# Summary

Survive from Scratch is a 3D open-world Role-Playing Game (RPG) including many Non-Player Characters (NPCs), Monsters, skills, crafting, building, and a demon lord.

The primary focus of the game is fighting, but other aspects such as building, completing quests, and talking to NPCs are significant aspects of the game.

The main goal of the game is to beat the demon lord, which requires collecting keys from four towers, each of which has many levels, to unlock.

This game has boss fights and challenges to give progression. Characters have a smart AI and can help you fight as allies or fight you as enemies.

*Plans to make the game massively multiplayer have been delayed indefinitely because multiplayer would take a long time to implement and would not be an important feature if there are less than ~100 global players. A smaller-scale coop mode might be added in the future with player-hosted rooms*.

Note: because C# only contains methods, both the words “method” and “function” refer to methods in this paper.

Contents

Summary 2

Definitions 2

World 3

Setting 3

Main World 3

Places 3

Towers 3

Levels 3

Teleportation 3

Items 4

Inventory 4

Crafting 4

Combat 5

Skills 5

Usable Skills 5

Ability Skills 5

Stats 5

HP 5

MP 5

ATK 5

Modifiers 5

Use 5

Typed Modifiers 5

Formula 6

Weapons 6

Armor 6

Characters 7

UI 8

Features 10

Fighting 10

Armor 10

Environments 10

Urban 10

Adventuring 10

Party 10

Crafting 11

Mechanics 11

Building 11

NPCs 11

Aggro 11

Enemies 11

Never ending 11

Stats 11

Health 11

Mana, etc. 11

Attack 12

Armor 12

Intelligence 12

# TDD Organization

This TDD is organized as follows:

Game engine is the heart of game development

Technologies describes software etc. used to create components of the game

# Game Engine

This game uses the Unity game engine. Unity is the most popular game engine, and I am familiar with it. Unity uses a system similar to an Entity-Component System. GameObjects are Unity’s entities. GameObjects are often used to represent what could be considered an “object” in layman’s talk. For example, a car might be represented by a GameObject with a Transform component describing the position, rotation, and scale of it. Unity’s GameObjects exist within a hierarchy, so the car GameObject might have child GameObjects for the body and each wheel. These child GameObjects would have Renderer components that queue the car wheel and body models for rendering.

Features of Unity include:

* Level design – simple drag-and-drop environment creation and GameObject system allows attaching Component script instances to virtual 3d GameObjects such as characters
* Rendering
* Physics engine
* Audio system
* Animation

Unity can compile and import C# scripts and assets of many different types, including images and 3d models.

# Technology

Many Assets, mostly 3D models and textures, were found online at the Unity Asset Store.

Level Design – Unity

3D Modelling – Blender

Image and Texture Editing – Photoshop and GIMP

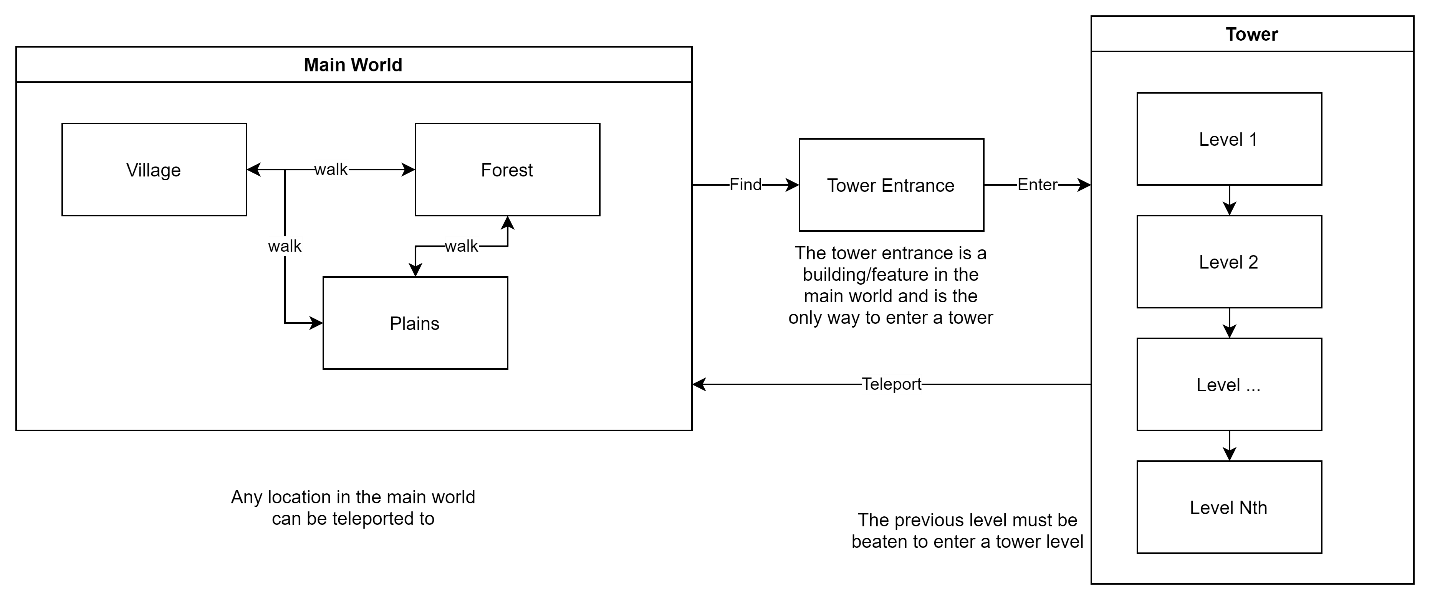
Audio Editing – Audacity

# Platforms

The main target platform for this game is Windows 10. However, the Unity game engine can build the game for many more platforms. Mac and Linux will likely be supported and updated for major updates. Because the mobile market is large, it is a possible future target. However, mobile games require lots of optimization and input design, while Mac and Linux might work with few changes.

# World Location Flowchart

The player can move and teleport between different locations as shown in the below chart. All parts of the Main World are connected “physically”; the player can walk continuously between them. Towers and their individual levels are not connected to the Main World or each other “physically”; traveling to and from them is always done through teleportation.



# Save

## Entities

Most elements in the game to be saved are classified as Entities. These include players, enemies, destroyable trees, and destroyable buildings. Each Entity is assigned an id and has Components that save. Not all entities have all types of components.

Component examples:

* SaveEntity – saves the basic data like position and rotation.
* StatScript – saves hp, mp, xp, etc. of the entity.
* Inventory – saves the items (materials, weapons, consumables, etc.) in the entity’s inventory.

# Script Organization

## Control

Control scripts are singleton classes that handle features. Control scripts derive from Unity’s Component class and are thus attached to empty GameObjects.

Examples:

* GameControl – handles teleporting and loading world locations.
* ProgressTracker – keeps track of statistics including number of monsters killed by type, total damage dealt, and active quests (including updating and saving).
* SettingsControl – handles the settings menu and applies setting changes.
* HelpControl – handles the help menu and showing specific help topics.

## Character

Instances of character scripts are attached to character GameObjects. These scripts handle features that are managed per character.

Examples:

* StatScript – handles the stats (hp, mp, xp, etc.) of the character and handles stat skills.
* Movement – handles moving the player in a desired direction set by other components.
* NPCControl – handles setting the desired movement direction and rotation of Movement.
* Loot – drops loot when the character dies.

## Item

Instances of item scripts are attached to item GameObjects. Item GameObjects are 3d representations of items that appear when the character equips an item or when an item is dropped in the world.

Examples:

* Droppable – allows and handles dropping an equipped item whose GameObject it is attached to.
* Collectible – allows an item dropped in the world to be picked up by the player.
* Consumable – allows an equipped item to be consumed.

## Quest

Instances of quest scripts represent quests for the player. ***Unlike most other categories of scripts described in this section, quest scripts do not derive from Unity’s Component. Quest classes follow a more object-oriented design pattern.*** For each type of quest, there is a quest class. Quest objects are serialized to JSON and saved by ProgressTracker.

Examples:

* KillQuest – a quest to kill a certain type of entity.
* TalkQuest – a quest to talk to a certain NPC.
* LocationQuest – a quest to reach a certain location.

Simple quests like these are strung together in a linear sequence of quests implemented in the ComplexQuest class.

All types of quests, including ComplexQuest implement the IQuest interface. IQuest includes common functions.

Examples of IQuest functions:

* GetQuestName – returns the name of the quest.
* GetDescription – returns a description of the quest, including progress if relevant.
* OnEntityKilled – called when an entity is killed by the player.
* OnTalked – called when the player talks to a NPC.

The KillQuest class must implement all of these functions, but since talking to a NPC is logically irrelevant to killing an enemy, the KillQuest class leaves the OnTalked method empty.