Module Guide for SFWRENG 4G06 - Capstone Design Project

Team #7, Wardens of the Wild
Felix Hurst
Marcos Hernandez-Rivero
BoWen Liu
Andy Liang

January 12, 2026

1 Revision History

Date	Version	Notes
Nov 13	1.0	Initial doc

2 Reference Material

2.1 Abbreviations and Acronyms

AC Anticipated Change
DAG Directed Acyclic Graph
M Module
MG Module Guide
OS Operating System
R Requirement
SC Scientific Computing
SRS Software Requirements Specification ProgName Explanation of program name
UC Unlikely Change

Contents

List of Tables

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team [3]. We advocate a decomposition based on the principle of information hiding [1]. This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by [3], as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed [3]. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Hardware that the game is made available in (Windows, Mac, Linux, Android, iOS, etc.).

AC2: Input device being used (mouse and keyboard, touchscreen, etc.).

AC3: Rendering resolution and pixel density for different display types.

AC4: Material properties for different destructible objects (softness, strength, friction, texture patterns, etc.).

AC5: Number, size, shape and behaviour of debris objects generated from cuts in destructible objects.

AC6: Pixel size used for pixelated cut edge rendering.

AC7: Algorithm used for irregular edge generation when destroying objects.

AC8: Debug and production visualization style of the cutting line and intersection points (colors, sizes, styles).

AC9: The physics properties of the environment (i.e. intensity of gravity).

AC10: Methodology for cut object selection (top, bottom, closest to player).

AC11: Mesh triangulation algorithm (Bresenhan algorithm versus alternatives).

AC12: Format and storage location of cut profile configuration data (JSON, databases, etc.).

AC13: Force range and patterns for destructible environment.

AC14: The lifetime and cleanup behaviour of debris fragments after spawning from destroyed objects.

AC15: The collision detection layers and filtering rules/parameters for raycasting.

AC16: The availability of accessibility modes, such as contrast adjustments for colorblindness.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Game engine (Unity).

UC2: Dimensionality (2D).

UC3: Focus and use of physics simulation for object behaviour and world simulation.

UC4: Requirement for real-time object destruction physics.

UC5: The fundamental geometric representation of objects as pixel-based polygons.

UC6: Use of entry and exit points as a basis of destructive tools.

UC7: The partitioning of objects into two entities (original object, cut off object) per tool use.

UC8: The coordinate system (right-handed 2D cartesian coordinates).

UC9: Use of mesh rendering techniques for displaying cut objects.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

Level 1	Level 2
Hardware-Hiding	Unity Input System
	Unity Physics 2D
	Unity Rendering
	Unity collider system
Behaviour-Hiding	Player Control module
	Raycast visualization module
	Cut execution module
	Debris generation module
Cut profile manager module	
Software Decision	Geometric operations module
	Polygon partitioning module
	Mesh generation module
	Pixelation module
	Physics update module
	Material properties module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

Design Decisions

The following design decisions are made to bridge the gap between requirements and implementation:

DD1: Dual Representation - All cut objects have a pixelated mesh texture using Bresenham algorithm applied to them in contrast with the smooth collision mesh even if the cut was irregular and/or diagonal in order to ensure adherence to the visual style(R6).

DD2: Material Profile System - In order to meet the requirement for different cutting behavior on different materials (R5) a JSON-based configuration system maps material names/tags to extendable cutting parameters (softness and strength).

DD3: Modular Cutting Pipeline - Cutting object flow can be broken into the following key phases of visualization, partitioning, mesh generation, and debris spawning thereby allowing each phase to be optimized independently and meeting the requirement for real-time interactive cutting (R3) in an extendable and maintainable way.

DD4: Entry-Exit Point Model - In order for players to have a clear understanding of what the tool destroys (R2), cuts are defined by exactly two intersection points for each object (entry and exit) which simplifies the logic and provides clear visualization and working feedback to the player

DD5: Area-Proportional Mass - For realistic physics behavior after cutting (R4), object mass is updated proportionally to the change in area, maintaining consistent density.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The Secrets field in a module decomposition is a brief statement of the design decision hidden by the module. The Services field specifies what the module will do without documenting how to do it. For each module, a suggestion for the implementing software is given under the Implemented By title. If the entry is OS, this means that the module is provided by the operating system or by standard programming language libraries. ProgName means the module will be implemented by the ProgName software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (-) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules

7.1.1 Unity Input System (M1)

Secrets The data structure and algorithm used to detect and process user input events (mouse position, button presses, keyboard input).

State Variables

- $mousePosition : \mathbb{R}^2$ — Current mouse position in screen coordinates
- $mouseButtonDown : \mathbb{B}$ — Whether left mouse button is pressed
- $keyboardInput$: seq of char — Currently pressed keyboard keys

Environment Variables

- Physical mouse device

- Physical keyboard device
- Screen resolution and DPI

Access Programs

Name	In	Out	Exceptions
GetMousePosition	-	Vector2	-
GetMouseButton	int	$\mathbb{B}$	-
GetKey	KeyCode	$\mathbb{B}$	-
ScreenToWorldPoint	Vector2	Vector2	-

Services Detects mouse movement and clicks for aiming and tool use, detects keyboard input for player movement and jumping and translates screen coordinates to world coordinates.

Implemented By Unity (UnityEngine.InputSystem)

Type of Module Library

7.1.2 Unity Physics2D (M2)

Secrets Library used for rigid bodies, physics, and collisions.

State Variables

- $gravity : \mathbb{R}^2$ — Global gravity vector
- $physicsTimeStep : \mathbb{R}^+$ — Fixed time step for physics updates

Environment Variables

- System time
- CPU processing capabilities

Access Programs

Name	In	Out	Exceptions
Raycast	Vector2, Vector2, $\mathbb{R}$, LayerMask	RaycastHit2D	
AddForce	Rigidbody2D, Vector2	-	-
OverlapPoint	Vector2	Collider2D	-

Services Performs 2D raycasting with filtering, manages Rigidbody2D components, detects collisions between 2D colliders, applies forces and velocities to objects, simulates gravity and momentum.

Implemented By Unity (UnityEngine.Physics2D)

Type of Module Library

7.1.3 Unity Rendering (M3)

Secrets The algorithm for rendering 2D sprites, meshes, and line renderers to the screen with proper depth sorting.

State Variables

- *renderQueue* : seq of RenderObject — Queue of objects to render
- *sortingLayers* : seq of Layer — Defined sorting layers

Environment Variables

- Graphics card capabilities
- Display output device
- Screen resolution

Access Programs

Name	In	Out	Exceptions
DrawMesh	Mesh, Material	-	-
DrawSprite	Sprite, Vector2	-	-
DrawLine	Vector2, Vector2, Color	-	-

Services Renders sprites, renders dynamically generated meshes with textures, renders LineRenderer components for visualization, manages render order and sorting layers, applies materials and shaders.

Implemented By Unity (UnityEngine.Rendering, UnityEngine.SpriteRenderer, UnityEngine.LineRende

Type of Module Library

7.1.4 Unity Collider System (M4)

Secrets The data structure and algorithm for storing and querying 2D collider shapes, including dynamic collider modification.

State Variables

- *colliders* : seq of Collider2D — All active colliders in scene
- *collisionMatrix* : LayerMask × LayerMask → $\mathbb{B}$ — Layer collision matrix

Environment Variables None

Access Programs

Name	In	Out	Exceptions
CreatePolygonCollider	seq of Vector2	PolygonCollider2D	-
UpdateColliderPoints	PolygonCollider2D, seq of Vector2	-	-
GetColliderBounds	Collider2D	Bounds	-

Services Provides BoxCollider2D and PolygonCollider2D components also used to update collider shapes dynamically at runtime for objects and manages collider events.

Implemented By Unity (UnityEngine.Collider2D, UnityEngine.PolygonCollider2D, UnityEngine.BoxCollider2D)

Type of Module Library

7.2 Behaviour-Hiding Modules

Secrets

The contents of the required behaviours.

Services

Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By

–

7.2.1 Player Controller (M5)

Secrets Controls behaviour of the player character such as movement speed, jump height, ground detection, etc.

State Variables

- $playerPosition : \mathbb{R}^2$ — Current position of player
- $playerVelocity : \mathbb{R}^2$ — Current velocity vector
- $isGrounded : \mathbb{B}$ — Whether player is touching ground
- $moveSpeed : \mathbb{R}^+$ — Movement speed in units/second
- $jumpForce : \mathbb{R}^+$ — Upward force applied when jumping

Environment Variables

- User input from M1
- Physics simulation from M2

Exported Constants

- $DEFAULT\_MOVE\_SPEED = 5.0$
- $DEFAULT\_JUMP\_FORCE = 10.0$
- $GROUND\_CHECK\_RADIUS = 0.2$

Access Programs

Name	In	Out	Exceptions
Move	$\mathbb{R}$	-	-
Jump	-	-	exc := $\neg isGrounded$
UpdateGroundCheck		$\mathbb{B}$	-
InitializeRaycast	-	-	-

Services Based on input from Unity's Input system, moves player character based on input, handles jumping when grounded, initializes and manages the raycast cutting system, visualizes ground check radius in editor.

Implemented By PlayerController.cs

Type of Module Abstract Object

7.2.2 Raycast Visualization Module (M6)

Secrets How the cutting line and intersection points are visualized to the user, including colors, sizes, and rendering properties.

State Variables

- $lineStartPoint : \mathbb{R}^2$ — Start position of cutting line
- $lineEndPoint : \mathbb{R}^2$ — End position of cutting line
- $entryPoint : \mathbb{R}^2$ — Entry intersection point
- $exitPoint : \mathbb{R}^2$ — Exit intersection point
- $targetObject$: GameObject — Currently targeted object
- $lineColor$: Color — Color of visualization line

Environment Variables

- Mouse position from M1
- Raycast results from M2
- Rendering system from M3

Exported Constants

- $ENTRY\_POINT\_COLOR =$ Green
- $EXIT\_POINT\_COLOR =$ Orange
- $LINE\_WIDTH = 0.05$
- $POINT\_RADIUS = 0.1$

Access Programs

Name	In	Out	Exceptions
DrawCuttingLine	Vector2, Vector2	-	-
DrawIntersectionPoints	Vector2, Vector2	-	-
HighlightCutRegion	seq of Vector2	-	-
DetectMouseClick	-	$\mathbb{B}$	-

Services Draws a line from player to mouse cursor, displays entry point (green dot) and exit point (orange dot) on target object, highlights the portion of object that will be cut off, filters out invalid raycast targets (player, debris), detects mouse clicks to execute cuts.

Implemented By Raycast.cs

Type of Module Abstract Object

7.2.3 Tool Raycast (M7)

Secrets Coordinates between controller and reshape and debris systems as the executor and manager of the tool use.

State Variables

- $currentCutTarget$: GameObject — Object being cut
- $entryPoint : \mathbb{R}^2$ — Cut entry point
- $exitPoint : \mathbb{R}^2$ — Cut exit point
- $cutLineDirection : \mathbb{R}^2$ — Direction of cut

Environment Variables

- Visualization state from M6
- Material profiles from M9

Access Programs

Name	In	Out	Exceptions
ExecuteCut	GameObject, Vector2, Vector2	-	exc := $target = null$
ApplyCutBehavior	GameObject, CutProfile	-	-
HighlightEdge	seq of Vector2	-	-
SelectCutPortion	GameObject	seq of Vector2	-

Services Responsible for applying the appropriate cut behavior and edge highlighting and cut selection coordinating between ObjectReshape and DebrisSpawner.

Implemented By RaycastReceiver.cs

Type of Module Abstract Object

7.2.4 Debris Generation (M8)

Secrets Generating and distributing debris fragments within the cut-off area, including fragment sizing and positioning.

State Variables

- $activeDebris$: seq of GameObject — Currently active debris objects
- $debrisLifetime : \mathbb{R}^+$ — Time before debris cleanup
- $explosionForce : \mathbb{R}^+$ — Force applied to debris

Environment Variables

- Physics system from M2
- Rendering system from M3
- Collider system from M4

Exported Constants

- $DEFAULT\_DEBRIS\_COUNT = 5$
- $DEFAULT\_LIFETIME = 10.0$
- $MASS\_MULTIPLIER = 0.1$

Access Programs

Name	In	Out	Exceptions
SpawnDebris	seq of Vector2, int, Material	seq of GameObject	-
PositionFragment	Vector2, seq of Vector2	Vector2	-
ApplyExplosionForce	GameObject, Vector2, $\mathbb{R}$	-	-
CleanupDebris	GameObject	-	-

Services Creates specified number of debris fragments based on object config and cut size, positions fragments within cut-off polygon using point-in-polygon testing, applies physics properties (mass, velocity, angular velocity) and applies explosion force radiating from cut centroid as well as managing debris lifetime and cleanup.

Implemented By DebrisSpawner.cs

Type of Module Abstract Object

7.2.5 Cut Profile Manager (M9)

Secrets Manager cut profiles and retrieval for material specific cutting profiles.

State Variables

- $cutProfiles$: Tag $\rightarrow$ CutProfile — Mapping of tags to profiles
- $defaultProfile$: CutProfile — Default cutting parameters

Environment Variables

- JSON configuration file

Exported Constants

- $DEFAULT\_SOFTNESS = 0.5$
- $DEFAULT\_STRENGTH = 0.5$
- $DEFAULT\_DEBRIS\_COUNT = 5$

Access Programs

Name	In	Out	Exceptions
LoadProfiles	string	-	exc := invalid JSON
GetProfileForObject	GameObject	CutProfile	-
ApplyIrregularCut	seq of Vector2, CutProfile	seq of Vector2	-

Services Loads cut profiles from JSON configuration and applies them based on object tag (otherwise supplies default profile when specific profile not found) and applies irregular cutting based on material properties (softness and strength parameters).

Implemented By CutProfileManager.cs

Type of Module Abstract Object

7.3 Software Decision Modules

Secrets

The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are not described in the SRS.

Services

Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By

–

7.3.1 Geometric Operations (M10)

Secrets Algorithms for fundamental geometric calculations on 2D shapes including the shoelace formula, cross products, distance calculations.

State Variables None (stateless module)

Environment Variables None

Exported Constants

- $EPSILON = 10^{-6}$ — Floating point comparison tolerance

Access Programs

Name	In	Out	Exceptions
CalculateArea	seq of Vector2	$\mathbb{R}^+$	-
CalculateCentroid	seq of Vector2	Vector2	-
CrossProduct	Vector2, Vector2	$\mathbb{R}$	-
Distance	Vector2, Vector2	$\mathbb{R}^+$	-
Normalize	Vector2	Vector2	exc := $\|v\| = 0$

Services Calculates polygon areas for object reshaping after a cut is applied from the raycast for both colliders and mesh renders.

Implemented By ObjectReshape.cs (geometric utility functions), PixelatedCutRenderer.cs (grid operations)

Type of Module Library

7.3.2 Polygon Partitioning (M11)

Secrets The algorithm for splitting a polygon into two parts, the original piece and the cutoff piece along a cut line using cross product side determination.

State Variables None (stateless module)

Environment Variables None

Access Programs

Name	In	Out	Exceptions
PartitionPolygon	seq of Vector2, Vector2, Vector2	(seq of Vector2) × (seq of Vector2)	-
DetermineKeepSide	seq of Vector2, seq of Vector2, Vector2	seq of Vector2	-
AddIntersectionPoints	seq of Vector2, Vector2, Vector2	seq of Vector2	-

Services Partitions polygon vertices based on which side of cut line they fall on, adds entry and exit points to both resulting polygons and determines which partition matches highlighted shape using centroid comparison while preserving vertex ordering.

Implemented By ObjectReshape.cs (CutOffPortion method)

Type of Module Library

7.3.3 Mesh Generation Module (M12)

Secrets Converting polygon vertices into renderable mesh data using ear clipping triangulation.

State Variables None (stateless module)

Environment Variables

- Rendering system from M3

Access Programs

Name	In	Out	Exceptions
TriangulatePolygon	seq of Vector2	seq of int	exc := $\|V\| < 3$
GenerateUVs	seq of Vector2	seq of Vector2	-
CreateMesh	seq of Vector2, seq of int	Mesh	-
IsEar	Vector2, Vector2, Vector2, seq of Vector2	$\mathbb{B}$	-

Services Implements ear-clipping algorithm to triangulate arbitrary polygons, automatically generates UV coordinates based on the polygon's bounding box, constructs Unity Mesh objects given the vertex data, supports both convex and concave shapes.

Implemented By ObjectReshape.cs (UpdateVisualMesh, TriangulatePolygon methods)

Type of Module Library

7.3.4 Pixelation Module (M13)

Secrets For visual uniformity, converting smooth diagonal cut lines into pixel-aligned staircase patterns using a modified Bresenham approach.

State Variables None (stateless module)

Environment Variables None

Exported Constants

- $DEFAULT\_PIXEL\_SIZE = 0.0625$

Access Programs

Name	In	Out	Exceptions
PixelateCutEdges	seq of Vector2, $\mathbb{R}^+$	seq of Vector2	-
SnapToGrid	Vector2, $\mathbb{R}^+$	Vector2	-
IdentifyCutEdges	seq of Vector2, Vector2, Vector2	seq of (int $\times$ int)	-

Services Pixelates only cut edges while keeping original edges intact, snaps points to a pixel grid with floor-based rounding, creates staircase patterns with horizontal and vertical segments only.

Implemented By PixelatedCutRenderer.cs

Type of Module Library

7.3.5 Physics Update Module (M14)

Secrets The algorithm for updating physics properties after shape modification including the area-based mass calculation method.

State Variables None (stateless module)

Environment Variables

- Physics system from M2
- Collider system from M4

Access Programs

Name	In	Out	Exceptions
UpdateCollider	GameObject, seq of Vector2	-	-
UpdateRigidbodyMass	Rigidbody2D, $\mathbb{R}^+$, $\mathbb{R}^+$	-	-
ConvertToPolygonCollider	GameObject	PolygonCollider2D	

Services Converts from BoxCollider2D to PolygonCollider2D to match new object shape and adjusts proportionally the RigidBody 2D mass to match the new area change using the shoelace area calculation, creates smooth collider for physics and pixelated visual mesh for visual consistency.

Implemented By ObjectReshape.cs (UpdateCollider, UpdateRigidbodyMass methods)

Type of Module Library

7.3.6 Material Properties Module (M15)

Secrets The data structure for storing material cutting characteristics and the algorithm for applying irregular edge offsets.

State Variables None (stateless module)

Environment Variables None

Exported Constants

- $MIN\_SEGMENTS = 3$
- $MAX\_SEGMENTS = 15$
- $SMOOTH\_THRESHOLD = 0.33$
- $JAGGED\_THRESHOLD = 0.67$

Access Programs

Name	In	Out	Exceptions
GenerateIrregularEdge	Vector2, Vector2, $\mathbb{R}$, $\mathbb{R}$	seq of Vector2	-
CalculateSegmentCount	$\mathbb{R}$	$\mathbb{N}$	-
CalculateMaxOffset	$\mathbb{R}$, $\mathbb{R}$	$\mathbb{R}^+$	-
ApplySineWave	Vector2, Vector2, int, $\mathbb{R}$	seq of Vector2	-
ApplyRandomOffset	Vector2, Vector2, int, $\mathbb{R}$	seq of Vector2	-

Services Defines softness parameter (0 = jagged and 1 = smooth) which affects segment count, defines strength parameter (0 = clean cut and 1 = maximum irregularity) affecting offset magnitude, generates irregular cut edges using different functions based on softness (sine waves for smooth, random for jagged and hybrid for medium).

Implemented By CutProfileManager.cs (ApplyIrregularCut, GenerateIrregularEdge methods)

Type of Module Library

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Table 2: Trace Between Requirements and Modules

Req.	**Modules**
R1: Player Movement and Control	M7.1.1, M7.1.2, M7.2.1
R2: Cutting Line Visualization	M7.1.1, M7.1.2, M7.1.3, M7.2.2
R3: Real-time Object Cutting	M7.1.2, M7.1.3, M7.1.4, M7.2.3, M7.3.1, M7.3.2, M7.3.3, M7.3.5
R4: Debris Generation	M7.1.2, M7.1.3, M7.1.4, M7.2.4, M7.3.1
R5: Material Specific Cutting	M7.2.3, M7.2.5, M7.3.6
R6: Pixelated Rendering	M7.1.3, M7.3.1, M7.3.3, M7.3.4
R7: Physics Simulation	M7.1.2, M7.1.4, M7.3.5
R8: Collision Detection	M7.1.2, M7.1.4, M7.2.2
R9: Visual Feedback	M7.1.3, M7.2.2, M7.2.3
R10: Multiple Cuts on Same Object	M7.2.3, M7.3.2, M7.3.3, M7.3.5

Table 3: Trace Between Requirements and Modules

Table 4: Trace between Anticipated Changes and Modules

AC	Modules
AC1	M7.1.1, M7.1.2, M7.1.3, M7.1.4
AC2	M7.1.1, M7.2.1
AC3	M7.1.3
AC4	M7.2.5, M7.3.6
AC5	M7.2.4
AC6	M7.3.4
AC7	M7.3.6
AC8	M7.2.2
AC9	M7.3.5
AC10	M7.2.3
AC11	M7.3.3
AC12	M7.2.5
AC13	M7.2.4
AC14	M7.2.4
AC15	M7.2.2

Table 5: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

The following list describes the direct dependencies between modules (A → B means A uses B):

- M7.2.1 → M7.1.1, M7.2.2
- M7.2.2 → M7.1.1, M7.1.2, M7.1.3, M7.2.3, M7.3.1
- M7.2.3 → M7.2.4, M7.2.5, M7.3.1, M7.3.2, M7.3.3, M7.3.4, M7.3.5
- M7.2.4 → M7.1.2, M7.1.3, M7.1.4, M7.3.1
- M7.2.5 → M7.3.6
- M7.3.1 → M7.1.2
- M7.3.2 → M7.3.1
- M7.3.3 → M7.1.3, M7.3.1
- M7.3.4 → M7.3.1
- M7.3.5 → M7.1.2, M7.1.4, M7.3.1
- M7.3.6 → M7.3.1

9.1 Use Hierarchy levels

Level 0 (Foundation - Hardware Hiding)

- M7.1.1
- M7.1.2
- M7.1.3
- M7.1.4

Level 1 (Utility - Basic Operations)

- M7.3.1

Level 2 (Algorithm - Core computations)

- M7.3.2
- M7.3.3
- M7.3.4
- M7.3.5
- M7.3.6

Level 3 (Services - System Coordination)

- M7.2.4
- M7.2.5

Level 4 (Coordination - Behaviour Management)

- M7.2.2
- M7.2.3

Level 5 (Application - User Interface)

- M7.2.1

10 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma]

11 Design of Communication Protocols

11.1 Inter-Module Communication

The system uses Unity's component-based architecture for module communication:

Event-Based Communication:

- **Input Events:** Unity's Input System generates events for mouse clicks, movement, and jumps
- **Collision Events:** Physics2D triggers `OnCollisionEnter2D` events when debris or cut pieces collide
- **Frame Events:** Unity's `Update()` and `FixedUpdate()` provide timing for continuous operations

Direct Method Calls:

- M6 (Raycast Visualization) calls M7 (Cut Execution) via `ExecuteCut()`
- M7 calls M11 (Polygon Partitioning) via `PartitionPolygon()`
- M7 calls M8 (Debris Generation) via `SpawnDebris()`
- M9 (Cut Profile Manager) is accessed via static extension method `GetCutProfileForObject()`

Component References:

- Modules get references using `GetComponent<T>()` for same-GameObject communication
- Modules use `FindObjectOfType<T>()` for singleton manager access
- Parent-child relationships use `GetComponentInChildren<T>()`

11.2 Data Flow

1. **Input Phase:** M1 → M5 → M6
2. **Visualization Phase:** M6 → M2 → M10 → M7
3. **Execution Phase:** M7 → M11 → M15 → M12 → M13 → M14 → M8
4. **Physics Phase:** M8 → M2 → M4

11.3 State Management

- Most modules are stateless (M10, M11, M12, M13, M14, M15)
- State-holding modules use private fields with controlled access (M6, M7, M8, M9)
- No global variables - all state is encapsulated in components
- State persists via Unity's serialization (inspector-editable fields marked with `[SerializeField]`)

12 Timeline

This section details the development timeline for each module, in organized phases.

12.1 Development Phases

Phase 1: Core Cutting System (Completed)

- M10: Geometric Operations Module
- M11: Polygon Partitioning Module
- M12: Mesh Generation Module
- M14: Physics Update Module

Phase 2: User Interaction (Completed)

- M5: Player Control Module
- M6: Raycast Visualization Module
- M7: Cut Execution Module

Phase 3: Enhanced Features (Completed)

- M8: Debris Generation Module
- M9: Cut Profile Manager Module
- M15: Material Properties Module

Phase 4: Visual Polish (Not Complete)

- M13: Pixelation Module
- Visual feedback refinements
- Debug visualization tools

13 Notation

The structure of the Module Guide (MG) for modules follows Hoffman and Strooper (1995), with adaptations for Unity game engine components and 2D geometric operations. The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995) and standard computational geometry references. For instance, the symbol := is used for assignment statements and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

13.1 Primitive Data Types

The following table summarizes the primitive data types used by the Destructible Environment System.

Data Type	Notation	Description
character	char	A single symbol or digit
integer	$\mathbb{Z}$	A number without a fractional component in $(-\infty, \infty)$
natural number	$\mathbb{N}$	A number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	Any number in $(-\infty, \infty)$
boolean	$\mathbb{B}$	A truth value: true or false

13.2 Derived Data Types

The Destructible Environment System uses several derived data types: sequences, vectors, polygons, and tuples.

Sequences are lists filled with elements of the same data type, denoted as seq of T where T is the element type (ex. seq of Vector2 represents a sequence of 2D points).

Vectors represent 2D points or directions in space:

- Vector2 := $(x : \mathbb{R}, y : \mathbb{R})$

Polygons are sequences of vertices that form closed shapes:

- Polygon := seq of Vector2

Tuples contain a list of values and can be of different types. For example (Vector2, Vector2, $\mathbb{R}$) represents a tuple containing two 2D vectors and a real number.

Functions are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification. For example:

- distance: Vector2 $\times$ Vector2 $\rightarrow \mathbb{R}$

13.3 Geometric Notation

The following geometric primitives and operations are used throughout the system:

Symbol	Type	Description
P_{entry}	Vector2	Entry point where cutting ray intersects object surface
P_{exit}	Vector2	Exit point where cutting ray leaves object surface
V	seq of Vector2	Ordered sequence of vertices defining a polygon
C	Vector2	Centroid (geometric center) of a polygon
$\hat{d}$	Vector2	Normalized direction vector where $\|\hat{d}\| = 1$
$\hat{n}$	Vector2	Unit normal (perpendicular) vector to $\hat{d}$
$\theta(P, C)$	$\mathbb{R}$	Angle from centroid C to point P

Parameter	Type	Range	Description
σ	$\mathbb{R}$	$[0, 1]$	Softness coefficient: 0 = jagged, 1 = smooth
ψ	$\mathbb{R}$	$[0, 1]$	Strength coefficient: 0 = clean cut, 1 = max irregularity
n_{seg}	$\mathbb{N}$	$[3, 15]$	Number of segments along cut edge
δ_{max}	$\mathbb{R}$	$\mathbb{R}^+$	Maximum perpendicular offset from cut line

13.4 Cut Profile Parameters

Material cutting behavior is defined by the following parameters:

13.5 Physics Parameters

The following parameters govern the physical behavior of objects and debris:

Parameter	Type	Description
m	$\mathbb{R}^+$	Mass of an object in kilograms
A	$\mathbb{R}^+$	Area of a polygon in square units
v	Vector2	Linear velocity vector
ω	$\mathbb{R}$	Angular velocity in degrees per second
F_{exp}	$\mathbb{R}^+$	Explosion force magnitude
κ	$\mathbb{R}^+$	Mass multiplier constant for debris fragments

13.6 Operators and Functions

13.6.1 Logical Operators

- $\wedge$: logical AND
- $\vee$: logical OR
- $\neg$: logical NOT
- $\Rightarrow$: implies

13.6.2 Set Operators

- $\in$: element of
- $\cup$: union
- $\cap$: intersection
- $\subseteq$: subset of
- $|$: such that (set builder notation)

13.6.3 Geometric Functions

Side(L, P): Determines which side of line L a point P lies on

- Type: (Vector2 $\times$ Vector2) $\times$ Vector2 $\to \mathbb{R}$
- Returns: positive (right), negative (left), or zero (on line)
- Definition: $(L_{end}.x - L_{start}.x) \times (P.y - L_{start}.y) - (L_{end}.y - L_{start}.y) \times (P.x - L_{start}.x)$

Area(V): Calculates the area of a polygon using the shoelace formula

- Type: seq of Vector2 $\to \mathbb{R}^+$
- Definition: $\frac{1}{2}|\sum_{i=0}^{n-1}(v_i.x \times v_{i+1}.y - v_{i+1}.x \times v_i.y)|$

Distance(P_1, P_2)**:** Euclidean distance between two points

- Type: Vector2 $\times$ Vector2 $\to \mathbb{R}^+$
- Definition: $\sqrt{(P_2.x - P_1.x)^2 + (P_2.y - P_1.y)^2}$

Normalize(v): Converts vector to unit length

- Type: Vector2 $\to$ Vector2
- Definition: $v/\|v\|$ where $\|v\| = \sqrt{v.x^2 + v.y^2}$

Lerp(a, b, t): Linear interpolation between values a and b

- Type: $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ where $t \in [0, 1]$
- Definition: $a + (b - a) \times t$

Sort_CW(V): Sorts vertices in clockwise order around their centroid

- Type: seq of Vector2 $\to$ seq of Vector2
- Ordering based on increasing $\theta(v_i, C)$

Snap(P, p_{size})**:** Snaps point to nearest grid position

- Type: Vector2 $\times \mathbb{R}^+ \to$ Vector2
- Definition: $(\lfloor P.x/p_{size} + 0.5 \rfloor \times p_{size}, \lfloor P.y/p_{size} + 0.5 \rfloor \times p_{size})$

13.7 Conditional Notation

Conditional rules in specifications follow the format:

$(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$

where c_i represents a condition and r_i represents the corresponding result. The conditions are evaluated in order and the first true condition determines the result.

13.8 State Variables

State variables represent the mutable state of system components:

- V_{visual} : seq of Vector2 - Current visual mesh vertices
- $V_{collider}$: seq of Vector2 - Current physics collider vertices
- highlighted : $\mathbb{B}$ - Whether object is currently highlighted
- cultivated : $\mathbb{B}$ - Whether a valid cut is currently targeted

13.9 Module Access Constants

The following constants define access restrictions for module operations:

- **exported:** Operation is accessible to external modules
- **local:** Operation is only accessible within the module
- **read-only:** State can be read but not modified externally

References

[1] David L. Parnas. *On the criteria to be used in decomposing systems into modules.* Comm. ACM, 15(2):1053–1058, December 1972.

[2] David L. Parnas. *Designing software for ease of extension and contraction.* In ICSE '78: Proceedings of the 3rd international conference on Software engineering, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press.

[3] D.L. Parnas, P.C. Clement, and D.M. Weiss. *The modular structure of complex systems.* In International Conference on Software Engineering, pages 408–419, 1984.