

**Assessment 2 - Develop Cross-platform Game**

11/04/2022

**─**

Alexander Burton

# 

# Changelog

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 1.0.0 | 11/04/2022 | Initial Setup: fill in document to comply with assessment requirements. |
| 1.0.1 | 15/04/2022 | Document additions |
| 1.0.2 | 22/5/2022 | Document additions |
| 1.0.3 | 26/05/2022 | Document additions |
| 1.0.4 |  |  |
|  |  |  |
|  |  |  |

# Contents

[Changelog 2](#_Toc77427064)

[Contents 3](#_Toc77427065)

[Introduction 5](#_Toc77427066)

[Rationale 5](#_Toc77427067)

[Background 5](#_Toc77427068)

[Terminology 5](#_Toc77427069)

[Non-Goals 5](#_Toc77427070)

[Proposed Design 5](#_Toc77427071)

[Software and Hardware Requirements 5](#_Toc77427072)

[Gameplay 6](#_Toc77427073)

[Gameplay Mechanics 6](#_Toc77427074)

[Mechanic #1 6](#_Toc77427075)

[Mechanic #2 6](#_Toc77427076)

[Mechanic #3 6](#_Toc77427077)

[Mechanic #4 6](#_Toc77427078)

[Controls 6](#_Toc77427079)

[Mappings 6](#_Toc77427080)

[System Architecture 7](#_Toc77427081)

[Data types 7](#_Toc77427082)

[Data Model 7](#_Toc77427083)

[Interface/API Definitions 7](#_Toc77427084)

[Impact 7](#_Toc77427085)

[Risks 7](#_Toc77427086)

[Alternatives 7](#_Toc77427087)

[Shader 8](#_Toc77427088)

[Shader types 8](#_Toc77427089)

[Render modes 8](#_Toc77427090)

[Processor functions 8](#_Toc77427091)

[Vertex processor 8](#_Toc77427092)

[Fragment processor 8](#_Toc77427093)

[Light processor 8](#_Toc77427094)

# 

# Introduction

## Rationale

Create a game that is designed for mobile but is also compatible with PC. The cross-platform design will be outlined below.

## Background

Cross-platform games are becoming more and more common with the use of cloud to sync content. This ability to put down one device and pic up another has led to the user expectation that games will have these features as standard. The design of a game that works on different platforms is the first stage of platform development. Following this it will be testing then implementation of the testing.

## Terminology

**AI** – or artificial intelligences is the references to a collection of algorithms that allow for decision on how to best complete a set task.

**Algorithm** – refer to the step-by-step instructions that aid the AI in completing a calculation.

**CPU** – Central processing system, is the motherboard is the brains pathways of a computer the CPU is the speed at which the brain thinks.

**Games** – a collection of graphics and input response algorithms that allow players to escape from reality.

**Hardware** – The electronic items that run software.

**Lexicographical ­**– is the ordering of numbers and letters in order of total number of items in order of order. So bab, aab, aba, bba, baa, would be ordered as follows aab, aba, baa, bab, bba. Same with number sets. 211, 221, 121, 112, 111, 122, 222, the set would be ordered as 111, 112, 121, 211, 212, 221, 222.

**Motherboard** – This is the brains of all computing units it is what allows all the other hardware items to talk to each other.

**Operating system** – A term used as a reference to a hardware platforms user interface.

**Reality** – a conceptual space where both logic and reason give way to personal gain and the erosion of self-worth.

**Software** – programs or instructions that allow users to execute tasks in an efficient manner.

**State machine** - allows character to engaged in some kind of action at any given time.

**Unity** – a game engine used in the created and deployment of games.

## Non-Goals

Allowing the game to have the ability to sync saves across user’s devices would be desirable but for this stage of the design it is not needed.

## Proposed Design

/ Start with a brief, high-level description of the solution. The following sections will go into more detail. /

## Software and Hardware Requirements

As unity is the development platform for the created of the inventory system, the following software and hardware requirements are recommended by unity. As Rubber Ducky Studios only uses Windows PCs the below information shall only relate to that system.

Unity Editor system requirements for windows PC;

* **Operating system version:** Windows 7 (SP1+) and Windows 10, 64-bit versions only.
* **CPU:** X64 architecture with SSE2 instruction set support.
* **Graphics API**: DX10, DX11, and DX12-capable GPUs.
* **Additional requirements:** Hardware vendor officially supported drivers. For development: IL2CPP scripting backend requires Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.

# Gameplay

## Gameplay Mechanics

### Mechanic #1

/ Detailed description of how the mechanic will work /

### Mechanic #2

/ Detailed description of how the mechanic will work /

### Mechanic #3

/ Detailed description of how the mechanic will work /

### Mechanic #4

/ Detailed description of how the mechanic will work /

## Controls

/ Brief description of how the controls will work (are they mappable or are they static?) /

### 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. /