Technical Design Document

Nate

Team Sign Off	
Nate Evans	NE
Alvina Angelin	AA
Haruki Uda	HU
Aron Sanchez	AS

Coding standards

- Maximum of one class per file
- Tabs for indentation
- Max line length 100 characters
- Open brackets placed below the opening statement
- Camel case variable names
- Underscore case asset names
- Imperative commit messages

Developer context

- IDE: Visual Studio
- Language: C++
- Version Control: Git with GitHub
- Target platforms: Windows
- Required technology: GPFramework

Technical challenges

- Time manipulation: To implement the mechanic where game time progresses based on player input, we will use a system where the game's simulation time scales with the player's movement. The time factor can be controlled by the player's input intensity (from 0 to max).
- Collision detection: Use rectangle2D to compare collision between different entities.
- Rendering: Uses static and animated sprites to display game elements.
- Procedural level generation: As a stretch – dynamically generate levels based on provided templates, ensuring the levels all remain playable.

Architecture

- Physics Engine: Custom time-dilation logic
- Audio Engine: FMod-API-Core
- Time Dilution Manager: dilutes game time advancement based on user input delta.
- Physics System: controls bullet physics

- Collision System: rectangle collision between entities

Development methodology

- Agile (adjusted to fit our timeline): weekly meetings and regular check ins on Discord

Third-party libraries

- SDL2 library
- Fmod sound engine

UML diagrams

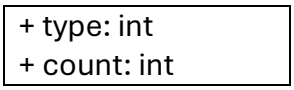
Base
+ Initialise() + Process() + Draw()

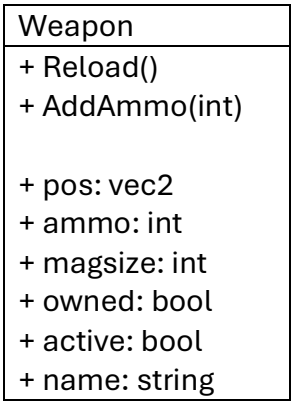
Entity : Base
+health: int +maxHealth: int +isAlive: bool +hitbox: struct +position: vec2 +velocity: vec2

Player : Entity
+ curWeapon: int

Enemy : Entity
+ damage: int + target: Player + isAngry: bool

Ammo: Entity


+ type: int
+ count: int


Weapon
+ Reload()
+ AddAmmo(int)
+ pos: vec2
+ ammo: int
+ magsize: int
+ owned: bool
+ active: bool
+ name: string