# Customisation TDD

## V\_00.01

## Author: Finn Varvell

# Introduction

## Rationale

This TDD is for implementing a character customisation system. This will, when a new game is loaded, allow the player to customise how their character looks, change the character’s name and class and use points to add to character stats. All the customisation done will be loaded into the game and directly affect the character in game. It will also save to a binary file so it can be loaded again when the player exits and continues.

## Background

Character customisation systems are systems in games that let the player customise their character how they want. They are most commonly in RPGs and games in which characters have more stats the player can see and change, and they can improve the experience of the game in multiple ways. By customising the look of their own character and naming them, the player is more likely to feel like they are controlling a character of their creation rather than one of the game developers’ creation and the experience will be more unique when replaying the game with a new character. Customising the stats and class of their character also lets the player feel more in control of their growth and abilities.

## Terminology

TDD – Technical design document

RPG – Role playing game

UI – User interface

GUI – Graphical user interface

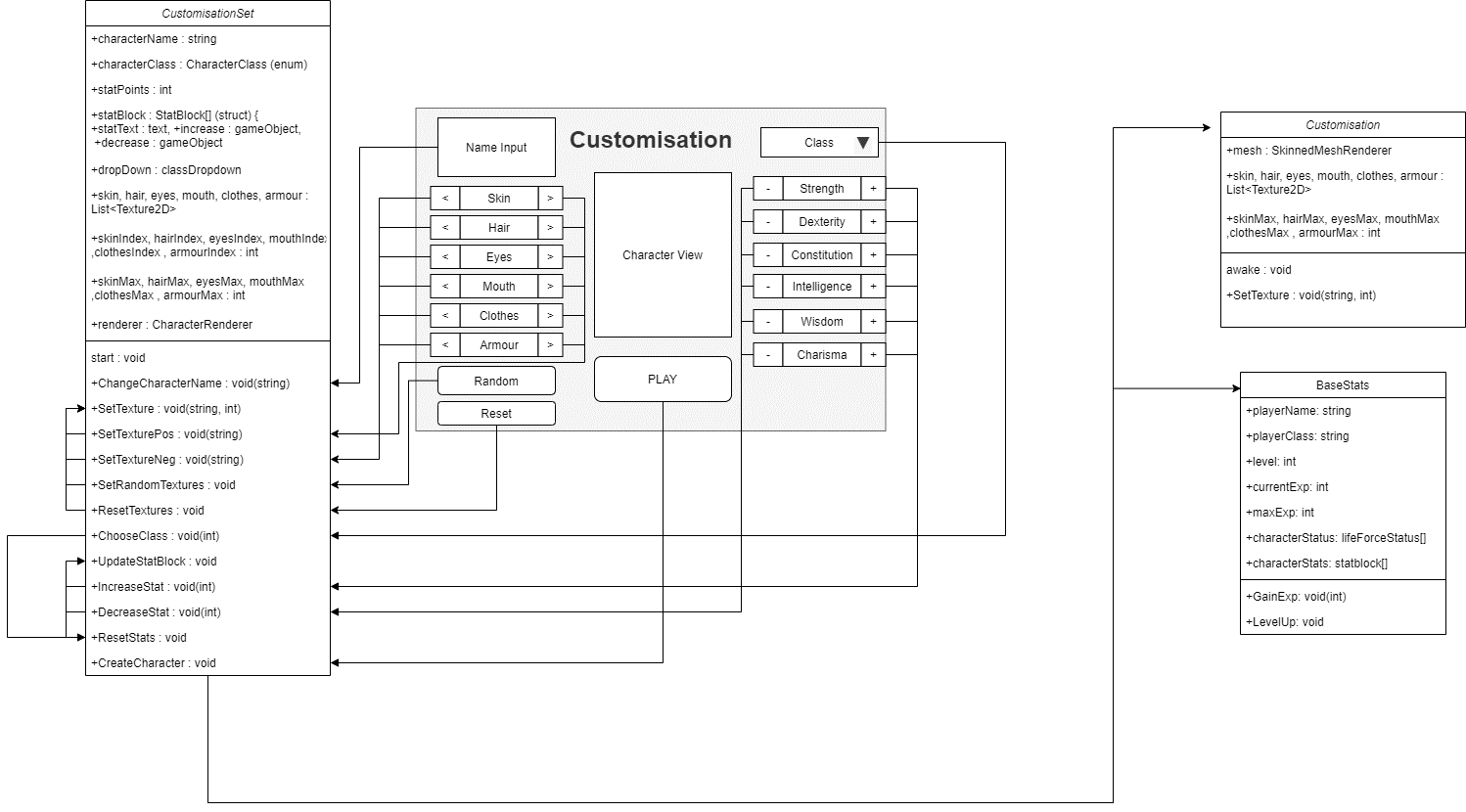
## Non-Goals

* When customising character appearance, player can use Random to set all the parts to random textures out of the options and Reset to set them all back to default
* The name entered by the player during customisation is stored and can be used in dialogue with NPCs so when the player is speaking it comes up with their name.

## Proposed Design

* Character customisation scene is loaded upon a new game
  + After customisation is finished, game is loaded and all customisation can be seen on the player and the HUD
  + In-game character has a name and class that comes from the customisation, and a level and experience
  + Customisation is saved so it doesn’t reset when exiting and continuing
* Character appearance can be customised
  + 6 different parts can be selected from a list – skin, hair, eyes, mouth, clothes and armour
  + Display of character so player can see them change as they change things
  + Character name can also be changed
* Character class can be chosen out of 8
* Character has 6 stats that can be seen on customisation screen
  + A limited number of points can be used to increase the stats above their defaults
  + Increased stats can then be lowered again to get points back
  + Different classes have different default stats and changing class resets stats and gives all points back
  + Different stats affect the game in different ways e.g. constitution affecting maximum Health and wisdom affecting maximum Mana
* Character screen can be toggled in-game
  + Character screen displays character name, class, level, experience, status (health, mana, stamina), and the 6 stats.
  + Character screen is updated with new values whenever it is opened

# 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

* String for character name
  + Strings are arrays of characters that make up text.
  + After the player name is entered in the text field, the changeCharacterName function which takes a string dynamically from the text field will set the player name to that string
* Dropdown for class dropdown and Class enum for character class
  + Enums are custom lists of identifiers. In this case, the values the enum can be will be all the different classes. An enum is used over an int because each value of the enum can be the name of the class rather than having to know which int relates to which class
  + The dropdown type is a reference to a dropdown UI element in the scene. By connecting to the dropdown in the scene, its options can be set to all of the classes when the customisation screen is opened.
  + The classes in the dropdown will be in the same order as the enum so choosing one in the dropdown will convert the index of the one selected into the matching enum and set the class to it
* Text for the stats text and gameObjects for increase and decrease buttons
  + a reference to the text allows it to be changed in code with the value, so for example when the value of strength is changed, the text for strength can change showing the new value. The text property of Text is a string, which can be a string showing the name of the stat added to a colon and the value to display in the scene e.g. “Strength: 5”
  + Referencing the gameObjects for the increase and decrease buttons allows them to be enabled and disabled when the player is able to use them.
* Renderer for character display on screen
  + By referencing the character renderer, the display of the character as things are edited can be changed to reflect the options the player has chosen
* Lists of Texture2Ds for the different parts of the character
  + Texture2Ds are classes that represent textures. Different textures for the character’s skin, hair, eyes, mouth, clothes and armour will be stored and set to the character based on the player’s choices.
  + Lists are dynamic arrays that can be added to and removed from, so a list for each of the 6 parts of the character will store all the possible textures that can be chosen from.
  + An int for the current selected texture and another int for the max value that can be selected will allow the player to increase and decrease the index to select a texture, with it looping back to the first one when reaching the max

## Data Model

* Customisation screen will be opened upon selected new game
* When player is finished with customisation, everything will be saved to a binary file and loaded into the game, with the selected textures being loaded into the Customisation script and put onto the character renderer and the selected name, class and stats being loaded into baseStats.
* Upon exiting the game, everything is saved to the binary file and the player will not have to customise again when they are loading a game next time, but upon selecting new game it will all be deleted so the player can customise again
* If the player selects load game but no data for customisation can be found, it will instead select new game and go to the customisation screen

## Interface/API Definitions

Describe how the various components talk to each other. URL and the format of the data and parameters used.

Libraries used:

UnityEngine.SceneManagement – Allows SceneManager to be used, which can call functions such as LoadScene, which changes the scene to another one based on its parameter.

UnityEngine.UI – Allows references to the Unity UI such as the text, buttons and dropdowns/sliders/toggles. All the functions of them can then be used, and the values can be changed.

System.IO – stands for Input/output and allows for files to be opened, read from, and written to, which is needed for saving data to binary files

System.Runtime.Serialization.Formatters.Binary – the library for using the binary formatter to serialize and deserialize the data to a binary file.

## Impact

* Performance – doing things to reduce the performance cost of running the game
  + Running events that only need to be run once a single time rather than every frame, for example updating text and the renderer to show the player what they are changing only at the moment of changing it rather than in update every frame.
  + Coroutines to wait for responses over multiple frames
  + Changing the scene with LoadSceneAsync so the loading can be done in the background
* Security
  + Public variables vs private variables – Public can be accessed outside of the class and changed from the editor and private can only be accessed from within the class. Also, protected variables are the same as private but can be accessed by classes inheriting from them.
  + Saving – When saving character data binary should be used over PlayerPrefs and other methods because it can save to any location with any name, making it harder to find, and it is also more difficult to edit.

## Risks

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

* I’m not sure how to get the lighting to work on the character model so it is not completely dark when facing certain ways
* I don’t really understand how the mesh and changing the textures on it works and need to research it more

## 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.

* Removing lighting shadows from the game entirely is a solution to the lighting problem, however this will make the environment look less interesting and finding a way for the character model to ignore shadows would be a better solution

# System Testing

## Testing

Show progress, Error reports and explain fixes you used.

**Saving customisation** – At first I had the data from the customisation screen saving to a file as the player changed values such as class, textures and stats but this caused a problem where the game would not load properly if the player had exited in the customisation screen and then tried to load from the main menu. To fix this, I made it only save all of the customisation data when it was finished and the player pressed play so all of the data was there. Then, if the player pressed load and not all the data was saved, the game instead loaded a new game and went to the customisation screen.

**Player name** – The name the player entered was not loading into the game and was ending up as the default ‘Adventurer’ even though everything else was being loaded correctly. After trying a few things it turned out this was because I was setting the variable called name which was getting confused with the name of the gameObject and not actually changing the value I wanted to change. After changing the variable name to characterName instead the problem was fixed.