## horizontal line



TDD Template

XX.XX.20XX

**─**

Your Name

Your Company

123 Your Street

Your City, ST 12345

# 

# Changelog

|  |  |  |
| --- | --- | --- |
| Version | Date | Changes |
| 1.0.0 | XX/XX/20XX | Initial Setup |
|  |  |  |
|  |  |  |
|  |  |  |

# Contents

[**Changelog**](#_5kn5bjry9gjp) **1**

[**Contents**](#_62o0kgg5oynz) **2**

[**Introduction**](#_oxk53rrzhe3j) **3**

[Rationale](#_sui8vh2z74cg) 3

[Background](#_ht161vq8mk3m) 3

[Terminology](#_je3u68g7apd6) 3

[Non-Goals](#_8q92eeds6myf) 3

[Proposed Design](#_a10xfv7dn01b) 3

[Software and Hardware Requirements](#_t60nfhfpuxia) 3

[**Gameplay**](#_jkl76wqkyn7y) **4**

[Gameplay Mechanics](#_vmoqexbqk7tf) 4

[Mechanic #1](#_we558vptzf6r) 4

[Mechanic #2](#_k07jf3o5x9cx) 4

[Mechanic #3](#_nw6w6su1lzyq) 4

[Mechanic #4](#_eup3qr74kmzw) 4

[Controls](#_m6z7rbcrryjc) 4

[Mappings](#_sskpmr5w3d3t) 4

[**System Architecture**](#_cwo5vxww8gb) **5**

[Data types](#_4sgi0ird3xzv) 5

[Data Model](#_7ono6u1cvktu) 5

[Interface/API Definitions](#_cga8s6z1ahbo) 5

[Impact](#_i80vs9qe7l24) 5

[Risks](#_230o9g1wesmu) 5

[Alternatives](#_9m502pllby8o) 5

[**Shader**](#_miq9noo9xeix) **6**

[Shader types](#_3wyw3b4cesv3) 6

[Render modes](#_o9291nq9ijt3) 6

[Processor functions](#_wph6am6x3u0b) 6

[Vertex processor](#_ne1x1wyliilo) 6

[Fragment processor](#_4nrb129bye6v) 6

[Light processor](#_tzyu70k510p2) 6

# Introduction

## Rationale

I am trying to accomplish a 3-door maze with a start and finish as well as a collectible to allow for the end of the maze.

## Background

There may be some references to the TDD for ‘AI Assignment 2’ as they use some of the same NavMesh components.

## Terminology

No Special terminology is used.

## Non-Goals

I have not yet figured out how the game ending collectible will be obtained and how it will ultimately allow the maze to be ‘Completable’

## Proposed Design

I am thinking for allowing the ending to put a waypoint towards the end game ‘Key’ which will contain a box collider ‘isTrigger’ and will SetActive the final waypoint. Once the final waypoint is reached it will trigger the end game message.

## Software and Hardware Requirements

Software - Unity

Hardware - Any PC (Post 2007 recommended)

# Gameplay

## Gameplay Mechanics

### Mechanic #1 - NavMesh Agent

/ The NavMesh Agent will simply follow waypoints and will be considering any obstacles in its path where it would then avoid that certain object and find a new route to take. /

### Mechanic #2 – End Game Key

/ To Achieve the end goal, you must retrieve the key to enable the ending of the maze. /

### Mechanic #3 - Doors

/ Doors will be around the maze with buttons in their vicinity that will activate the door enabling it to be opened to venture through the maze toward the end. /

### Mechanic #4

/ Detailed description of how the mechanic will work /

## Controls

/ There are no controls available, there is a simple Birdseye view camera /

### Mappings

|  |  |  |  |
| --- | --- | --- | --- |
| **Control** | **Function** | **Device** | **Mappable** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 

# System Architecture

/ If the design consists of a collaboration between multiple large-scale components, list those components here — or better, include a diagram [UML]. /

## Data types

/ Describe the data types you will be using and how they work. /

## Data Model

/ Describe how the data is stored and used. /

## Interface/API Definitions

/ Describe how the various components talk to each other. For example, if there are REST endpoints, describe the endpoint URL and the format of the data and parameters used. /

## Impact

/ Describe the potential impacts of the design on overall performance, security, and other aspects of the system. /

## Risks

/ If there are any risks or unknowns, list them here. Also, if there is additional research to be done, mention that as well. /

## Alternatives

/ If there are other potential solutions which were considered and rejected, list them here, as well as the reason why they were not chosen. /

# Shader

## Shader types

/ Instead of supplying a general purpose configuration for all uses (2D, 3D, particles), Godot shaders must specify what they are intended for. Different types support different render modes, built-in variables, and processing functions. /

## Render modes

/ Different shader types support different render modes. They are optional and, if specified, must be after the shader\_type. Render modes are used to alter the way built-in functionality is handled. For example, it is common to use the render mode unshaded to skip the built-in light processor function. /

## Processor functions

/ Depending on the shader type, different processor functions may be optionally overridden. For “spatial” and “canvas\_item”, it is possible to override vertex, fragment, and light. For “particles”, only vertex can be overridden. /

## Vertex processor

/ The vertex processing function is called once for every vertex in “spatial” and “canvas\_item” shaders. For “particles” shaders, it is called once for every particle. /

## Fragment processor

/ The fragment processing function is used to set up the Godot material parameters per pixel. This code runs on every visible pixel the object or primitive draws. It is only available in “spatial” and “canvas\_item” shaders. /

## Light processor

/ The light processor runs per pixel, but also runs for every light that affects the object (and does not run if no lights affect the object). It exists as a function called inside the fragment processor and typically operates on the material properties setup inside the fragment function. /