Technical Design Document

SUSUCUBE

Team W4
Team SuS

Team Information

Developers

Tay Hao Cheng	--	183322F	--	Team Leader / Programmer
Ryan Tan	--	182414L	--	Programmer, Scribe
Nam Kai Zhe	--	180716S	--	Programmer
Lorenzo	--	173582Y	--	Programmer

Project Overview

Game Concept

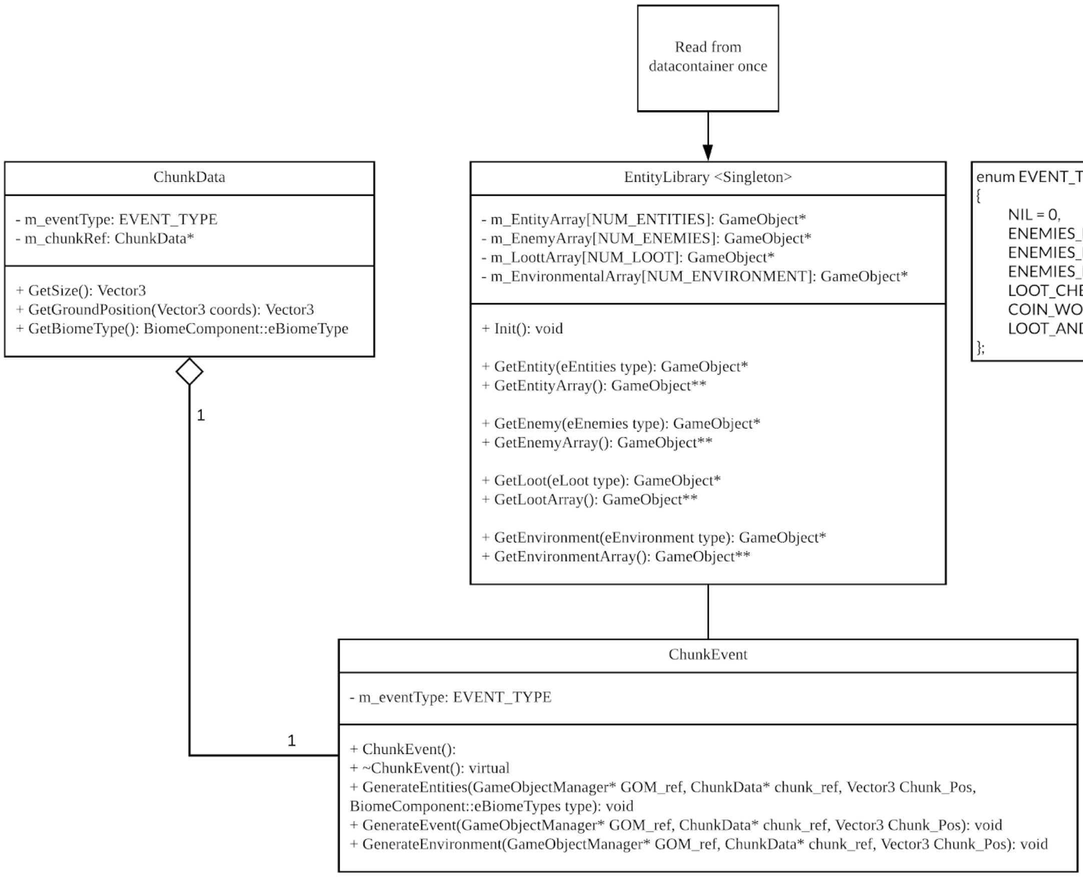
SusuCube is a top down shooter through a procedurally generated world. The world is generated through chunks/rooms. Each chunk is an open exterior area with different biomes. Each chunk can have different weather which affects both visuals, gameplay and enemies. Players combat enemies with customisable guns. Enemies drop gun parts which can be attached to the current weapon during gameplay. The player will be able to interact with objects in the world to create/move/hold cover.

Gameplay

Map

Chunks, sized in increments of 16 on the x and z-axes, are loaded from chunk files, which contain the size and contents of the chunk in an extensible file format. They are placed on a grid where each chunk is a multiple of 16 unity apart from the next.

When a chunk is generated, it will generate environmental objects & an event that takes place on that tile. An event may be the spawning of enemies, chests or both.


Read from datacontainer once
ChunkData
- m_eventType: EVENT_TYPE
- m_chunkRef: ChunkData*
+ GetSize(): Vector3
+ GetGroundPosition(Vector3 coords): Vector3
+ GetBiomeType(): BiomeComponent::eBiomeType
EntityLibrary <Singleton>
- m_EntityArray[NUM_ENTITIES]: GameObject*
- m_EnemyArray[NUM_ENEMIES]: GameObject*
- m_LoottArray[NUM_LOOT]: GameObject*
- m_EnvironmentalArray[NUM_ENVIRONMENT]: GameObject*
+ Init(): void
+ GetEntity(eEntities type): GameObject*
+ GetEntityArray(): GameObject**
+ GetEnemy(eEnemies type): GameObject*
+ GetEnemyArray(): GameObject**
+ GetLoot(eLoot type): GameObject*
+ GetLootArray(): GameObject**
+ GetEnvironment(eEnvironment type): GameObject*
+ GetEnvironmentArray(): GameObject**
enum EVENT_TYPE
{
NIL = 0,
ENEMIES_LEVEL_1,
ENEMIES_LEVEL_2,
ENEMIES_LEVEL_3,
LOOT_CHEST,
COIN_WONDER
LOOT_AND_ENEMIES,
};
1
1
ChunkEvent
- m_eventType: EVENT_TYPE
+ ChunkEvent():
+ ~ChunkEvent(): virtual
+ GenerateEntities(GameObjectManager* GOM_ref, ChunkData* chunk_ref, Vector3 Chunk_Pos, BiomeComponent::eBiomeTypes type): void
+ GenerateEvent(GameObjectManager* GOM_ref, ChunkData* chunk_ref, Vector3 Chunk_Pos): void
+ GenerateEnvironment(GameObjectManager* GOM_ref, ChunkData* chunk_ref, Vector3 Chunk_Pos): void

Game Objects

The following is a list of logical objects planned for the game:

- Guns: primary weapon of the player, shoots bullets and is modifiable
- Gun parts: attachable objects to the gun. Will wear out with use.
    - Scope/Muzzle: affects horizontal recoil

    - Muzzle: affects fire rate
    - Clip: affects max ammo per magazine
    - Parts may come with an Augment: A special object that may spawn with the weapon part that can affect the bullets fired or the player’s stats
- Cover: object that takes damage over time from enemies, can be moved by player
- Treasure Chest: Drops a random weapon part and despawns afterwards

Characters:

- Character name: Description of character

Physics / Collision

There are 2 collider types.

1. Ball
2. Chunk

Collisions are only check between ball and other collider type. (i.e chunks do not collide with each other).

Ball and ball collisions calculate exit velocity based on transfer of momentum.

Chunk and ball collisions truncate the ball to its axis-aligned bounding box (AABB) and perform collision checking based on the AABB and the chunk.

Player Actions

Movement

Character moves UDLR on the x-z axis. Character can jump in the y axis.

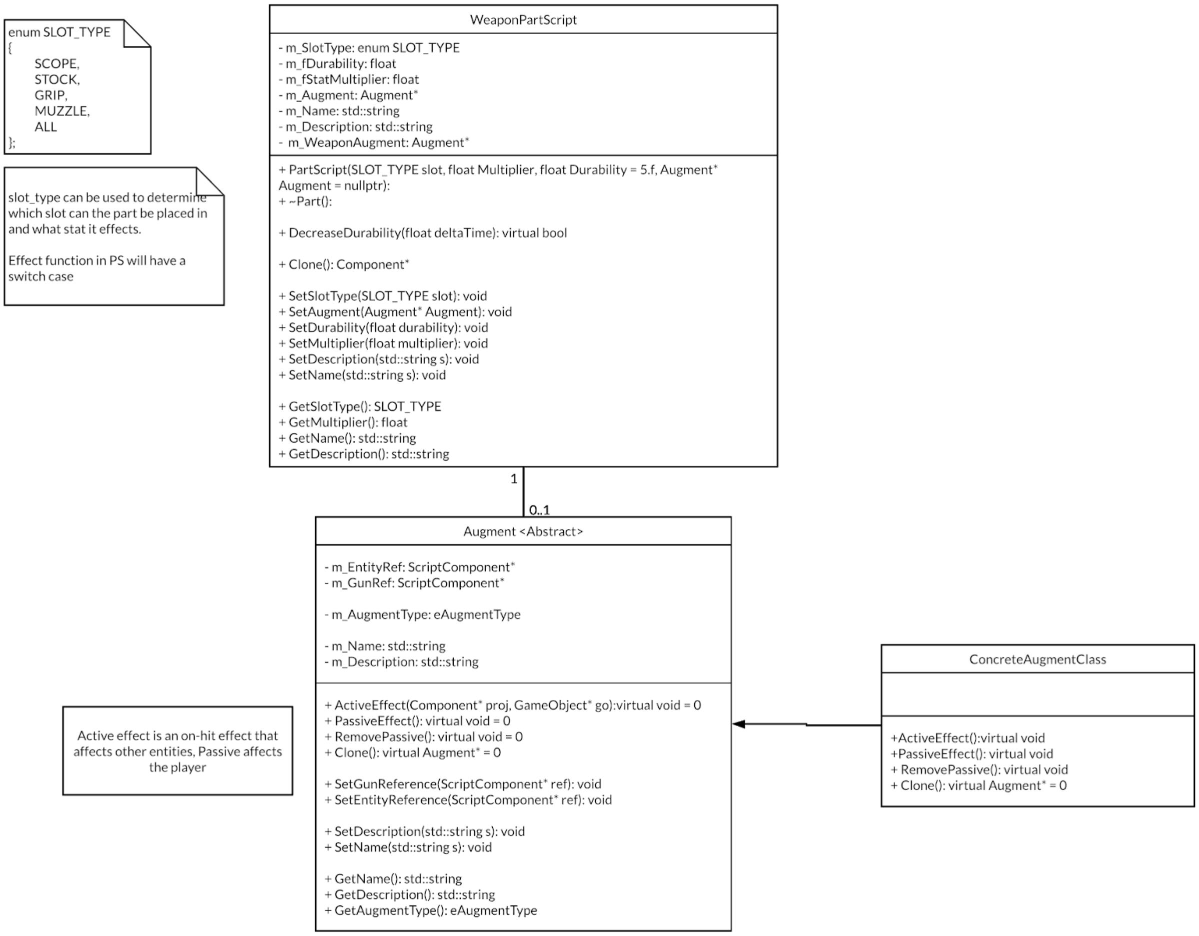
Character has addition movement states:

- Sprint: Increase movement speed. Unable to fire nor customise gun. Increase detection radius.
- Dodging: Quick move in reticle direction. U

Interaction

Players will be able to interact with various world objects to create cover for themselves, or to place an offensive object. I.e player can flip tables to create a destructible cover for themselves. Players will also be able to pick up weapon parts on the ground by colliding into them.

Weapon Customisation


enum SLOT_TYPE
{
SCOPE,
STOCK,
GRIP,
MUZZLE,
ALL
};
WeaponPartScript
- m_SlotType: enum SLOT_TYPE
- m_fDurability: float
- m_fStatMultiplier: float
- m_Augment: Augment*
- m_Name: std::string
- m_Description: std::string
- m_WeaponAugment: Augment*
+ PartScript(SLOT_TYPE slot, float Multiplier, float Durability = 5.f, Augment* Augment = nullptr):
+ ~Part():
+ DecreaseDurability(float deltaTime): virtual bool
+ Clone(): Component*
+ SetSlotType(SLOT_TYPE slot): void
+ SetAugment(Augment* Augment): void
+ SetDurability(float durability): void
+ SetMultiplier(float multiplier): void
+ SetDescription(std::string s): void
+ SetName(std::string s): void
+ GetSlotType(): SLOT_TYPE
+ GetMultiplier(): float
+ GetName(): std::string
+ GetDescription(): std::string
1
0..1
Augment <Abstract>
- m_EntityRef: ScriptComponent*
- m_GunRef: ScriptComponent*
- m_AugmentType: eAugmentType
- m_Name: std::string
- m_Description: std::string
+ ActiveEffect(Component* proj, GameObject* go):virtual void = 0
+ PassiveEffect(): virtual void = 0
+ RemovePassive(): virtual void = 0
+ Clone(): virtual Augment* = 0
+ SetGunReference(ScriptComponent* ref): void
+ SetEntityReference(ScriptComponent* ref): void
+ SetDescription(std::string s): void
+ SetName(std::string s): void
+ GetName(): std::string
+ GetDescription(): std::string
+ GetAugmentType(): eAugmentType
ConcreteAugmentClass
+ActiveEffect():virtual void
+PassiveEffect(): virtual void
+ RemovePassive(): virtual void
+ Clone(): virtual Augment* = 0

slot_type can be used to determine which slot can the part be placed in and what stat it effects.

Effect function in PS will have a switch case

Active effect is an on-hit effect that affects other entities, Passive affects the player

For Weapon customisation, the player will be equipped with a game object with a gun script will have 4 vectors weapon part pointers which correspond to the 4 types of weapon parts available in the game , the stats fired by the weapon equipped is affected by the type/number of parts equipped. There are no set limitations of how many parts of one type can be equipped, the stats will just stack. In addition to this, parts may come with 1 augment object attached. The augment affects the parameters of the bullets fired or the player script.

Part Types:

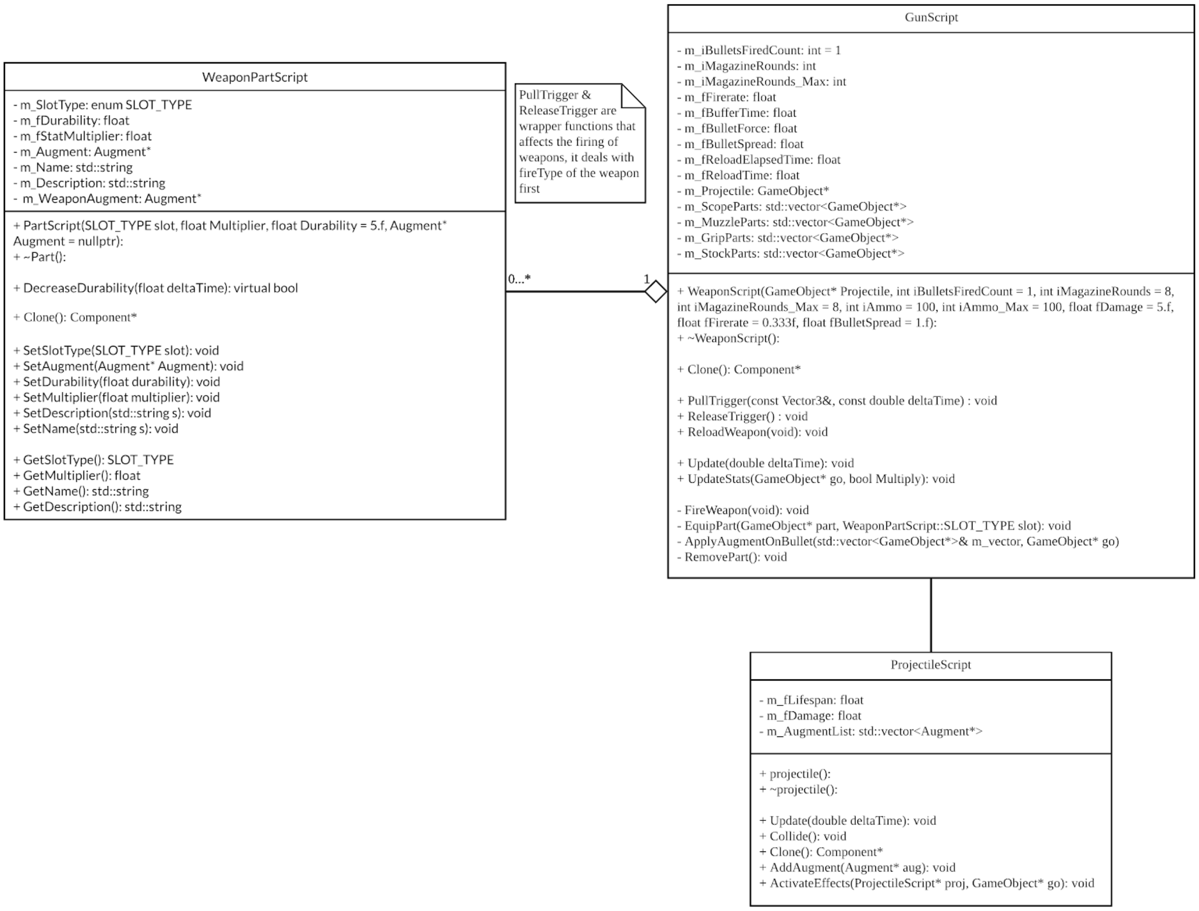
- Muzzle: affects bullet spread
- Magazine: affects magazine rounds
- Scope/Grip: affects fire rate

Augment Types:

- Black Hole Augment: When a bullet hits an entity, the bullet becomes a black hole that lasts for a second..
- Explode Augment: When a bullet hits an entity, the bullet explodes, pushing back the target struck and dealing more damage.
- Reload Augment: Reload time is shortened.

Combat

The player will be handling a gun when he/she engages in combat. The gun faces the reticle constantly and fires bullets in the direction of the reticle.

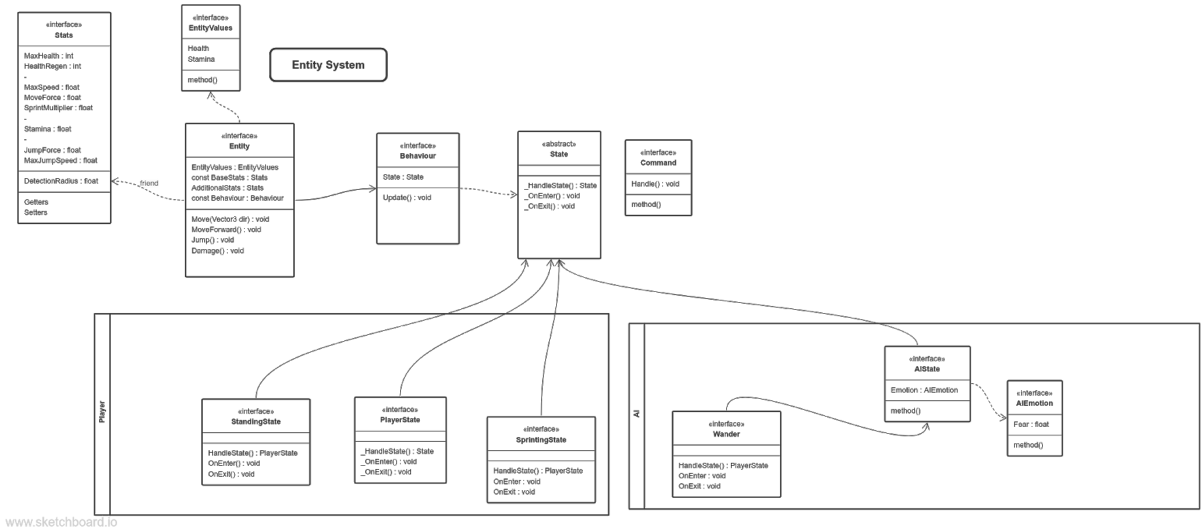

WeaponPartScript
- m_SlotType: enum SLOT_TYPE
- m_fDurability: float
- m_fStatMultiplier: float
- m_Augment: Augment*
- m_Name: std::string
- m_Description: std::string
- m_WeaponAugment: Augment*
+ PartScript(SLOT_TYPE slot, float Multiplier, float Durability = 5.f, Augment* Augment = nullptr):
+ ~Part():
+ DecreaseDurability(float deltaTime): virtual bool
+ Clone(): Component*
+ SetSlotType(SLOT_TYPE slot): void
+ SetAugment(Augment* Augment): void
+ SetDurability(float durability): void
+ SetMultiplier(float multiplier): void
+ SetDescription(std::string s): void
+ SetName(std::string s): void
+ GetSlotType(): SLOT_TYPE
+ GetMultiplier(): float
+ GetName(): std::string
+ GetDescription(): std::string
0...*
1
GunScript
- m_iBulletsFiredCount: int = 1
- m_iMagazineRounds: int
- m_iMagazineRounds_Max: int
- m_fFirerate: float
- m_fBufferTime: float
- m_fBulletForce: float
- m_fBulletSpread: float
- m_fReloadElapsedTime: float
- m_fReloadTime: float
- m_Projectile: GameObject*
- m_ScopeParts: std::vector<GameObject*>
- m_MuzzleParts: std::vector<GameObject*>
- m_GripParts: std::vector<GameObject*>
- m_StockParts: std::vector<GameObject*>
+ WeaponScript(GameObject* Projectile, int iBulletsFiredCount = 1, int iMagazineRounds = 8, int iMagazineRounds_Max = 8, int iAmmo = 100, int iAmmo_Max = 100, float fDamage = 5.f, float fFirerate = 0.333f, float fBulletSpread = 1.f):
+ ~WeaponScript():
+ Clone(): Component*
+ PullTrigger(const Vector3&, const double deltaTime) : void
+ ReleaseTrigger() : void
+ ReloadWeapon(void): void
+ Update(double deltaTime): void
+ UpdateStats(GameObject* go, bool Multiply): void
- FireWeapon(void): void
- EquipPart(GameObject* part, WeaponPartScript::SLOT_TYPE slot): void
- ApplyAugmentOnBullet(std::vector<GameObject*>& m_vector, GameObject* go)
- RemovePart(): void
ProjectileScript
- m_fLifespan: float
- m_fDamage: float
- m_AugmentList: std::vector<Augment*>
+ projectile():
+ ~projectile():
+ Update(double deltaTime): void
+ Collide(): void
+ Clone(): Component*
+ AddAugment(Augment* aug): void
+ ActivateEffects(ProjectileScript* proj, GameObject* go): void

PullTrigger & ReleaseTrigger are wrapper functions that affects the firing of weapons, it deals with fireType of the weapon first

The gun has the following behaviours:

- Auto-firing bullets
- Unlimited ammunition, magazine based
- Delayed Reloading
- Bullet spread

Entity System / AI



Entity class is a base class for movement. Behaviours interact with entity.
Each behaviour has a state.
Player has 2 states. Standing and sprinting.
Movement is called through a MovementCommand, child of Command.
AIEntity is a subclass of Entity that contains additional data for AI such as target direction and move towards.
AI uses the same behaviour, but with different states.
AI follow a state system with 3 main states. Wander, idle, and combat. Combat state is triggered when entity is near player. Combat state can be melee, ranged, boss or flee.

Victory conditions

There are 3 waves of bosses. Players must survive and defeat all 3 for victory.

Graphics

Presets

2 preset options, LOW and HIGH. LOW should be used for development to reduce load times. All assets should be available in HIGH.
While running in LOW, if asset is not available, use HIGH asset.

Shaders

Fog shader
Improved fog / mist shader.

- Dissipate on y height.
- Around player instead of camera
- Noise of min dist
- Fog density based on vertex pos and time to create an uneven surface

CRT Shader

User Interface

Inventory screen

A constant UI element detailing the contents of inventory. Collectible items include only weapon parts.

Screen flow

- Login Screen
  - Username
  - Pass
  - Create
  - Login
- Main Menu
  - Play
  - Options
  - Controls
  - Quit
- Main Game
  - Win Screen
    - Return to main menu
  - Lose Screen

        - Return to main menu

Audio

2 main background track for when play is at high health and low health.

Code Overview

File format

Commenting

Naming Conventions

CC == Camel Case
U == First letter upper case
L == First letter lower case

Variables

```
s_v_camelCaseWithLowercaseFirstLetter
Global variables prefixed with g_
Static variables prefixed with s_
Member variables prefixed with m_
```

Example:

```
String s_studentName = “boi”;
int -> i
unsigned -> u
float -> f
double -> d
bool -> b
char -> c
string -> s
std::vector<{type}> -> vec_type
e.g. ivec_List
```

Maps

```
std::map<{type}> -> map_type
e.g. map_Meshes
String key values for map are CCU

Vector3 -> v
```

Class objects omit the type

Ex

```
     Person PersonA;
```

Functions

```
void ThisIsAFunction(void)
{

}
```

Never use inline braces

Enums

Enums should be CCU but start with a lower case e.

Keys should be caps with underscores

Ex

```
     eDirections{UP,DOWN,LEFT,RIGHT};
```

Source-Control

Branching

Branch names should follow the path of SP3/

Feature branch will be in SP3/Feature

Bugfix will be in SP3/Bugfix

Build will be in SP3/Build

Builds are for release build for testing/presentation

Existing build branches include Testing, Working.

Flow

Devs first merge current testing branch into feature branch.

Once merge conflicts are solved and build is running, merge into testing branch.

Once build is tested and stable, merge to working branch.

Working branch is for build meant for presentation/supervisor meets. It is the latest stable version.

Project Schedule

Gantt Chart

Task List

Legend

- Removed: 🟥
- Added: 🟦

Map Generation

- Map Generation (Kai Zhe)
    - Map Tile Types
    - Map Loader
        - Map File Format
    - Map Visualiser Tool
    - Template Joining
    - Terrain Collider
    - Occlusion / Player Visibility
- Map Design
    - Level Design
    - Difficulty Scaling
- Biomes (Lorenzo)
    - Preset terrains
    - Defining types of biomes
    - Foliage spawning
        - Interaction between foliage and player
            - environmental hazards
            - For Player Support (e.g. Cover)
    - Textures to differentiate models in different biomes
    - Particle System
        - Weather effects
        - From foliage
    - Spawning Structures
    - Transitioning between different biomes
- Weathers (Lorenzo)
    - Rain
    - Snow

UI

- Inventory (Hao Cheng)
    - 6 slots
- Weapon Slots (Hao Cheng)
    - 4 Slots for the 4 kinds of weapon parts
- Player UI (Hao Cheng)
    - Health Bar
    - Stamina Bar
    - Magazine Rounds
- Boss Meter (Hao Cheng)
    - Fills up over time, fills faster if player is stationary

Menu

- Main Menu
- Login Menu

Player Data

- Username
- Password
- Abilities unlocked
- coins

Weapon

- Weapon (Ryan Tan)
    - Weapon handling
        - Fire
        - Reload
        - Switching weapons (Maybe)
    - Bullet collision
        - Collision with rigidbody
        - Collision with terrain
    - Aggregated Parts (can be equipped)
        - Types of parts
        - It affects stats of the weapon equipped
        - Durability of parts
        - Augments (may come with the part)
            - Affects player/bullet fired
    - Bullet effects
        - Single-target/AOE Damage
        - Push-back
- Grenade (Ryan Tan)
    - Grenade Physics

        - Curved motion
    - Grenade collision
        - Collision with terrain
        - Collision with rigidbody
    - Grenade effects
        - AOE damage
        - Push-back

Entities

- Player
    - Health
    - Stamina
    - Movement
        - WASD Movement
        - Jump
        - Crouch
        - Sliding
        - Dodge
            - Slow down time for quick dodge
        - Sprint
    - Inventory
    - Movement
    - Attack
- Interactables
    - Flipping cover
    - Giant boulder
    - Loot box
- Enemy
    - AI
    - Loot Drop
        - Percentage based
        - May drop a weapon part that may come with an augment

UI

- Button System
- Inventory
- Magazine Rounds
- Player stats
  - Health
  - Stamina
- Abilities
- Reload

Game Logic

- Enemy Spawning
- Loot dropping
- Win condition
  - Endless Shooter
  - Defeat 3 stages of boss
- Lose Situation

Project Schedule

Legend

- Removed: 🟥
- Added: 🟦
- Implemented but removed: 🟧

- Week 1
  - Player movement
  - UI elements
  - Weapons firing
  - Weapon customisation
  - Unique Weapon parts
  - Map Generation
  - Terrain Collider
  - Biomes
- Week 2
  - Enemy AI
  - Enemy Spawning
  - Particles
  - Map Visualiser
  - Interactables
  - Occlusion
  - Weather
  - Generic weapon parts
  - Weapon Augment
  - Chunk Events
  - Loot Dropping
- Week 3
  - Debugging / Game Testing
  - Preparation of documentation
  - Preparation of presentation
  - Ability System