahedral, with deformation and fracturing being based on applied force fields, the model employs Hooke's law of elasticity, stress tensors, and FEM to describe the deformations and stresses on the objects allowing for fracture modeling and plastic behavior. The model achieves real-time performance by optimizing force computation by precomputing the products of specific values. They also optimized the Jacobian calculations by using the Block-Gauss-Seidel method meaning they can simultaneously solve three equations for every vertex of the tetrahedral mesh. Muller et al (2004) suggested another variation on FEM, based on cubical elements of uniform size linked via shared vertices. The generation of the cube mesh is done by dividing the bounding box into cubes and generating elements for those intersecting or inside of the visual mesh. the system employs FEM for deformations allowing for dynamic simulation. The fracture algorithm utilizes stress tensors to identify and fracture the mesh when a stress threshold is exceeded, the solution also suggests a surface animation algorithm ensuring watertight mesh generation, this includes a unique approach to surface fracture where the mesh integrity is preserved by not cutting any triangles. However this method uses cubes of a single size for its cube mesh and as such has many redundant pieces, this could be further optimized by using cubes of various sizes. The authors also mention an issue where individual small features of the surface mesh are associated with the same cube element.

5.1.2. Mass-Spring Models

In Mass-Spring Models the object is represented as a collection of mass points interconnected by springs, with the primary goal of simulating the motion, deformation, and interactions of the object under external forces.

Mazarak et al (1999) introduce an interconnected voxel (mass-spring) representation for physically based Modeling of blast wave impacts on objects, and the object fracture is simulated through the breaking of links in between the connected voxel’s structure. To be able to compute the split fragments the entire scene is described with a connectivity graph, with the nodes being the voxels and the arcs being the connections between voxels. Martins et al. (2001) further expanded this to include foundation voxels, and the ability to replace voxels with particles to increase the performance of the solution making it real-time. The main drawback of these techniques is that the model is limited to a voxel representation.

Yan et al. (2023) proposed a fast and realistic fracture simulation algorithm based on the spring-mass model and aerodynamics theory. That involves 2 steps. preprocessing where an object is subdivided using a 3-dimensional Voronoi diagram. The mass of the fragments created in this step is then calculated based on volume, and a spring-mass system using peridynamics simulates the object's fracture, with each point being viewed as a spring node. This method is very efficient, this method shows how preprocessing can be utilized to achieve real-time simulation of advanced fracture techniques.

5.1.3. Meshless methods

Meshless methods operate directly on scattered particles that approximate the model, they also often use Radial Basis Functions (RBFs) to interpolate and approximate values at points within the model.

Chen et al. (2013) proposes a hybrid animation approach for computer graphics to simulate brittle material fracture using smoothed particle hydrodynamics (SPH) with linear elastic modeling which enables accurate fracture evaluation and propagation. The method introduces a novel shape representation scheme that reduces the cost of tracking surface cracks by using a tetrahedron-based surface model and adaptive sampling of fracture points. However, the crack surface does not contain much detail and the meshes used must be small, which would cause issues in games.

5.1.4. Other

Loeiz et al (2013) suggest a method that simulates the dynamic destruction of objects in three steps, fracture initiation where modal analysis estimates contact properties. In the next step, the fracture starts based on the stress of the material stress. Once the fracture is initiated, the fracture is spread through the material. Finally flood filling separates the mesh into regions, each representing a fragment.

Ning et al (2012) suggest a novel technique where the model of the object is discretized into small elements known as movable cellular automata (MCA). Each automaton is a sphere with a mass and size, organized in 3-Dimensonal space following a specific set of rules. The simulation occurs in discrete time steps with external forces activating the automata, causing movements, and updating of forces and displacement of the automata. The fracturing criteria are based on the displacement and equivalent stress of the automata. These factors are then used to determine whether connections break when critical loads are applied. This is specifically designed using CUDA, meaning that it will only run on NVIDIA GPU which for a game where each user has different hardware would cause many issues.

Sellan (2023) introduces a novel method for real-time object destruction by precomputing fracture modes, this approach involves iteratively solving sparsified eigenvalues to obtain the lowest energy modes, this has the effect of producing a fracture pattern that is more realistic, whilst still allowing for impact dependent fracture patterns, without needing a crack propagation simulation, making precompute fractures that are more realistic.

5.2. Geometry based (or procedural)

Geometry-based methods produce slightly less detailed but still visually plausible fractures without needing the physical properties of the material, making them less computationally demanding than Physical methods, they also allow more control over the design of the fracture unlike most Physical methods, however, stress propagation is not default within these methods, as they omit the physical properties of the material.

Mould (2005) suggests an algorithm using Voronoi diagrams based on path distance through a graph, the algorithm uses a 4-connected lattice with each node representing a texture pixel. Local control over the crack placement is obtained using an image-guided parameter to help determine the edge costs, as well as the width of the cracks, which varies based on their distance from the generation site, allowing for a more realistic tapering effect. Another method that implements Voronoi diagrams is suggested by Clothier et al (2016) who suggests a novel method, using a 3D Voronoi Subdivision Tree, they start by creating a tree structure from a convex object, if this object is concave, it undergoes Hierarchical Approximate Convex Decomposition (HACD) to transform it into multiple convex objects. The process involves random placement of points within the convex polyhedron, creating bisector planes, and recursively populating the tree until completion. The resulting structure adapts to impact scenarios, dynamically subdividing the object only, when necessary, thus optimizing computational resources and allowing the simulation to run in real-time.

Moving on from Voronoi diagrams, Martinet et al (2004) present a procedural method for simulating cracks and fractures in solid materials with an emphasis on control over the patterns, fragment sizes, and shapes. The approach uses a hybrid tree model, combining skeletal implicit surfaces and triangular meshes. The key steps of the algorithm are to define a 2D crack pattern in the graph editor, which is then mapped onto the object surface and used to generate cracks through Boolean operations. The resulting fractures demonstrate realistic outcomes whilst still maintaining computational efficiency. Another method that allows user control over the fracture simulation is suggested by Zhang et al (2018) which utilizes a distance transform algorithm to generate realistic batik-like crack patterns, the process involves locating starting points based on the distance transform and then propagating cracks with perturbations for a natural appearance. The algorithm also incorporates user-controlled parameters such as noise, amplitude, and filtering, allowing for the customization of crack characteristics.

Looking at approaches that involve a preprocessing step, Su et al (2009) suggest a novel approach to fracture simulation by suggesting a prescoring technique to efficiently generate fracture patterns in advance, prescoring involves dividing all the space into regions and scoring each region to determine how likely it is to fracture under different conditions. This creates a map of potential fractures that can be dynamically triggered in real-time.

Finally, for geometry-based techniques, Muller et al (2013) suggest a novel method for the dynamic destruction of complex objects where they use a compound mesh of convex primitives generated by their novel Volumetric Approximate Convex Decomposition technique. The main fracture algorithm has 5 main steps, they first align the fracture pattern, so it aligns with the impact location, they then compute the intersection of all the cells with all the convexes, they weld the connected convexes together forming compounds from the convexes in each cell, finally they detect disjointed islands and separate them into individual compounds. After the convex has been fractured it undergoes a Partial fracture which will isolate the fracture to the impact location, finally, they clip this against the visual mesh. One thing this paper brings up is the use of polygons instead of triangles as it reduces the complexity of the computations required, this allows for a real-time solution that allows for complex environments and models.

5.3. Example-based methods

Example-based methods rely on preexisting data or examples to guide the simulation in new scenarios, these are usually images, that represent a fracture pattern that should either be used as a guide to fracture the object or used to fracture similar patterns. Like geometry-based methods, example-based methods also don’t contain stress propagation.

Glondu et al (2012) introduce a novel method for simulating 3-dimensional fractures, the technique starts with user-defined crack patterns which are processed into statistical properties that then guide Bayesian optimization to refine the parameters for physically based fracture simulation.

5.4. Stress Propagation

Whilst physically based methods, simulate the stress being applied to an object, and so can simulate the stress propagation, Geometry based, and Example based methods do not, and as so separate stress propagation algorithms will have to be implemented to achieve the desired functionality.

5.4.1. Von Mises Stress

Von Mises Stress is a scalar quantity used in engineering to predict the failure of materials under complex loading conditions. It does this by condensing all three normal and sheer stresses of the material into one scalar value, distortion energy. (von Mises, 1913), (a b Hill, 1998), (Kazimi, 2001), (Ford, 1963).

5.4.2. Tresca Criterion

Unlink Von Mises Stress Tresca Criterion focuses solely on maximum shear stress, proposing that failure will occur when the largest shear stress value of the material reaches its maximum value, as such Tresca Criterion is computationally simpler than Von Mises but less accurate. (Ali, 2022), (Flom, 2013), (Wright, 2011), (Yu, 2022).

5.4.3. Principle Stress

The principal stress of a material is the maximum (Major Principal Stress) and minimum (Minor Principal Stress) stresses that occur in a material when subjected to complex loads before the material undergoes failure. These act as a fundamental concept in solid mechanics, but for more complex scenarios more advanced methods are often implemented. (Wang, 2018), (Lisle, 1987), (Mogi, 1967), (Gudehus, 2020).

6. Methodology

This section outlines the research and implementation's nature and any potential analysis with existing systems or methodologies. This section will also outline the rationale behind choosing this research design.

The system will be developed with C# inside of Unity 3D this was chosen, as it’s a popular game development tool with a large user base, as well as being full of the features that will be needed for the smooth development of the tool. This will ensure development will be streamlined with the easy integration of testing and performance analysis tools.

For the creation of the object fracturing system, this research proposes a design based upon the OpenFracture library (Greenheck, 2024), which is based upon (Sloan, 1993), (Lawson, 1977), (Cline & Renka, 1984) this will be used to pre-fracture the mesh, as well as fracture the mesh in real-time, this method will allow for the perfect balance between performance and realism. Alongside this base implementation, several performance enhancements will be made to further ensure that the developed system runs in real time.

Following the fracturing of the object, stress propagation will be applied to the remaining structure, this algorithm will cycle through each of the pieces of the object generated in the fracture step and calculate their principal stress, which will be based on the amount of connects an object has, to determine whether the object needs to start fracturing again which will start the previous step on the given object.

A comparative analysis will then be conducted to evaluate the performance and functionality of the developed RTDDS against existing systems used in the gaming industry. This analysis aims to identify strengths, weaknesses, and areas that need improvement, by benchmarking the efficiency, scalability, and visual fidelity of the created solution. As well as this user testing will be conducted, to collect structured qualitative feedback on the realism, responsiveness, and overall satisfaction of the destruction mechanics.

The experimental approach allows for a hands-on investigation of the challenges and opportunities associated with implementing RTDD. Also, objective comparisons of the RTDD Systems' performance can be made, informing future development and optimization efforts, and contributing to practical insights by giving a detailed account of the development of RTDD systems.

7. Project Design

Before implementing the destruction system, it is important to have a design that will act as a mold for the implementation, for this an extensive list of flowcharts & class diagrams (Appendix C & D respectively) have been produced that show not only the flow of the program, but also how each of the individual classes will be constructed.

These diagrams have been created to match the design within Section 6. These designs also account for demo/example scripts that show how to implement the system, these designs include one for a player-controlled character, as well as a ray and collision-based way to initiate fractures.

8. Implementation

All the code for the RTDDS will be created in a custom namespace “ReubenMiller” as well as have its own assembly definition, this is to adhere better to the Unity guidelines for creating a custom package, which this project aims to be. Within the namespace, there are several different subspaces, including “*. Fracture” and “*. Demo” These will be used to better implement modularity into the system. This means that all the fracturing and stress propagation code will go within “ReubenMiller.Fracture” and all the demo/example scripts will reside within “ReubenMiller.Fracture. Demo”.

8.1. Fracturing

Using the given designs, and methodology the first thing to be implemented is the fracturing of the object, the algorithms used for this are from Greenheck’s OpenFracture (2024) library as stated in Section 6.

8.1.1. Fracture

To implement the fracturing of the object the first step is to create a template object that will be used as a base for all the fragments that are going to be created, this is done by copying the mesh renderers, filters, colliders, and Rigidbodies attached to the original object.

Once this template has been created the main fracture can be run this works by first converting the base mesh into a custom MeshData class, which will allow for better manipulation of the data within the mesh. Following this we recursively slice (Section 8.1.2.) the mesh along an arbitrary fracture plane until the desired fragment count is achieved, finally, we need to convert the MeshData into a game object.

To do this there first needs to be a check for disconnected meshes, this means that one mesh could have 2 or more separate parts which is fine for systems that do not require realistic results but as the aim of this system is to be as realistic as possible we need to run a find disconnected meshes step that will loop through all of the vertices of the mesh and check to see if there are any disconnections, which if there is it will return them as a separate mesh. Once the disconnections have been separated into different meshes, each mesh is added to a new GameObject with the collider MeshFilter and Rigidbody copied over.

One important thing to note here is that when all the new meshes are created they all need new convex colliders to be generated, which is a very costly operation to perform, as such when testing a large bottleneck of the program was within this step, to fix this before creating the GameObject's for each mesh a collider creation step is run. This involves utilizing parallelization to bake multiple meshes at once massively speeding up the creation of the colliders.

8.1.2. Slicer

The slicer is how the fracture breaks down the mesh into separate segments, it works by splitting the given mesh data into two separate meshes, one above and one below the slice plane, this will then be used by the fracture (Section 8.1.1.) to create the 2 new meshes.

The first step for the slicer is to define the 2 new meshes that will be created, the top slice and bottom slice, after this is done it will loop through each of the vertexes of the given mesh and determine whether it is above or below the slice plane adding the vertex to the corresponding MeshData.

After sorting the vertexes, the triangles then have the same treatment, splitting all the triangles to either above or below the slice plane, however, if a triangle is intersection by the slice plane, then it is set to be processed later to split the triangle.

After the triangles that don't need splitting have been processed we can begin to split the triangles that need splitting this is done by first determining which type of split needs to happen, with two possible, points p1, and p2 above the plane and p3 below or the opposite with p1 and p2 below and p3 above, if the first is true then we know that the intersection with the plane would have formed a quad and as such we need to convert this into 2 triangles this is done by getting the plane intersection points and inserting a new edge. Then we can add these two new triangles to the above mesh the triangle created from p3 and the 2 intersection points with the plane to the bottom mesh. If the second is true, then the same thing occurs but is reversed. As well as this the line that is created from generating the new triangles that lie within the plane is added to the edge list to be used in the next step.

Finally, after all the vertexes and triangles have been moved to their respective mesh, a filling algorithm is run that will fill in the newly created holes from splitting the mesh, luckily as we split with a plane this only needs to be run once, as we can just copy the vertex/triangles and flip the normals for the opposite side. This stage is done by feeding the vertices and constraints (edges of the mesh that would have been generated in the previous step) into the constrained triangulator (Section 8.1.3.) which itself is based upon (Sloan, 1993).

8.1.3. Constrained Triangulator

The triangulator works by first being initialized with all the input points and constraints, which it promptly projects onto a 2D plane as the algorithm works within 2D. the overarching functionality of the triangulator is simply, to add the super triangle, normalize coordinates, triangulate the points, and then finally apply the constraints before returning all the created triangles. It's within this deceivingly simple set of steps that the complexities of this algorithm begin to be known.

Looking at the first step, the algorithm will insert a super triangle, this triangle is a large artificial triangle that completely encloses all the triangles and is added to be the initial element of the triangulation process to simplify the process. Following the addition of the super triangle all the 2D coordinates are normalized (scaled between 0 and 1) this allows for a faster and more stable processing.

The next step is to compute the triangulation, this works by looping through each point, which has been sorted according to an ordered grid (Section 8.1.4.), and finding the triangle in which the point lies, this is done by looping through all of the triangles and checking that the point is on the right side of each of the edges of the current triangle, which if it is means that the point is inside of the triangle so it is inserted into it, this will replace the current triangle with 3 new triangles, to ensure the newly created triangle is added correctly adjacency table will be updated as well as restoring the Delauney Triangulation. To restore the triangulation, it will check if the current triangle circumscribes the inserted point, and if it does it will swap the diagonals of the quadrilateral formed by the given triangle and the triangle adjacent to the edge that is opposite of the newly added point.

Next, the constraints are applied to the triangulation, this is done by looping through the edges of the triangulation that intersect a constraint edge and removing them, the removal of the edge can be a bit more complex as 2 triangles can share an intersection edge, as such we need to find these triangles (which form a quad) and then swap the diagonal between them (flipping it), adding the new edge to the list of created edges.

After this all the triangles that violate any constraints or share vertexes with the super triangle are discarded, this is done by searching through all the points and checking each of them to make sure they are adhering to all the constraints. The final list of triangles is then created from all the triangles that have not been discarded and are returned, having successfully filled the given face.

The original OpenFracture library (Greenheck, 2024) was split into two separate sections Triangulator and Constrained Triangulator, but for this implementation, the regular Triangulator would never be used alone the decision was made to merge the two scripts together, allowing for the same functionality but with a slightly refined code base.

8.1.4. Bin Sort

This code will sort objects based on their location within a grid. The grid in question has even rows ordered right to left while odd rows are ordered left to right making a snaking-like pattern, the code will then assign bin numbers to each of the given objects based on their position within the grid. To sort the items the algorithm uses a Counting sort that allows for partial sorting, by setting the desired area to sort.

8.2. Stress Propagation

Following the implementation of the fracture, the next thing to be implemented is the stress propagation, this will be broken down into 2 main steps a pre-processing step where the connections of the object are calculated and a runtime step that will propagate the stress throughout the object.

8.2.1. Calculate Connections

The calculate connections step is run as a pre-processing step to save on performance in run time, this step will calculate which objects are touching, meaning that they are connected. It does this by looping through all the objects that need to be connected and for each one, it will get the triangles and verts for that object. It will then loop through all the triangles of the object and calculate the center of the triangle, after doing this it will loop through all the other children and find the closest point on the collider of the object, if the distance from this point to the center of the current triangle is small enough (close to 0) then the connection will be made.

8.2.2. Propagate Stress

The propagate stress algorithm is only run on the first frame, and when an object is destroyed, this is to ensure that the algorithm is not running when it does not need to, this also improves the performance of the system. When an object is destroyed the first part of the stress propagation is to propagate the stress among the object, this involves looping through all of the connections of the piece that is to be destroyed, and checking the forces that are currently being applied to them, if any of these forces are now too much, because of the removal of the piece, then that object is set to be destroyed.

The next step of the stress propagation system is to check for floating clusters (groups of objects) that can be formed when an object is destroyed. This algorithm will loop through all the connections of the object to be destroyed and check if they are connected to a root object, which if they are not means the object needs to fall as it is a floating cluster. To find the cluster a separate cluster finder algorithm is run that will take in a connection and perform a depth-first search to get all the other connections, these techniques work as the list of connections between each of the fragments acts as a multi-directional graph that connects all of them, so by treating them as nodes, they can be quickly searched.

An important thing to note is that will the two above functions there is both a regular and an Async version of them, the async version allows the resource-intensive task to be completed across multiple frames significantly reducing the effect on performance.

8.3. Destruction

The destruction system is the system that ties all the previous methods together, with the Pre-Fracture utilizing fracture and calculate connections scripts, and the Runtime Fracture utilizing the fracture and stress propagation scripts. These allow all these methods to be used within the game engine.

8.3.1. Pre-Fracture

The Pre-Fracture script will take in all of the information for not only the pre-fracturing of the object but also the information for the fracturing of the created fragments, this is so that they don’t have to be set individually after fracturing, to make this script function there is an accompanying editor script that will add a button to the editor that will fracture the object when clicked.

When the fracture button is clicked the script will create a new parent object to hold all the children as well as a template object to base all the fragments off, after this it will call the main fracture function, passing over all the required information. Once the fracture has been completed it will add the stress propagation scripts to the parent object as well as set all the fracture properties of the created fragments, finally, it will calculate all the connections between all the fragments that have been created and disable itself, so the new object containing the fractured version of the object can be seen.

8.3.2. Runtime Fracture

The Runtime Fracture script will manage fracturing and propagating the stress of all the fragments created from the pre-fracture (Section 8.3.1.). this script will be activated using a mixture of ray cast and collision (Section 8.4.3. This is to allow for pre-processing time, meaning that the fracture and stress propagation happens before the bullet from the player collides with the object, this massively improves performance, as the costly processing can be split among multiple frames.

For the actual functionality, it is much the same as Pre-Fracture (section 8.3.1.) until after the fracture has been completed where in the runtime fracture it will propagate the stress of the object (Section 8.2.) to check if any surrounding objects will be affected by the destruction of this object. Another important distinction between the two is that instead of connecting and setting up the created fragments, this will add the fade destroy script to them, which will make them shirk and be destroyed, this is to make sure the world does not become too cluttered with object as well as helping with performance.

Another important differentiation between this and Pre-Fracture is that for each of the operations that need completing in this step, there is an async variant so this script has to account for that, this means having a lot of branching statements, to ensure that all of the actions are performed in the correct order, before testing they were one after another causing numerous errors but once adding Action callbacks we can perform operations in the correct order at the cost of code readability.

8.4. Example Scripts

For ease of use and easy demonstrations, a group of demo/example scripts has been implemented that show how to interact with the developed systems, as well as to traverse demo content.

8.4.1. Player

The player script implements basic Rigidbody-based movement, along with Cinemachine camera controls. This script allows the user to move around and interact with the world via shooting. All the elements of the player movement are also customizable to allow for easy adaption to other games. As well as this a top-down version of the player was created, that instead of moving the camera will fire in the direction that the player pressed on the screen.

8.4.2. Fracture Initiation

Because of the pre-processing time, there is a specific method that must be used to start fracture initiation, this is first by utilizing the ray cast activation script to fire a ray in a given direction when hitting an object within the “destructible” layer of the ray will start the pre-processing steps, then at the same time as the ray, a bullet is fired with a trigger collider, that when this hits the object (tagged “destructible”) it will apply the destruction that has been performed. Although it might be inconvenient to have to implement the interactions like this it ensures a performant experience as without the pre-processing time it is very difficult to perform run-time fractures on anything but simple meshes.

8.4.3. UI

This script allows the traversal of different scenes to show off the different simulations that can be performed with the produced system. It also allows the changing of settings of the simulation, pausing of the game, and togging of the visibility of the UI.

9. Testing and Evaluation

Following the successful implementation of the game extensive testing has been completed, these tests cover all aspects of the implementation touching on the functionality, performance, and user feedback. This testing strategy was chosen as it covers all aspects of the project and will allow for better identification of whether the project has met the required aims and objectives.

To ensure that all the code functions as expected extensive functionality and usability testing were performed on all aspects of the code, these tests can be seen in Appendix D. These tests revealed several issues with the code that were then resolved and retested to ensure that the issues were properly resolved. Following this testing, the project reached its final stage in development, benchmarking, and user testing.

9.1. Benchmarking

Along with the functionality and usability testing extensive benchmarking was also performed to determine if the project met the criteria of being "real-time" For this a set of tests where created that would test the fracturing of an object with 10 different pre-fracture sizes (10, 20, 30, 40, 40, 60, 70, 80, 90, 100) all with 5 different runtime fracture fragments sizes (5, 10, 15, 20, 25). The compressed raw test data can be seen in Appendix E. This test data was gathered using a unity profiler along with a profile analyzer, each data point is from a 300-frame snapshot taken from when the fracture occurred, this raw data has been omitted for clarity.

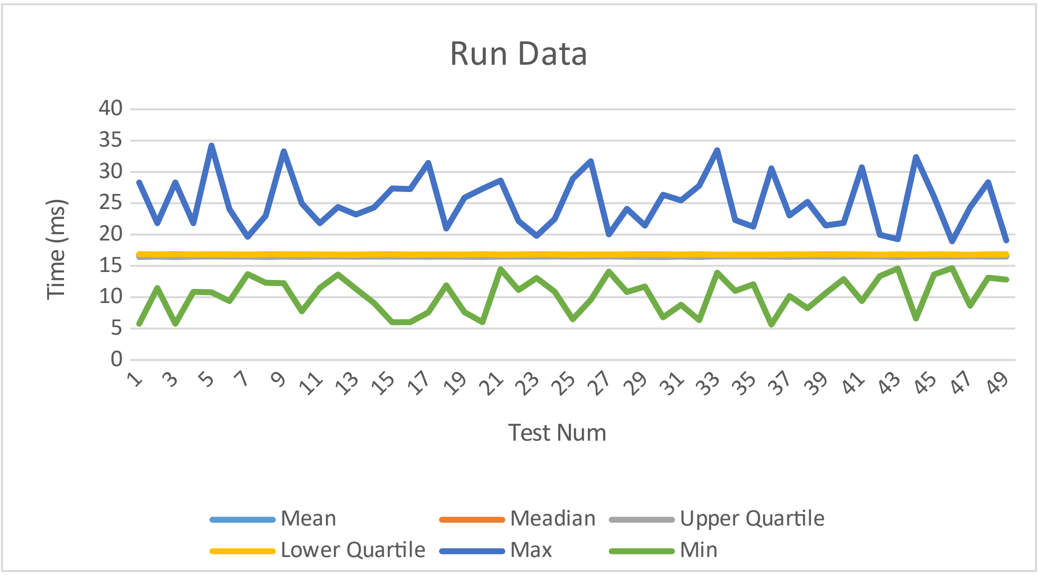


Figure 2. benchmarking data

Figure 1. shows the plot of the compressed benchmarking data, as can be seen from the chart none of the fracture events surpass 35ms in processing time, with only 2 points surpassing 33.3ms (time requirement for 30FPS) that being point 5 at 34.1941 and point 33 at

33.4485 even though these points surpass the 33.3 its is only by less than 0.9ms making these points insignificant as they represent the max from all 300 frames from the given snapshot.

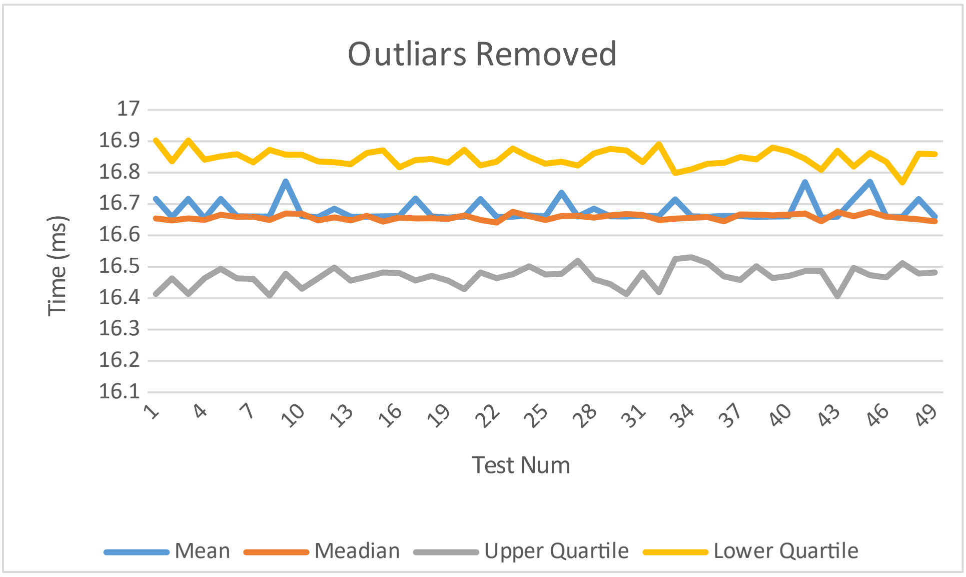


Figure 3. Benchmarking data with omitted outliers

When omitting these data points that are representative of the outliers of the data set, we can see that the project has an average of around 16.6ms (60FPS) with the possibility for a higher frame rate as the project has VSync enabled meaning that the project is locked to 60FPS. However, because of the spikes when performing the fracture events, the project performs at a stable 30FPS and a near-stable 60FPS.

9.2. User Testing

The testing pool consisted of games students, lecturers, and students with no correlation to games, this is to ensure that the results can be without bias and to ensure that the potential bias from people deeply intertwined with games (preconceived notions of computer game destruction) can be mitigated. to ensure accordance with the European Union’s General Data Protection Regulation (GDPR) all results from the survey will remain anonymous, with no personal data being recorded of any kind.

The questionnaire consisted of 7 questions, 4 multiple choice questions, and 3 open-ended. These questions were chosen, to best gather opinions about the produced system. The full list of responses can be found in Appendix F. The questions were as follows:

1. The destruction in the video felt realistic. (1-5)
2. The destruction happened smoothly without noticeable lag. (1-5)

3. The Destruction had a high level of detail. (1-5)
4. The overall experience of watching the destruction was engaging. (1-5)
5. Describe any instances where the destruction felt unrealistic or unnatural. (e.g., Buildings collapsing in an unexpected way) (open-ended)
6. What aspects of the destruction detail impressed you the most? (open-ended)
7. Were there any specific features you found missing that would enhance the realism? (open-ended)

To answer said questions the participants were asked to watch a 2-minute video of the destruction taking place, this was chosen as it is the easiest way to distribute the destruction system to a large range of participants. From the results it can be seen that the users did not seem to think the destruction in the video felt very realistic, with a score of 3.6 for question 1 on average but looking at question 5 we can see this is mostly down the size of the pre-fracture and the explosion force at which the particles get shot out, both of these things are easily changeable Within the system. As such it's reasonable to say with these minor changes the realism score would increase.

Looking at question 2 most of the participants gave a high score with an average of 4.6, this fits with the results found in the benchmarking that the solution runs at a stable frame rate. Looking at question 4 with an average of 4.08 it can be seen that the destruction provides an engaging experience, so whilst this version is lacking in realism, as also seen by an average score of 3.75 for question 3. It would seem the users are happy with the look of the destruction.

This sentiment is further solidified but questions 6 and 7 with the consensus being that the level of detail, VFX, and performance were all very impressive, but the size of the chunks and the speed at which they fly off could be improved, which as stated before would be a simple to change to make.

10. Conclusion

Several techniques to fracture mesh and perform stress propagation were analyzed in the research for this project. Comparing this work to them, it can be seen as a performant solution to both the fracturing of meshes and stress propagation, with most of the techniques analyzed, either having both and running at non-real-time or not having the desired features.

One of the key contributions of this work is the optimizations put into place to ensure that the system functions within the real-time constraint, this was achieved by utilizing multiple tricks available in video games, one major trick used is pre-processing the fracture before it occurs so it can be quickly swapped out, this along with techniques to quickly generate all of the colliders for the meshes created, ensures that the solution runs in real time, another technique that was used was the utilization of async operations to be able to split the running of the complex algorithms among several frames reducing the total impact it has on performance. With the project produced running at 60PFS, dropping to 30FPS at the time of fracture proving that a dynamic technique can be implemented to run in real-time, allowing for higher visual fidelity whilst not breaking the performance.

The main limitation of the implemented technique is within the limitations of the fracture algorithm used, as it requires convex, non-self-intersecting models otherwise it will generate the fractures with large visual errors that warrant it unusable. Another limitation is the lack of localized fracturing, this means that a fracture is applied to the entire object, thus needing the addition of the pre-processing step to properly fracture the object, whilst not a massive issue, techniques such as Müller's (2013) Real-time dynamic fracture with volumetric approximate convex decompositions implements systems utilizing visual and hidden meshes that show much promise, with high performance and localized fracture events.

11. Future Work

In the future, several expansions on the fracture algorithm could be made to allow it to have support for self-intersecting and open meshes, this would allow for a better user experience as users could just grab any mesh and it would work. As well as implementing a localized fracture technique that implements a visual and action mesh along with constructive solid geometry to clip in the changes from the active mesh to the visual mesh would reduce the total number of objects needed and increase the performance of the system.

As well as these, a more extensive stress propagation system could also be implemented, that allows for custom material properties, meaning that different materials would break at different points, making the system based on the area of the connections could also help improve the realism of the system massively.

12. References

Cameron, T. (2009). History 101: Destructible Environments in Videogames [online]. Available from: http://www.gamernode.com/history-101-destructible-environments-in-videogames/ [Accessed 9 February 2024].

Unreal Engine. (2022). Unreal Engine 5 offers significant new potential for the simulation industry. [online]. Available from: https://www.unrealengine.com/en-US/blog/unreal-engine-5-offers-significant-new-potential-for-the-simulation-industry [Accessed 9 February 2024].

George, S. (2020). Games, Simulations, Immersive Environments, and Emerging Technologies. (eds) Encyclopedia of Education and Information Technologies [online]. Springer, Cham. Available from: https://doi.org/10.1007/978-3-030-10576-1_36 [Accessed 9 February 2024].

Butler, S. (2022). 30 FPS Games Are Here to Stay. Here's Why. How To Geek[online]. Available from: https://www.howtogeek.com/841748/30-fps-games-are-here-to-stay-heres-why/ [Accessed 9 February 2024].

Buday, R., Baranowski, T., Thompson, D. (2012). Fun and Games and Boredom. Games for Health Journal [online]. Available from: http://doi.org/10.1089/g4h.2012.0026 [Accessed 9 February 2024].

Steele, A. (2023) Immersion in Gaming and Entertainment: Creating Engaging Virtual Worlds. Medium [online]. Available from: https://medium.com/@amysteele1999/immersion-in-gaming-and-entertainment-creating-engaging-virtual-worlds-1ad55e908805 [Accessed 9 February 2024].

Simply Put Psych. (2023). From Destruction to Creation: Exploring the Psychological Dynamics in Videogames. Gaming Psych [online]. Available from: https://simplyputpsych.co.uk/gaming-psych/from-destruction-to-creation-exploring-the-psychological-dynamics-in-videogames [Accessed 9 February 2024].

Bally Midway. (1986). Rampage [Computer Game]. Bally Midway. Available from: https://archive.org/details/arcade_rampage [Accessed 9 February 2024].

Eclipse Software Design (1994). Iron Solider. [Computer Game] Atari Corporation. Available from: https://www.myabandonware.com/game/iron-soldier-hr3 [Accessed 09 February 2024].

Volition. (2001) Red Faction. [Computer Game] THQ. Available from: https://store.steampowered.com/app/20530/Red_Faction/ [Accessed 9 February 2024].

Fandom. (2024). Geo-Mod. Red Faction Wiki. [online]. Available from: https://redfaction.fandom.com/wiki/Geo-Mod [Accessed 9 February 2024].

Volition. (2009). Red Faction: Guerrilla. [Computer Game] THQ. Available from: https://store.steampowered.com/app/667720/Red_Faction_Guerrilla_ReMarstered/ [Accessed 9 February 2024].

DICE. (2010). Battlefield: Bad Company 2. [Computer Game]. Electronic Arts. Available from: https://store.steampowered.com/app/24960/Battlefield_Bad_Company_2/ [Accessed 9 February 2024].

DICE. (2008). Frostbite [Computer Program]. Available from: https://www.ea.com/frostbite [Accessed 9 February 2024].

DICE. (2011). Battlefield 3. [Computer Game]. Electronic Arts. Available from: https://www.ea.com/en-gb/games/battlefield/battlefield-3 [Accessed 9 February 2024].

Atomic Games. (2011). Breach. [Computer Game]. Available from: https://archive.org/details/breach_202303 [Accessed 9 February 2024].

Ubisoft Montreal. (2015). Tom Clancy's Rainbow Six Siege. [Computer Game] Ubisoft. Available from: https://www.ubisoft.com/en-gb/game/rainbow-six/siege [Accessed 9 February 2024].

Sumo Digital. (2019). Crackdown 3. [Computer Game]. Microsoft Studios. Available from: https://www.crackdown.com [Accessed 9 February 2024].

Corden, J. (2018). Microsoft Studios' Brian Stone on how Crackdown 3 leverages the cloud. Windows Central [online]. Available from: https://www.windowscentral.com/inside-crackdown-3s-azure-cloud-powered-destruction [Accessed 9 February 2024].

Keen Software House. (2020). Medieval Engineers. [Computer Game] Keen Software House. Available from: https://www.medievalengineers.com [Accessed 9 February 2024].

Tuxedo Labs. (2022). Teardown. [Computer Game] Tuxedo Labs and Saber Interactive. Available from: https://www.teardowngame.com [Accessed 9 February 2024].

Embark Studios. (2023). The Finals. [Computer Game] Embark Studios. Available from: https://www.reachthefinals.com [Accessed 9 February 2024].

Radian Games. (2022). Instruments of Destruction. [Computer Game]. Radian Games. Available from: https://store.steampowered.com/app/1428100/Instruments_of_Destruction/ [Accessed 9 February 2024].

Randwerk. (2023). ABRISS - build to destroy. [Computer Game]. Astragon Entertainment. Available from: https://store.steampowered.com/app/1671480/ABRISS__build_to_destroy/ [Accessed 9 February 2024].

Christofferson, A., James, A., Obrien, A., Rowland, T. (2022). Level Up: The Future of Video Games is Bright. Bain & Company [Online]. Available from: https://www.bain.com/insights/level-up-the-future-of-video-games-is-bright/ [Accessed 9 February 2024].

NVIDIA. (2019). NVIDIA PhysX. [Computer Program] NVIDIA. Available from : https://www.nvidia.com/en-us/drivers/physx/physx-9-19-0218-driver/ [Accessed 9 February 2024].

Havok. (2023). Havok. [Computer Program]. Microsoft. Available from: https://www.havok.com/havok-physics/ [Accessed 9 February 2024].

Epic Games. (2023). Unreal Engine. [Computer Program]. Epic Games. Available from: https://www.unrealengine.com/en-US [Accessed 9 February 2024].

Muguercia, L., Bosch, C., & Patow, G.A. (2014). Fracture modeling in computer graphics. Comput. Graph., 45, pp. 86-100. Available from: https://doi.org/10.1016/j.cag.2014.08.006 [Accessed 9 February 2024].

Parker, E.G., & O'Brien, J.F. (2009). Real-time deformation and fracture in a game environment. Symposium on Computer Animation. Available from: https://doi.org/10.1145/1599470.1599492 [Accessed 9 February 2024].

O'Brien, J.F., & Hodgins, J.K. (1999). Graphical modeling and animation of brittle fracture. Proceedings of the 26th annual conference on Computer graphics and interactive techniques. Available from: https://doi.org/10.1145/311535.311550 [Accessed 9 February 2024].

Müller, M., McMillan, L., Dorsey, J., Jagnow, R. (2001). Real-Time Simulation of Deformation and Fracture of Stiff Materials. Computer Animation and Simulation 2001. Eurographics. Springer, Vienna. Available from: https://doi.org/10.1007/978-3-7091-6240-8_11 [Accessed 9 February 2024].

Muller, M., Teschner, M. and Gross, M. (2004) Physically-based simulation of objects represented by surface meshes. Proceedings Computer Graphics International, 2004. Crete, Greece. pp. 26-33. Available from: https://doi.org/10.1109/CGI.2004.1309189 [Accessed 9 February 2024].

Mazarak, O., Martins, C., & Amanatides, J. (1999). Animating Exploding Objects. Graphics Interface. Available from: http://www.cs.duke.edu/courses/cps124/spring08/assign/03_papers/explosion.pdf [Accessed 9 February 2024].

Martins, C., Buchanan, J.W., & Amanatides, J. (2001). Visually believable explosions in real-time. Proceedings Computer Animation 2001. Fourteenth Conference on Computer Animation (Cat. No.01TH8596). pp. 237-259. Available from: https://doi.org/10.1109/CA.2001.982398 [Accessed 9 February 2024].

Yan, M. and Wu, D. (2023). A New Fracture Simulation Algorithm Based on Peridynamics for Brittle Objects. IEEE Access. vol. 11. pp. 88609-88617. Available from: https://doi.org/10.1109/ACCESS.2023.3305631 [Accessed 9 February 2024].

Chen, F., Wang, C., Xie, B., Qin, H. (2013). Flexible and rapid animation of brittle fracture using the smoothed particle hydrodynamics formulation. Computer Animation and Virtual worlds. Volume 24, issue 3-4. pp.215-224. Available from: https://doi.org/10.1002/cav.1514 [Accessed 9 February 2024].

Glondu, L. Marchal, M. and Dumont, G. (2013) Real-Time Simulation of Brittle Fracture Using Modal Analysis. IEEE Transactions on Visualization and Computer Graphics. vol. 19, no. 2. pp. 201-209. Available from: https://doi.org/10.1109/TVCG.2012.121 [Accessed 9 February 2024].

Ning, J., Xu, H., Wu, B., Zeng, L., Li, S., Xiong, Y. (2012) Modeling and animation of fracture of heterogeneous materials based on CUDA. Vis Comput 29, 265–275 (2013). Available from: https://doi.org/10.1007/s00371-012-0765-1 [Accessed 9 February 2024].

Sellán, S., Luong, J., Mattos, L., Silva, D., Ramakrishnan, A., Yang, Y., Jacobson, A. (2023). Breaking Good: Fracture Modes for Realtime Destruction. ACM Trans. Graph. 42, Article 10. Available from: https://doi.org/10.1145/3549540 [Accessed 9 February 2024].

Mould, D. (2005). Image-guided fracture. Proceedings of Graphics Interface 2005 (GI '05). Canadian Human-Computer Communications Society, Waterloo. pp. 219–226. Available from: https://doi.org/10.1145/1089508.1089545 [Accessed 9 February 2024].

Clothier, M., and Bailey, M. (2016). 3D Voronoi Subdivision Tree for granular materials. 2015 6th International Conference on Computing. Communication and Networking Technologies. pp. 1-7. Available from: https://doi.org/10.1109/ICCCNT.2015.7395194 [Accessed 9 February 2024].

Martinet, A., Galin, E., Desbenoit, B., and Akkouche, S. (2004) Procedural modeling of cracks and fractures) Proceedings Shape Modeling Applications. Genova, Italy. pp. 346-349. Available from: https://doi.org/10.1109/SMI.2004.1314524 [Accessed 9 February 2024].

Zhang, J., Duan, F., Zhou, M., Jiang, D., Wang, X., Wu, Z., Huang, Y., Du, G., Liu, S., Zhou, P., Shang, X. (2018) Stable and realistic crack pattern generation using a cracking node method. Front. Comput. Sci. 12, pp.777–797. Available from: https://doi.org/10.1007/s11704-016-5511-9 [Accessed 9 February 2024].

Su, J., Schroeder, C., and Fedkiw, R. (2009). Energy stability and fracture for frame rate rigid body simulations. Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Association for Computing Machinery. New York, NY, USA. pp.155–164. Available from: https://doi.org/10.1145/1599470.1599491 [Accessed 9 February 2024].

Müller, M., Chentanez, N., & Kim, T. (2013). Real time dynamic fracture with volumetric approximate convex decompositions. ACM Transactions on Graphics. 32. pp.1 - 10. Available from: https://doi.org/10.1145/2461912.2461934 [Accessed 9 February 2024].

Glondu, L., Marchal, M., Dumont, G. (2012) Real-Time Simulation of Brittle Fracture Using Modal Analysis. IEEE Trans Vis Comput Graph. pp.201-9. Available from: https://doi.org/10.1109/tvcg.2012.121 [Accessed 9 February 2024].

Mises, R. (1913) Mechanik der festen Körper im plastisch- deformablen Zustand. Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-

Physikalische Klasse. pp.582-592. Available from: http://eudml.org/doc/58894 [Accessed 9 February 2024].

Hill, R. (1998) The Mathematical Theory Of Plasticity. Oxford Academic. https://doi.org/10.1093/oso/9780198503675.001.0001 [Accessed 9 February 2024].

Kazimi, S. M. A. (2001) Solid Mechanics. Tata McGraw-Hill.

Ford, H. (1963) Advanced Mechanics of Materials. University of California. Wiley.

Roostaei, A., Jahed, H. (2022) 2 - Fundamentals of cyclic plasticity models. Cyclic Plasticity of Metals. pp.23-51. Available from: https://doi.org/10.1016/B978-0-12-819293-1.00011-5 [Accessed 9 February 2024].

Flom, Y. (2013). 2 - Strength and margins of brazed joints. Advances in Brazing. Woodhead Publishing. pp.31-54. Available from: https://doi.org/10.1533/9780857096500.1.31 [Accessed 9 February 2024].

Wright, R. (2011). Chapter Eleven - Mechanical Properties of Wire and Related Testing. Wire Technology. pp.127-155. Available from: https://doi.org/10.1016/B978-0-12-382092-1.00011-7 [Accessed 9 February 2024].

Yu, T., Xue, P. (2022). Chapter 5 - Plastic constitutive equations. Introduction to engineering plasticity. Elsevier. pp.89-122. Available from: https://doi.org/10.1016/B978-0-323-98981-7.00005-1 [Accessed 9 February 2024].

Wang Z, Liu P, Chan A. (2018) A state-dependent soil model and its application to principal stress rotation simulations. International Journal of Distributed Sensor Networks. Available from: https://doi.org/10.1177/1550147718808751 [Accessed 9 February 2024].

Lisle, R. (1987) Principal stress orientations from faults: An additional constraint [online]. Available from: https://www.researchgate.net/publication/285196485_Principal_stress_orientations_from_faults_An_additional_constraint [Accessed 9 February 2024].

Mogi, K. (1967). Effect of the intermediate principal stress on rock failure. Advancing each and space sciences. Available from: https://doi.org/10.1029/JZ072i020p05117 [Accessed 9 February 2024].

Gudehus, G. (2021) Implications of the principle of effective stress. Acta Geotech. 16.Available from: https://doi.org/10.1007/s11440-020-01068-7 [Accessed 9 February 2024].

Sloan, S. W. (1993) A fast algorithm for generating constrained Delaunay triangulations. Computers & Structures 47.3. pp.441-450.

Lawson, C. L. (1977) Software for C1 surface interpolation. Mathematical software. Academic Press. pp.161-194.

Cline, A. K., Renka, R. L. (1984) A storage-efficient method for construction of a Thiessen triangulation. The Rocky Mountain Journal of Mathematics (1984). pp.119-139.

Greenheck, D. (2024). OpenFracture [Computer Program]. Available from: https://github.com/dgreenheck/OpenFracture [Accessed May 2024].

13. Appendices

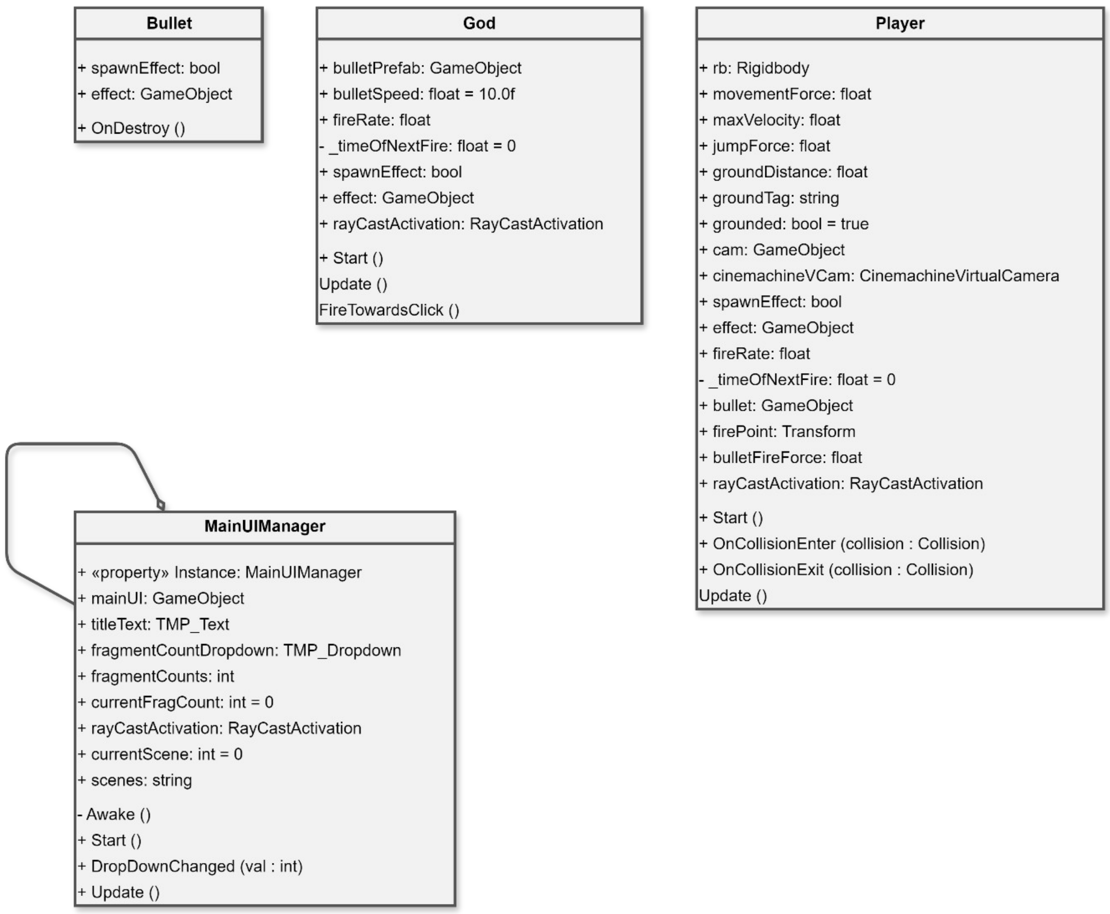
Appendix A

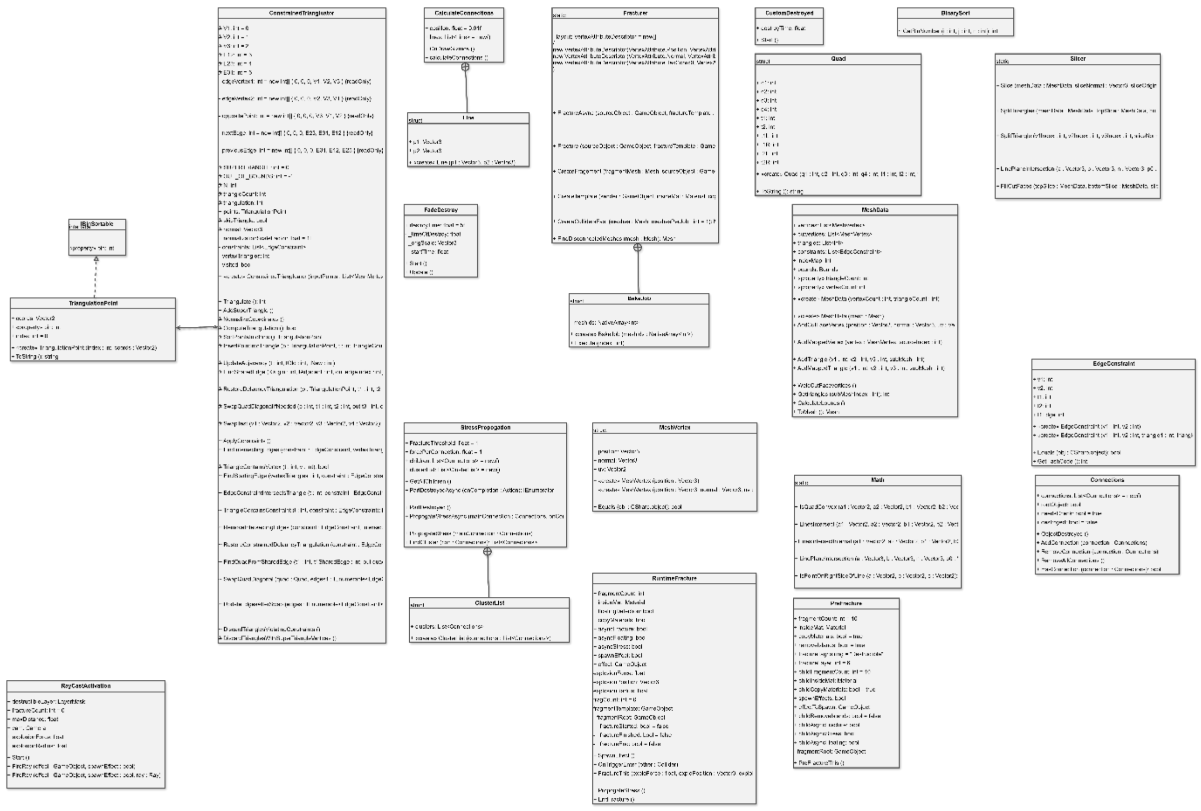
Example Games that use proprietary software

Games such as Teardown, Red Faction, and Battlefield. As well as software like, RayFire and PullDownIt all use proprietary systems to implement dynamic destruction.

Appendix B

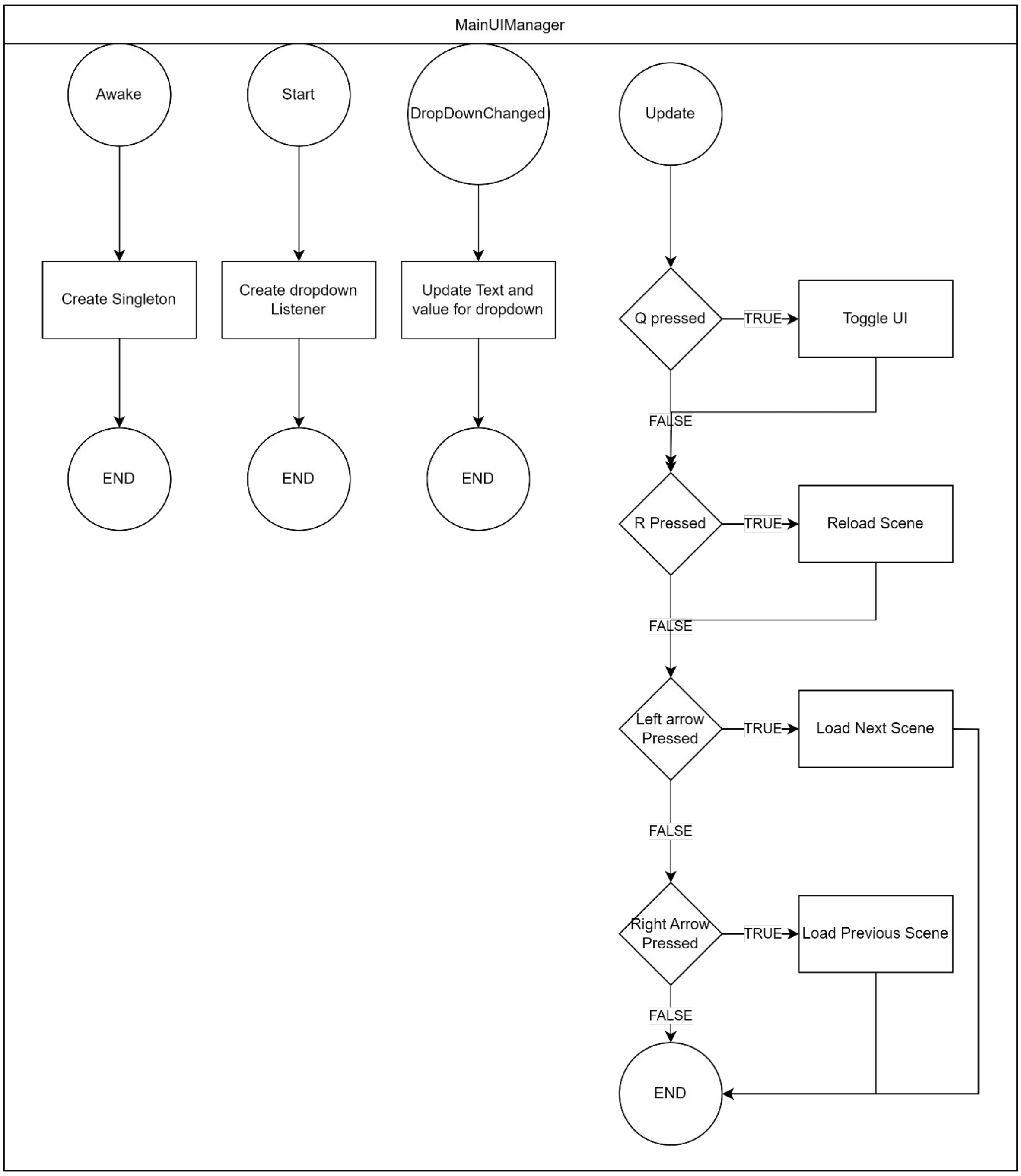
Class Diagrams

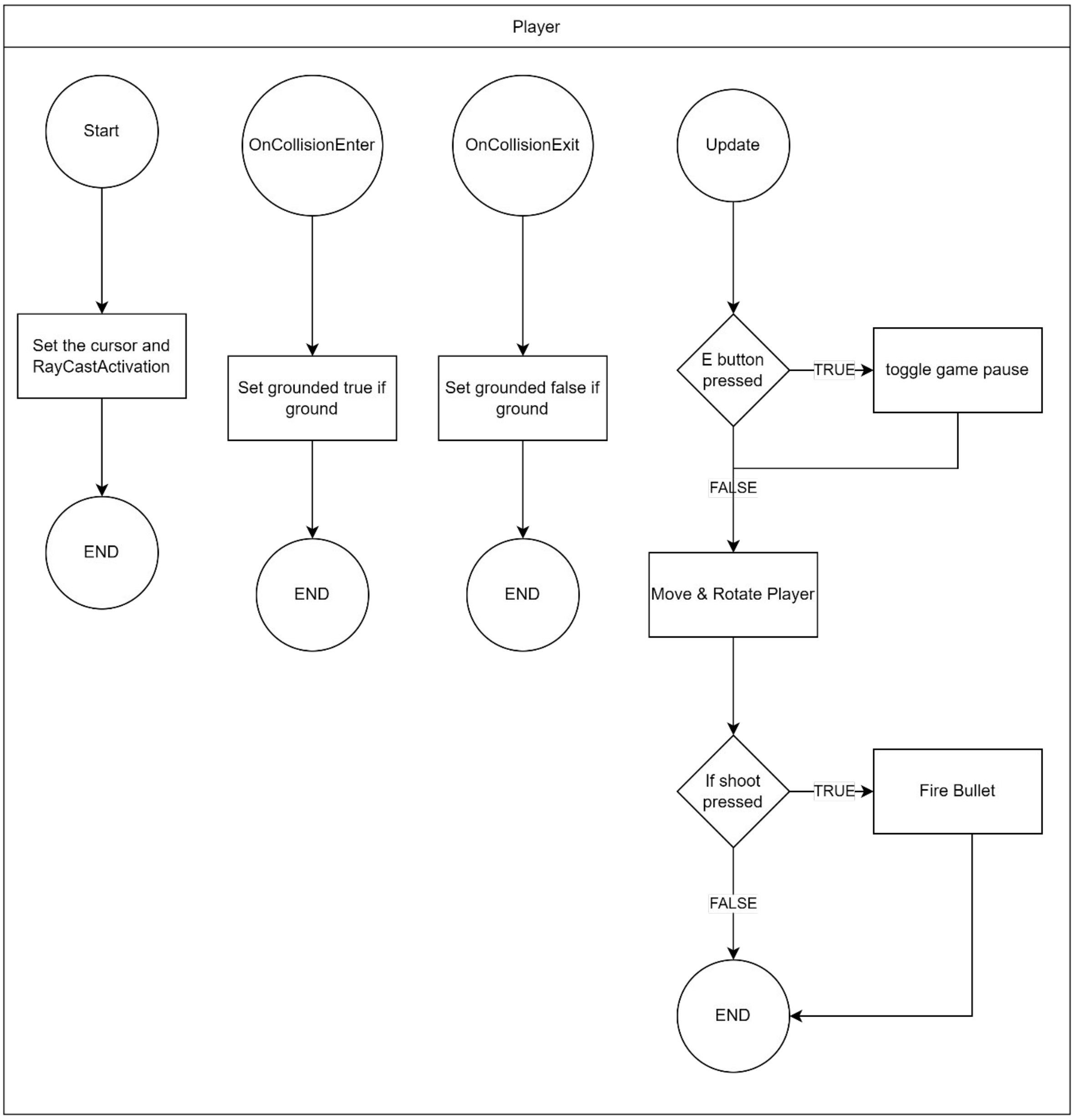


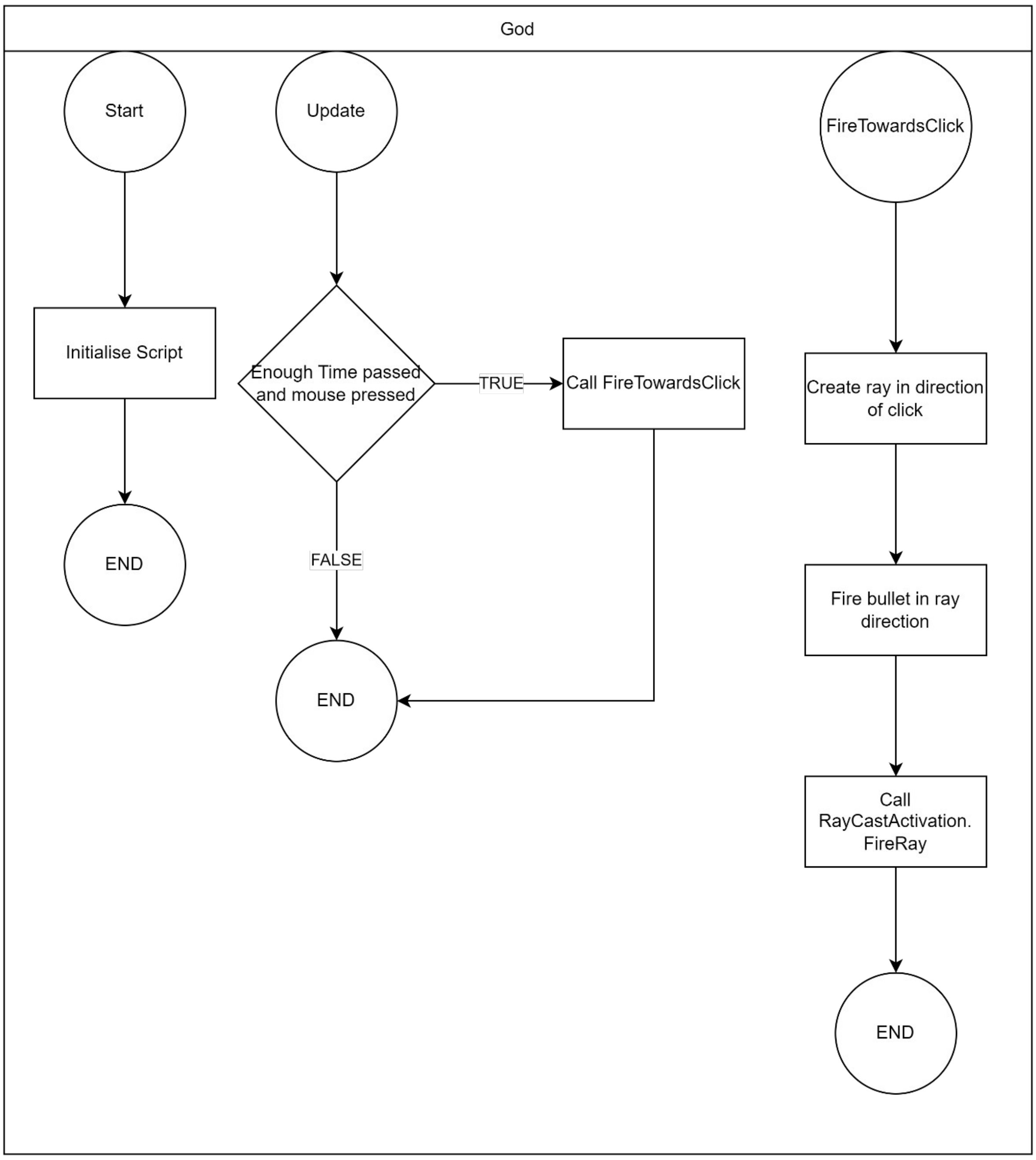

ConstrainedTriangulator
CalculateConnections
Fracturer
CustomDestroyed
BinarySort
Quad
Slicer
IBinSortable
TriangulationPoint
Line
FadeDestroy
BakeJob
MeshData
EdgeConstraint
StressPropagation
MeshVertex
Math
Connections
ClusterList
RuntimeFracture
PreFracture
RayCastActivation

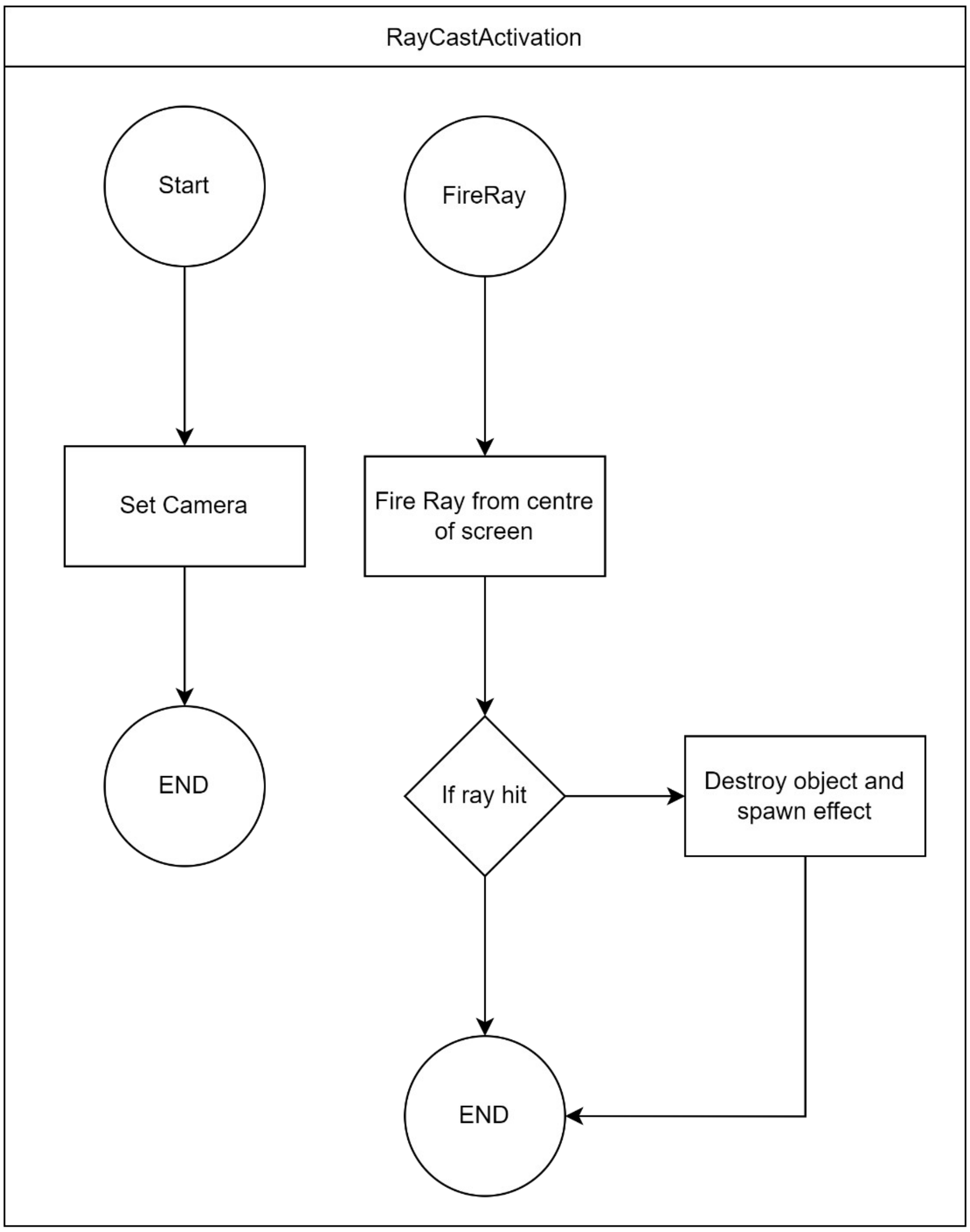
Appendix C

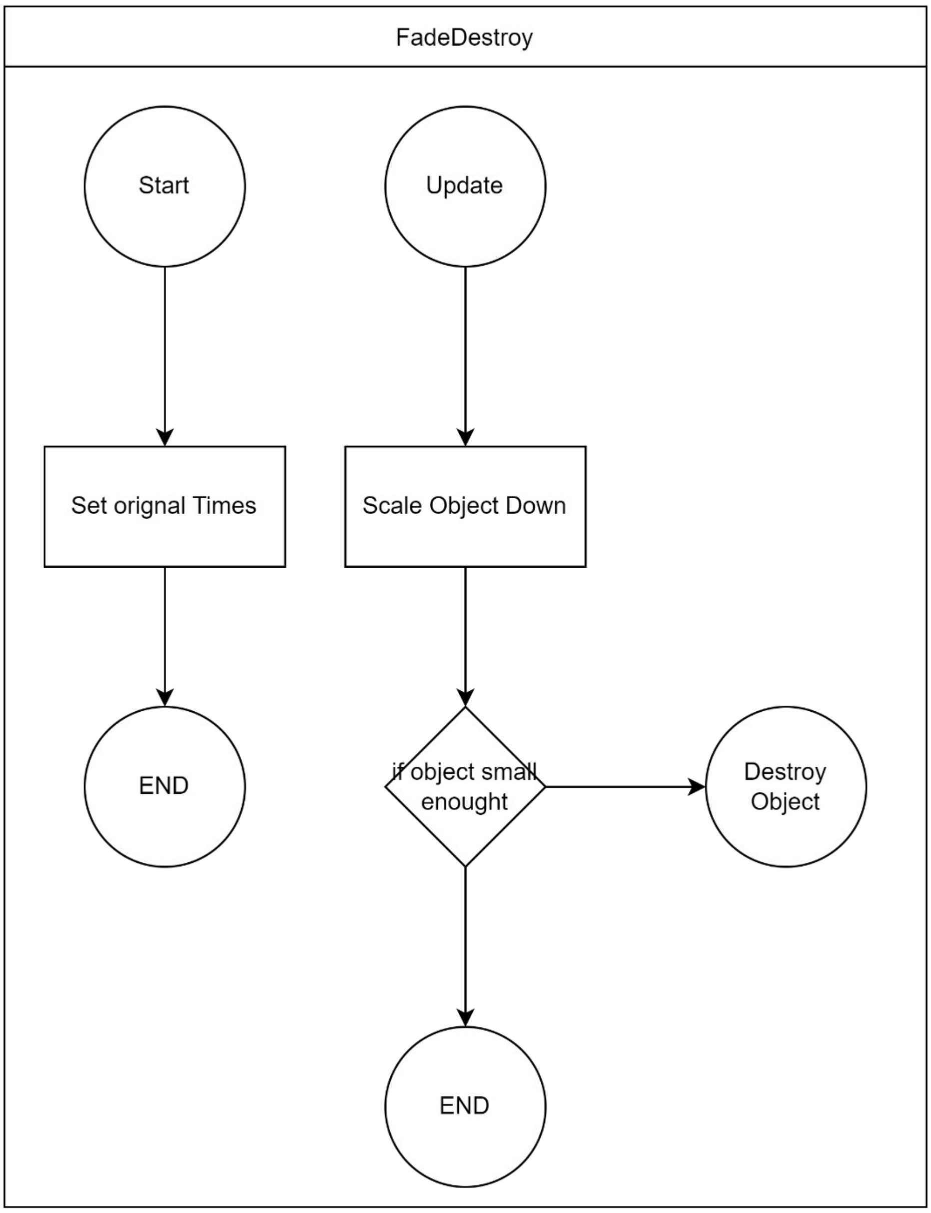
Flowcharts

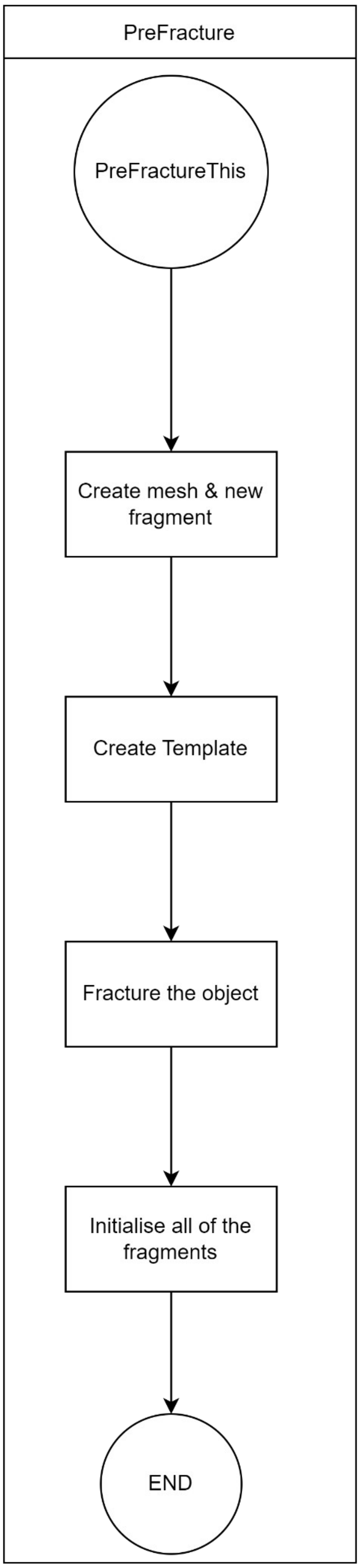


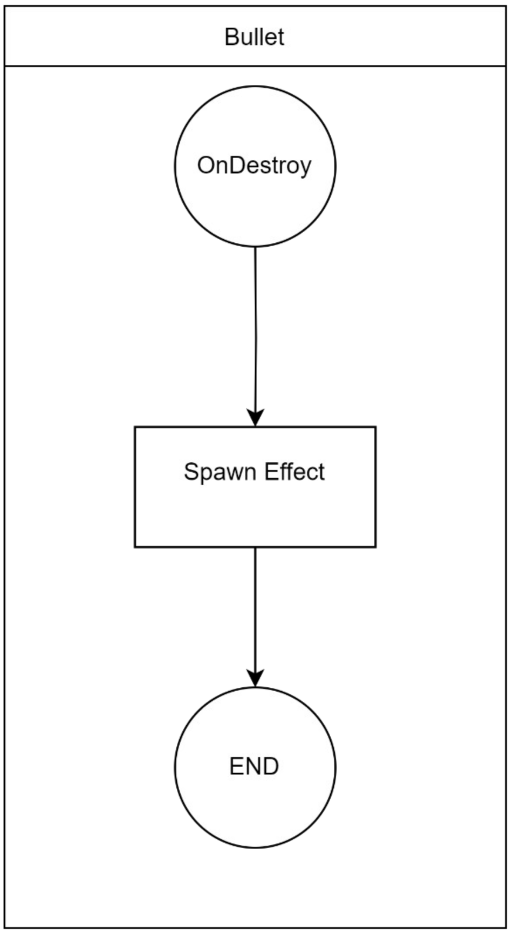

Player
Start
Set the cursor and RayCastActivation
END
OnCollisionEnter
Set grounded true if ground
END
OnCollisionExit
Set grounded false if ground
END
Update
E button pressed
TRUE
toggle game pause
FALSE
Move & Rotate Player
If shoot pressed
TRUE
Fire Bullet
FALSE
END

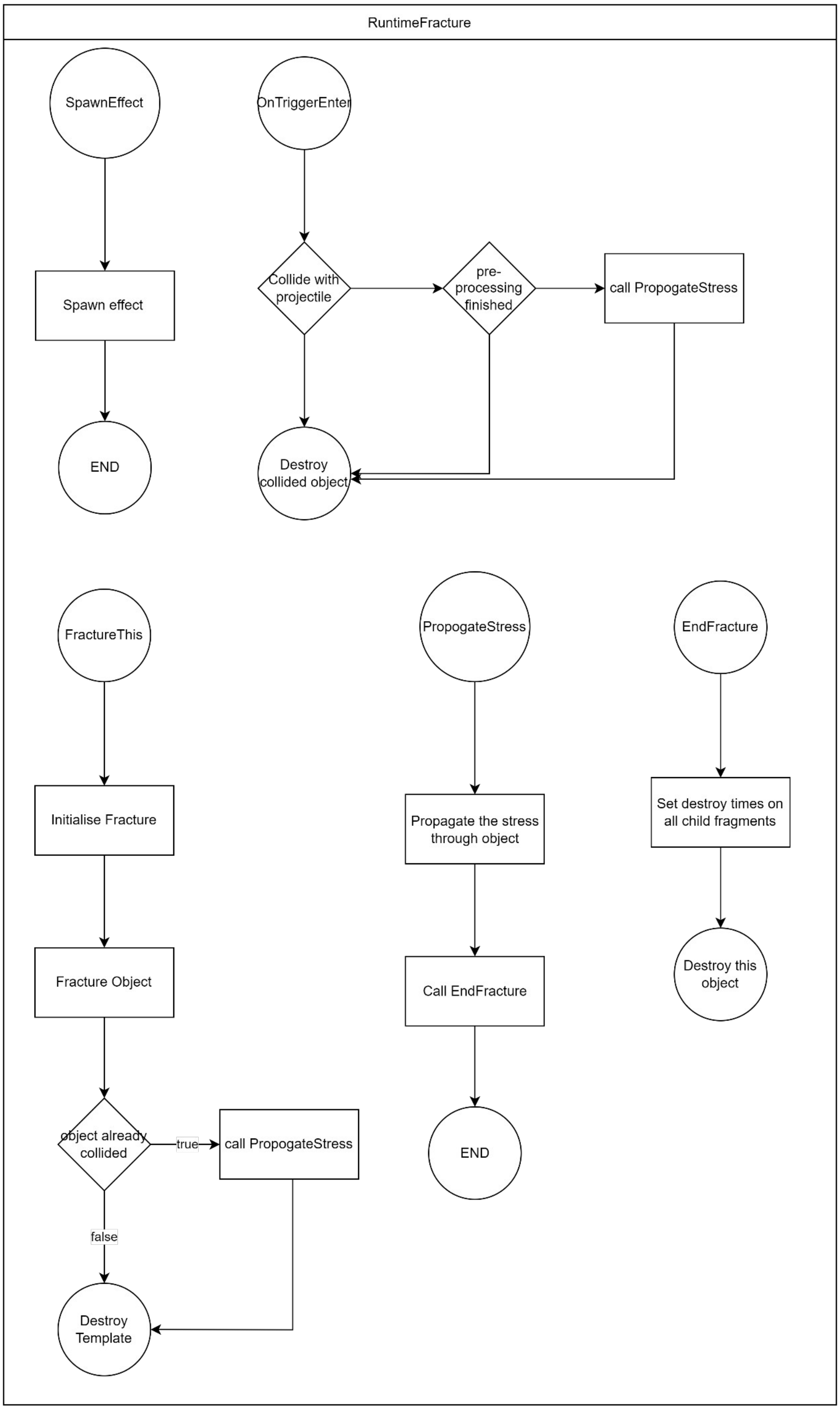

God
Start
Initialise Script
END
Update
Enough Time passed and mouse pressed
TRUE
Call FireTowardsClick
FALSE
END
FireTowardsClick
Create ray in direction of click
Fire bullet in ray direction
Call RayCastActivation. FireRay
END

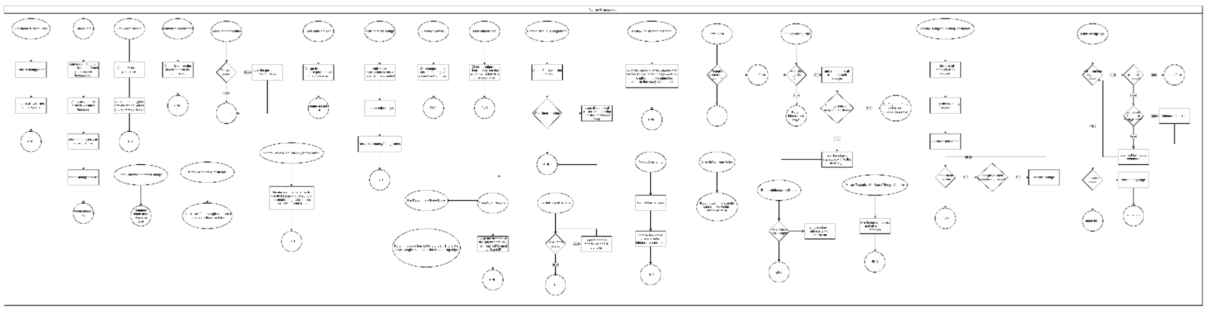

RayCastActivation
Start
Set Camera
END
FireRay
Fire Ray from centre of screen
If ray hit
Destroy object and spawn effect
END

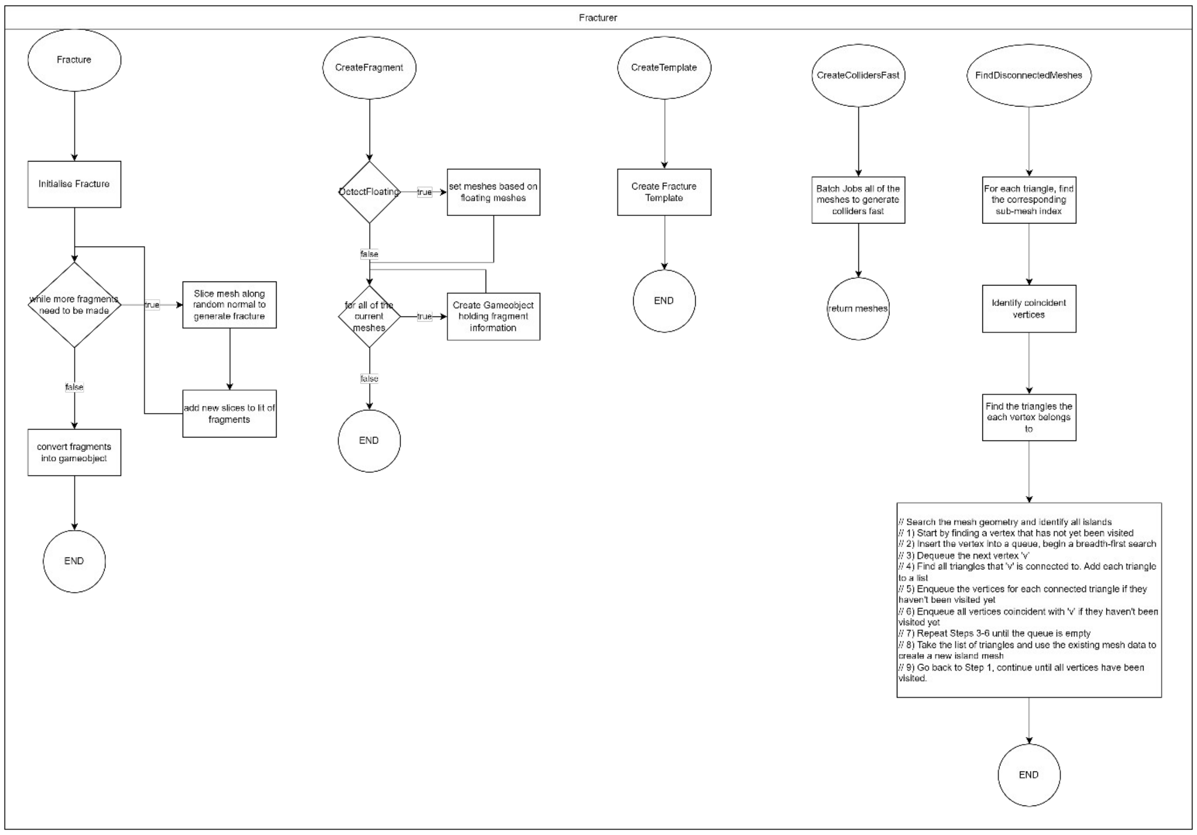

FadeDestroy
Start
Update
Set orignal Times
Scale Object Down
END
if object small enought
Destroy Object
END


PreFracture
PreFractureThis
Create mesh & new fragment
Create Template
Fracture the object
Initialise all of the fragments
END

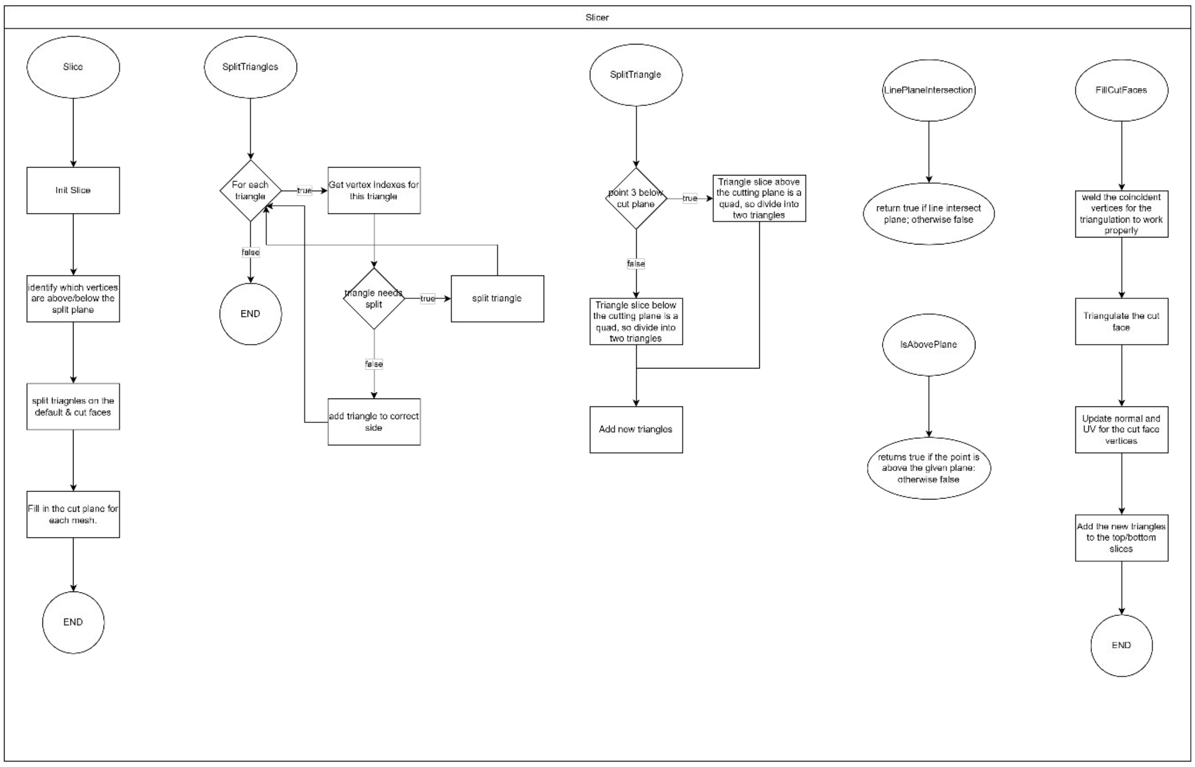

Bullet
OnDestroy
Spawn Effect
END

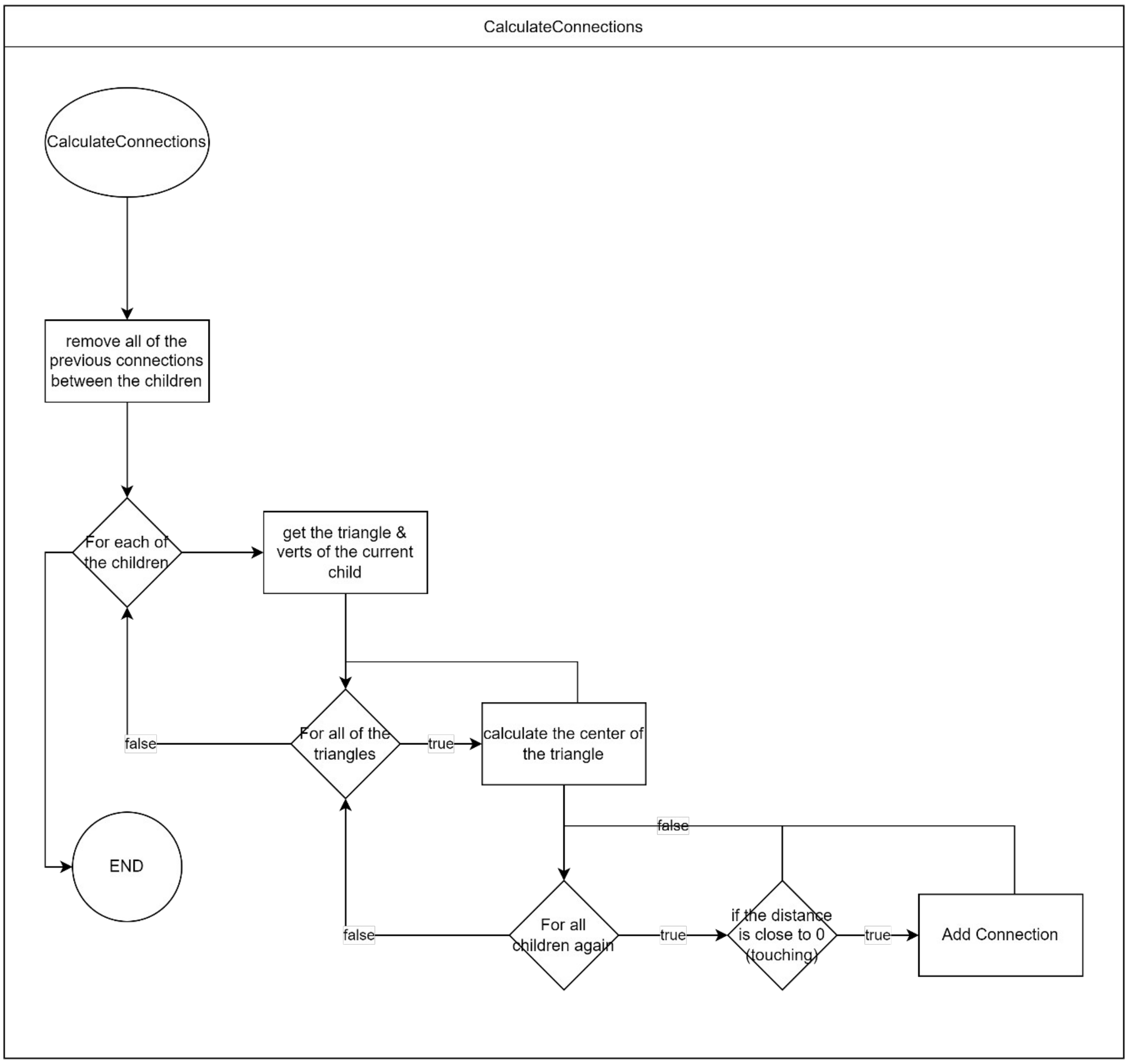

RuntimeFracture
SpawnEffect
Spawn effect
END
OnTriggerEnter
Collide with projectile
pre-processing finished
call PropogateStress
Destroy collided object
FractureThis
Initialise Fracture
Fracture Object
object already collided
true
call PropogateStress
false
Destroy Template
PropogateStress
Propagate the stress through object
Call EndFracture
END
EndFracture
Set destroy times on all child fragments
Destroy this object

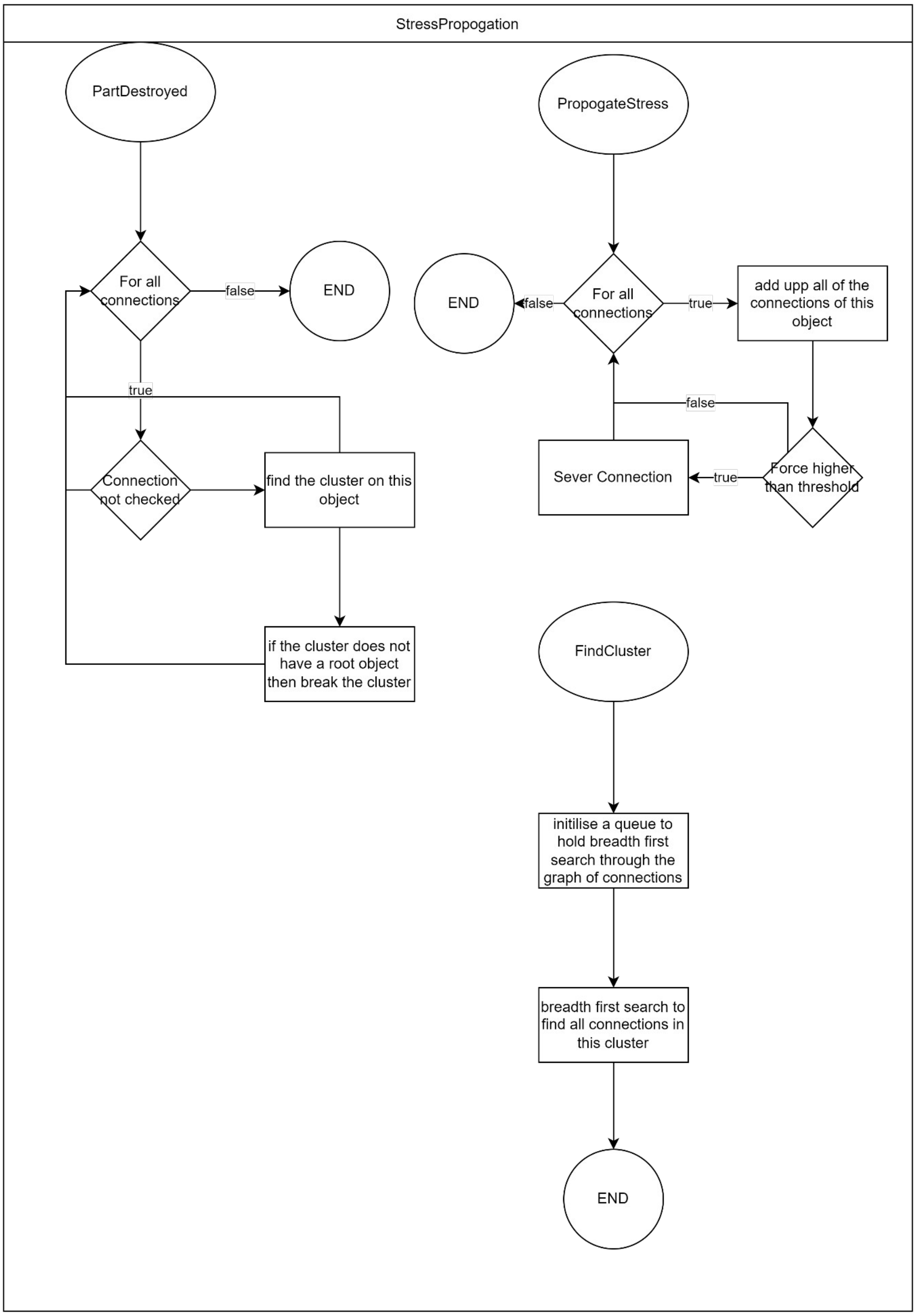


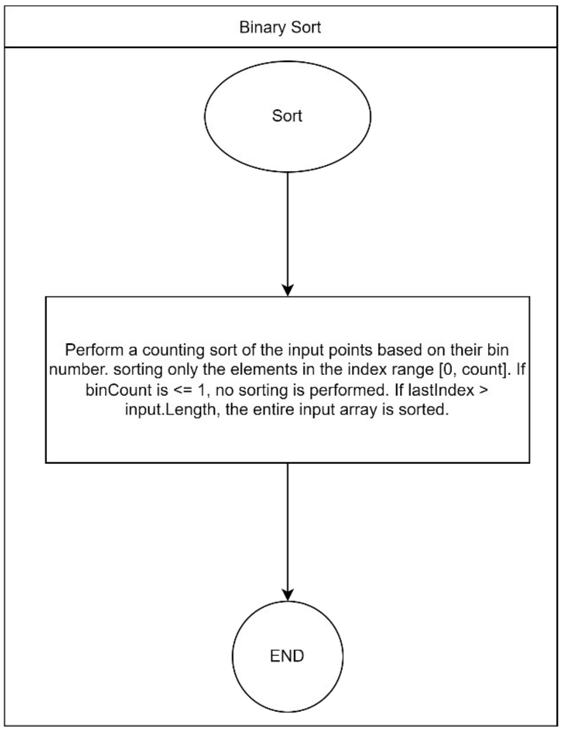

Fracturer
Fracture
Initialise Fracture
while more fragments need to be made
true
Slice mesh along random normal to generate fracture
add new slices to lit of fragments
false
convert fragments into gameobject
END
CreateFragment
DetectFloating
true
set meshes based on floating meshes
false
for all of the current meshes
true
Create Gameobject holding fragment information
false
END
CreateTemplate
Create Fracture Template
END
CreateCollidersFast
Batch Jobs all of the meshes to generate colliders fast
return meshes
FindDisconnectedMeshes
For each triangle, find the corresponding sub-mesh index
Identify coincident vertices
Find the triangles the each vertex belongs to
END

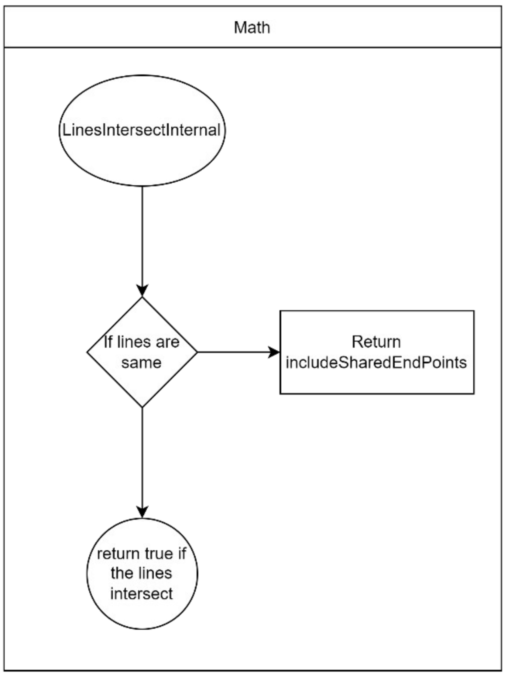
```
// Search the mesh geometry and identify all islands
// 1) Start by finding a vertex that has not yet been visited
// 2) Insert the vertex into a queue, begin a breadth-first search
// 3) Dequeue the next vertex 'v'
// 4) Find all triangles that 'v' is connected to. Add each triangle to a list
// 5) Enqueue the vertices for each connected triangle if they haven't been visited yet
// 6) Enqueue all vertices coincident with 'v' if they haven't been visited yet
// 7) Repeat Steps 3-6 until the queue is empty
// 8) Take the list of triangles and use the existing mesh data to create a new island mesh
// 9) Go back to Step 1, continue until all vertices have been visited.
```


Slicer
Slice
Init Slice
identify which vertices are above/below the split plane
split triagnles on the default & cut faces
Fill in the cut plane for each mesh.
END
SplitTriangles
For each triangle
true
Get vertex indexes for this triangle
false
END
triangle needs split
true
split triangle
false
add triangle to correct side
SplitTriangle
point 3 below cut plane
true
Triangle slice above the cutting plane is a quad, so divide into two triangles
false
Triangle slice below the cutting plane is a quad, so divide into two triangles
Add new triangles
LinePlaneIntersection
return true if line intersect plane; otherwise false
IsAbovePlane
returns true if the point is above the given plane: otherwise false
FillCutFaces
weld the coincident vertices for the triangulation to work properly
Triangulate the cut face
Update normal and UV for the cut face vertices
Add the new triangles to the top/bottom slices
END


CalculateConnections
CalculateConnections
remove all of the previous connections between the children
For each of the children
get the triangle & verts of the current child
For all of the triangles
true
false
calculate the center of the triangle
false
END
For all children again
true
false
if the distance is close to 0 (touching)
true
Add Connection


StressPropogation
PartDestroyed
For all connections
false
END
true
Connection not checked
find the cluster on this object
if the cluster does not have a root object then break the cluster
PropogateStress
For all connections
false
END
true
add upp all of the connections of this object
Force higher than threshold
true
Sever Connection
false
FindCluster
initilise a queue to hold breadth first search through the graph of connections
breadth first search to find all connections in this cluster
END


Binary Sort
Sort
Perform a counting sort of the input points based on their bin number. sorting only the elements in the index range [0, count]. If binCount is <= 1, no sorting is performed. If lastIndex > input.Length, the entire input array is sorted.
END


Math
LinesIntersectInternal
If lines are same
Return includeSharedEndPoints
return true if the lines intersect

Appendix D

Testing Table

Test No.	Description	Expected Outcome	Actual Outcome	Solution
Demo Scripts				
God				
1	top left of the screen clicked	a bullet is fired from the clicked position on the screen down into the word	the bullet is fired on the top left of the screen	N/A
2	bottom left of the screen is clicked	a bullet fired on the bottom left of the screen	the bullet is fired on the bottom left of the screen	N/A
3	middle of the screen is clicked	a bullet fired in	the bullet is fired	N/A

		the middle of the screen	in the middle of the screen	
MindManager				
4	Dropdown changed from 5 -> 10	when breaking objects 10 fragments are now produced	10 fragments are produced when breaking an object	N/A
5	Dropdown changed from 10 -> 15	when breaking objects 15 fragments are now produced	15 fragments are produced when breaking an object	N/A
6	Q button pressed, menu active	menu is disabled	the menu disappears	N/A
7	Q button pressed, menu not active	menu is enabled	the menu appears	N/A
8	R button pressed	the current scene is reloaded	the current scene is reloaded, but when breaking object 5 is destroyed instead of the set 15	Changed the code to update the fragmen t count when a new scene is loaded
9	Left arrow pressed	the previous scene is loaded	scene "9" is loaded (this being the previous because the current was "0")	N/A
10	Right arrow pressed	The next scene is loaded	scene "1" is loaded	N/A

Player				
11	E button pressed, cursor locked	the game is paused, allowing the user to change the current fragment count	the cursor gets unlocked but the game does not appear to be paused	updated the code to stop the running of the rest of the function, pausing the game
12	E button pressed, cursor not locked	the game is unpaused, allowing the user to shoot, move, and interact with the environment again	game pauses	N/A
13	mouse moved, in a circular motion	the player looks in a circular motion	the player looks in a circular motion	N/A
14	forward key pressed, backward key	the player moves forwards and then backwards	moves forwards and then backwards	N/A
15	jump button pressed	the player jumps	the player jumps, but can also jump infinitely	added a ground check, so the player can only jump once per

				ground
16	left mouse button pressed	the player shoots a bullet, that when colliding with a destructible will break it	a bullet is fired, but the mesh breaks straight away	change the code so that the mesh will break, only when the collision happens, this involved creating a pre-calculation of the fracture that is then swapped out when the collision happened
RayCastActivation				
17	ray fired, no object hit	nothing happens	nothing happened	N/A
18	Ray fired a destructible hit	the object is broken	the object fractures	N/A

19	Ray fired nondestructible hit	nothing happens	nothing happened	N/A
Destruction Scripts				
PreFracture				
20	Prefecture, 10 fragments, prototype material 1 as inside mat.	the object is fractured, producing 10 fragments under a new game object, all with the prototype 1 material	a new object with 11 children, all with prototype 1 material	11 objects not 10 is because a floating object detectio n function is run that splits an object resulting in 11 total, although this makes exact numbers harder, it is vital in realism
21	Prefecture, 50 fragments, prototype material 1 as inside mat.	the object is fractured, producing 50 fragments under	a new object with 57 children, all with	for the same reason above

		a new game object, all with the prototype 1 material	prototype 1 mat	there are more objects than expected , without running the island detectio n algorith m it produce s the correct amount
22	Prefecture, 10 fragments, copy materials	the object is fractured, producing 10 fragments under a new game object, all with the same materials as the parent	10 fragments were produced all with the same materials as the parent object	N/A
RuntimeFracture				
23	Fracture, 10 fragments, all syncs, already collided	object breaks into 10 fragments, operations are split into multiple frames	10 fragments appear, and there is a slight delay caused by the async	N/A

24	Fracture, 10 fragments, no async, already collided	object breaks into 10 fragments, with all operations happening	object breaks into 10 fragments with no delay	N/A
25	Fracture, 10 fragments, all syncs, no collision	the object appears the same, but a nonactive object is made that holds the fractured version but over multiple frames	a nonactive object is created that holds the fractured version	N/A
26	Fracture, 10 fragments, no syncs, no collision	the object appears the same, but a nonactive object is made that holds the fractured version, but all this frame	a nonactive object is created that holds the fractured version	N/A
27	Fracture 25 fragments	the object is split into 25 fragments	25 fragments appear, with async there is a slight gap	N/A
28	Fracture 100 fragments	the object is split into 100 fragments	100 fragments appear, with async there is a noticeable gap	N/A
Editor Scripts				
Buttons				

29	find the connection button pressed	the find connection function is run, and the connections for the current object are found	nothing happened, with error	the children of the object need to be found before this can be done, so if they have not been found then call that function first
30	Pre pre-fracture button pressed	fractures the current object according to the given settings	the object is refracted	N/A
31	The get children button is pressed	all of the children of the current object are returned	all of the children a found	N/A
Open Fracture				
Fractures				
32	Fracture called sending null mesh	returns an error, about not having a valid mesh	returns not valid mesh error	N/A

33	The fracture called sending 10 fragment count, no floating detection	source mesh split into 10 fragments	split into 10 pieces	N/A
34	The fracture called sending 20 fragment count, not floating detection	source mesh split into 20 fragments	split into 20 pieces	N/A
35	The fracture called sending 30 fragment count, no floating detection	source mesh split into 30 fragments	split into 30 pieces	N/A
36	Fracture called sending 10 fragment count, with floating detection	source mesh split into 10 fragments	split into 12 pieces	as mention ed in a previous test this is due to island detectio n enabled, as such is intended behavior
37	The fracture called sending 20 fragment count, with floating detection	source mesh split into 20 fragments	split into 25 pieces	N/A
38	The fracture called sending 30 fragment count, with floating detection	source mesh split into 30 fragments	split into 37 pieces	N/A
39	CreateFragment called with mesh with no triangles	does nothing as not a valid input mesh	nothing happens	N/A
40	createFragment called with valid mesh	creates a fragment and	fragment created, with the	N/A

		sets it to be the child of the parent transform	correct mesh, collider, and rigid body	
41	CreateTemplate called with keep tag false	templated created with no tag	tag is "Untagged"	N/A
42	CreateTemplate called with keep tag true	template created that has the tag "Destructible"	tag is "Destructible"	N/A
43	CreateTemplate called with copy materials true	template created that uses the parent's materials	the template uses the parent materials	N/A
44	CreateTemplate called with copy materials false	template created that uses the given material, not the parent materials	the template uses the prototype 1 material	N/A
45	CreateTemplate called no rigid body attached	return no rigid body error	error showing that there is no rigid body is shown	N/A
46	CreateTemplated called no collider attached	returns no collider error	error showing that there is no collider is shown	N/A
47	CreateColliderFast called sending 0 meshes and 1 mesh per job	does nothing as no meshes are given	wastes time dispatching jobs even though no meshes	updated the code to return an empty list if no meshes
48	CreateColliderFast called sending 100 meshes and 10 mesh per job	100 meshes are processed in	meshes are processed	N/A

		batches of 10		
49	CreateColliderFast called sending 10000 meshes and 1 mesh per job	10000 meshes are processed in batches of 1	meshes are processed	N/A
50	CreateColliderFast called sending 100 meshes and -10 mesh per job	error, as cant have a negative amount	stops running of the program	set meshes per job to be 1 if negative
51	FindDissconnectedMehses called with 0 disconnections	The same mesh is returned	nothing happens	N/A
52	FindDissconnectedMehses called with 1 disconnections	2 meshes are returned, and each side of the separation	turned into 2 meshes	N/A
53	FindDissconnectedMehses called with 2 disconnections	3 meshes are returned	turned into 3 meshes	N/A
54	FindDissconnectedMehses called with 10 disconnections	11 meshes are returned	turned into 11 meshes	N/A
Constrained Triangulator				
55	ConstrainedTriangulator called sending less than 3 input points	returns null as cannot operate with less than 3 points (need 3 for a triangle)	returns null	N/A
56	ConstrainedTriangulator called sending more than 3 input points	initializes the triangulator	triangulator is initialized	N/A
57	Triangulate called with N less than 3	returns an empty list as before cannot create triangles with less than 3 points	errors about index out of range	added a check to the start of the function to see if N Is

				greater than 3
58	Triangulate called with N less than 3	calculates the triangulation and returns an array representing the indices of the triangles	array accurately representing the indices is returned	N/A
59	InsertPointIntoTriangle sending a point and triangles	3 new triangles are created, with 1 being overridden by the previous triangle	2 new triangles were created and they original changed to have a point as a vertex	N/A
60	SwapQuaddiagonalIfNeeded called with a swap needed	swaps the triangles, and returns true	returns true	N/A
61	SwapQuaddiagonalIfNeeded called with a swap not needed	nothing happens	returns false	N/A
62	SwapTest sends a triangle that circumscribes the point	returns true	returns true	N/A
63	SwapTest sends a triangle that does not circumscribe the point	return false	returns false	N/A
64	FindStartingEdge called with no starting edge to be found	returns false and creates a new starting edge	returns false, and creates a new edge	N/A
65	FindStartingEdge is called with the starting edge to be found	returns true returning the edge	returns true, with the edge	N/A
66	EdgeConstraintIntersectionTriangle called a triangle that intersects constraint	returns false and sets intersectionEdge Index to -1	intersectionEdge Index = -1 and returns false	N/A

67	EdgeConstraintIntersectionTriangle called a triangle that does not intersect the constraint	returns true with the index of the edge	returns true setting the index to the intersection edge	N/A
68	DiscardTrianglesWithSuperTriangleVerticies called with 4 triangles containing super triangle verts	4 triangles are set to discarded	4 triangles discarded	N/A
69	DiscardTrianglesViolatingConstraints called with 2 triangles violating constraints	2 triangles are set to discarded	2 triangles discarded	N/A
70	RemoveIntersectinEdges called with 10 edges intersection	Swap the diagonals on the edges to ensure they do not intersect	diagonals are swapped and edges no longer intersect	N/A
Slicer				
71	Slice called sending no mesh data	does nothing as no data to work with	does nothing	N/A
72	Slice called sending a mesh with plane located so 20 triangles above and 10 below	mesh is sliced, with 20 above and below	2 meshes created with, ignoring new faces, 20 verts on the first and 10 on the second	N/A
73	Split triangles called with 4 triangles that need to be split (lie on the plane)	4 calls to split triangle, to split the given triangles found to be on the plane	4 triangles are split	N/A
74	SplitTriangle called with two verts above cut normal	divide the top quad into 2 triangles and	cut into 3 triangles, with 2 above and one	N/A

		keep below the same	below the cut normal	
75	Line plane intersection with no intersection	returns false with no intervention	no intervention is found	N/A
76	Line plane intersection with intersection at point (10, 10, 10)	returns true with the intersection at point (10,10,10)	intersection at (10,10,10) found	N/A
77	IsAbovePlane called with a point that is below the plane	returns false	returns false	N/A
78	IsAbovePlane called with a point that is above the plane	returns true	returns true	N/A
Stress Propagation				
Calculate Connections				
79	CalcualteConnections called with 3 objects in a line	sets the first object to 1 connection, second to 2, and third to 1	sets all objects to have 3 connections	changed the code to use the closest point on the collider to determin e the distance between the object, rather than the direction distance between

				the object triangles
80	CalcualteConnections called with 3 objects in a line	sets the first object to 1 connection, second to 2, and third to 1	sets all objects to have 3 connections	updated the code again to transfor m the mesh point into world coordina tes rather than local space to ensure the correct distance is calculate d
81	CalcualteConnections called with 3 objects in a line	sets the first object to 1 connection, second to 2, and third to 1	1, 2, 1	N/A
Stress Propagation				
82	PartDestroyed, with 2 clusters that are no longer connected and 1 cluster that is	2 clusters are found that are not connected to	2 disconnected clusters break off	N/A

		a root object, thus falling. And one cluster that is connected so nothing happens to them		
83	PartDestroyed, with a single root object destroyed	all objects are set to fall	all object fall	N/A
84	PropogateStress is called with 2 connecting objects needing to fall	checks connections and 2 of them don't have enough support to stay so they fall	2 pieces fall	N/A
85	PropogateStress called with 0 connecting objects needing to fall	checks connections and nothing happens, as they all have other connections that are holding them up	nothing happens	N/A
86	FindCluster called with a cluster of 1	1 object is found in the cluster	list of length 1 returned	N/A
87	FindCluster called with a cluster of 10	10 objects are found	list of lengths 10	N/A
88	FindCluster called with a cluster of 100	100 objects are found	list of length 100	N/A
89	FindCluster called with a cluster of 1000	1000 objects are found	list of length 1000	N/A
90	FindCluster called with a cluster of 10000	10000 objects are found	list of length 10000	N/A
Utils				
Binary Sort				

91	Sort called with a count of 1	the first item in the list is sorted	returns the same list	N/A
92	Sort called with a count equal to the list	all items in the list are sorted	returns sorted list	N/A
93	Sort called with a count equal to 10	The first 10 items in the list are sorted	returns list with the first 10 items sorted	N/A
Math				
94	IsQuadConvex called sending a nonconvex quad	returns false	FALSE	N/A
95	IsQuadConvex called sending a convex quad	returns true	TRUE	N/A
96	LinesIntersect sending 2 lines that don’t intersect	returns false	FALSE	N/A
97	LinesIntersect sending 2 lines that intersect	returns true	TRUE	N/A
98	LinesIntersecInternal with 2 lines with shared endpoints	returns true	TRUE	N/A
99	LinesIntersectInternal is called with two lines that are different, but still contain shared points	returns true	TRUE	N/A
100	LinesIntersectInternal but no internal intersection	returns false	FALSE	N/A
101	IsPointOnRightSideOfLine called with a point on the right side of the line	returns true	TRUE	N/A
102	IsPointOnRightSideOfLine called with a point on the left side of the line	returns false	FALSE	N/A

Appendix E
Compressed Benchmarking Data

Pre Fracture Fragments	**Runtime Fragments**	**Mean (Ms)**	**Median (Ms)**	**Upper Quartile (Ms)**	**Lower Quartile (Ms)**	**Max (Ms)**	**Min (Ms)**
10	5	16.71596567	16.6542	16.4132	16.90235	28.2988	5.791
10	10	16.65825	16.64755	16.463	16.83535	21.8206	11.4776
10	15	16.71570967	16.6493	16.4818	16.8229	28.6176	14.4729
10	20	16.66277433	16.66495	16.481325	16.8335	25.4493	8.8037
20	5	16.65825	16.64755	16.463	16.83535	21.8206	11.4776
20	10	16.684955	16.65755	16.49765	16.833025	24.396	13.621
20	15	16.65882767	16.64085	16.46375	16.8346	22.1645	11.1603
20	20	16.66168867	16.6489	16.418825	16.890375	27.7844	6.3524
20	25	16.76967633	16.66985	16.486225	16.84425	30.73	9.396
30	5	16.71596567	16.6542	16.4132	16.90235	28.2988	5.791
30	10	16.65951067	16.64795	16.455875	16.826475	23.2096	11.3309
30	15	16.659601	16.67525	16.47645	16.876775	19.7929	13.0666
30	20	16.71485633	16.6531	16.524875	16.798525	33.4485	13.9185
30	25	16.65592367	16.64465	16.48605	16.809025	19.9738	13.3697
40	5	16.654167	16.6495	16.4637	16.84107	21.813	10.851

			5		5	6	
40	10	16.659695	16.66245	16.46815	16.86225	24.3348	9.066
40	15	16.663657	16.6612	16.501275	16.85005	22.4925	10.8777
40	20	16.661103	16.6561	16.53025	16.810575	22.3054	11.0166
40	25	16.65926667	16.67425	16.405975	16.869375	19.253	14.5868
50	5	16.71636467	16.66535	16.493275	16.852075	34.1941	10.7922
50	10	16.66074567	16.64405	16.48195	16.870875	27.3413	6.0053
50	15	16.66054133	16.6488	16.475675	16.828275	28.8942	6.491
50	20	16.65961033	16.6582	16.512175	16.828125	21.2606	12.0835
50	25	16.715793	16.66075	16.496525	16.81875	32.3783	6.6174
60	5	16.660802	16.6591	16.463075	16.8587	24.0195	9.3923
60	10	16.661833	16.6566	16.480175	16.81685	27.2695	6.0311
60	15	16.73618367	16.66165	16.477625	16.83475	31.6991	9.5775
60	20	16.66222433	16.64485	16.46975	16.830975	30.5618	5.6489
60	25	16.77114633	16.6748	16.473325	16.862675	26.0522	13.6301
70	5	16.66048833	16.65955	16.46145	16.83275	19.6361	13.6757
70	10	16.71736033	16.654	16.456325	16.8396	31.4318	7.5812

70	15	16.660038	16.6621	16.519075	16.8224	20.0419	14.101
70	20	16.66178767	16.66685	16.458175	16.8499	23.0544	10.2101
70	25	16.65992233	16.65945	16.466375	16.834225	18.914	14.6475
80	5	16.660156	16.64875	16.408475	16.87205	22.9977	12.3057
80	10	16.66031033	16.65465	16.4713	16.843175	20.972	11.8988
80	15	16.68497733	16.65655	16.459975	16.861125	24.1183	10.8367
80	20	16.65879433	16.6659	16.5019	16.84215	25.2206	8.2516
80	25	16.65926333	16.6553	16.5116	16.768325	24.3477	8.647
90	5	16.77175633	16.6699	16.477775	16.857475	33.2858	12.2494
90	10	16.65715867	16.65255	16.456	16.831375	25.8741	7.6185
90	15	16.66085233	16.66405	16.444625	16.875825	21.4385	11.7045
90	20	16.65990367	16.6639	16.46425	16.87985	21.4595	10.6424
90	25	16.715623	16.6509	16.478675	16.8602	28.3525	13.1169
100	5	16.661279	16.66905	16.43005	16.857025	24.9697	7.7478
100	10	16.659345	16.6634	16.428675	16.8727	27.3051	6.0301
100	15	16.66045667	16.66815	16.41255	16.870525	26.3549	6.8063
100	20	16.661109	16.6662	16.47065	16.86767	21.850	12.916

					5	1	4
100	25	16.6593623 3	16.6447	16.48172 5	16.85867 5	19.079 5	12.811

Appendix F
Raw Questionnaire Data

The destruction in the video felt realistic.	**The destruction happened smoothly without noticeable lag.**	**The Destruction had a high level of detail**	**The overall experience of watching the destruction was engaging.**	**Describe any instances where the destruction felt unrealistic or unnatural. (e.g., Buildings collapsing in an unexpected way)**	**What aspects of the destruction detail impressed you the most?**	**Were there any specific features you found missing that would enhance the realism?**
4	5	3	4	0:12 The top of the building explodes and there's a massive chunk that flies up - I don't think that would be	amount of sub meshes on screen with commendable runtime framerate	destruction over time - perhaps some sort of stability system?

				possible		
4	4	4	5	I feel the clumps that broke off from the building were too large to be considere d "Realistic "	The VFX on impact and the falling rate	A puff of smoke when the buildings hit the ground could help with realism
4	5	4	4	Broken parts of the buildings blew away too easily and too far and didn't feel like they had the weight a building should.	How fast/non laggy it was, all real-time and dynamic	Smaller fractures in the building
2	5	4	4	Pieces sometime s fly off with high velocity into the air, and	It can split meshes into pieces with (seemingly) random shapes, but they never	Making the pieces have less speed dependin g on how

				some of them look pretty polygonal and blocky	look contorted or unrealistic	big they are, for example making the larger pieces unable to fly high into the air, and maybe crack into smaller pieces when hitting a surface at high enough speed
4	5	4	3	That skyscraper coming down in giant chunks felt a bit off. Buildings tend to break apart in smaller	The level of detail	The number of debris pieces on the screen without any slowdown. It really made the destructi

				sections, not in massive, clean breaks.		on feel massive and chaotic.
4	5	3	4	giant chunks flying across the city at that speed? More like they'd crumble and fall nearby.	the VFX was really nice the way the explosions impacted the buildings looked fantastic. The dust, sparks, and debris all felt real and immersive.	While the random debris pieces were cool, having the size and speed of debris more closely tied to material and weight would be a nice touch.
4	4	4	4	some of that debris looked a little too boxy, Like chunks ripped from a	Seeing the destruction happen in real time was amazing. No cuts, no weird transitions,	The destructi on felt very instant. Perhaps a system where buildings

				video game, not a real building.	just pure, dynamic mayhem.	weaken and develop fractures before complete collapse would be even more immersiv e.
3	5	3	5	Buildings wouldn't crumble in such large, clean pieces.	They weren't just pre-made chunks, they looked like real fragments of the structures.	Large chunks shouldn't fly as far, and maybe even break apart on impact.
3	4	4	4	The parts leaving the building felt too large, I also feel they flew away too much	the dynamic buildings looked really nice how you could shoot down the bottom and the rest of the building would fall	more debris would be nice, the current VFX was good but some more detailed VFX could

						make it even better
5	5	5	5	the piece just shrinking looks a bit odd, also the force at which the pieces shoot away	the VFX looked really nice	some more debris effect, to tie in the destructi on after the initial break
4	5	4	4	The way the bomb falls from the sky	Building collapsing	More smoke when the bombs explode
3	4	3	3	The destroyed parts felt too smooth. The internals of buildings being completel y solid felt a bit immersio n-	How little lag there was.	Perhaps the broken pieces break again when they hit the ground or other objects.

				breaking.		
3.666666667	**4.666666667**	**3.75**	**4.083333333**			