Technical Design Document

|  |  |
| --- | --- |
| **Version** | **Description** |
| 1.0 | Initial document setup. |
| 1.1 | Made extensive changes to the majority of the document.  Decided to remove the “High Score” feature as it was outside the scope of original project requirements and time to release is limited. |
| 1.2 | Added section for play testing. |

# Game Overview

## Game Summary

Simon is a game focused on pushing a player’s memory to his/her limits. The player must remember a series of lights and repeat the sequence exactly. If the player succeeds, the series of lights becomes slightly longer and more complex but if the player selects the wrong sequence the game ends.

# Development Environment

## Development Team

* Richard Delamore: game design, programming and graphics.

## Programming Language and Graphical Framework

The game is being developed using C++ and 2D Bootstrap library provided by AIE (Academy of Interactive Entertainment).

## IDE

Microsoft Visual Studio is being used to develop the game.

## Source Control Procedures

Source control is provided by GitHub.

## Third Party Libraries

The game uses the following third party libraries:

* 2D Bootstrap
* Cmath.h
* Time.h

## Project Management Tools

No Project Management Tools are being used to develop this game.

## Other Software

Photoshop was being used to create the graphical elements for this game.

## Platform

PC only game.

# Development Overview

# Timeline

|  |  |
| --- | --- |
| Alpha | 9/5/2019 |
| Beta | 13/5/2019 |
| Gold | 14/5/2019 |
| Playtesting | 13/5/2019 |

# Game Overview

## Gameplay

The player must remember a series of lights and repeat the sequence exactly. If the player succeeds, the series of lights becomes slightly longer and more complex. The game ends when the player fails to select the correct sequence.

## Game Objects and Logic

**SimonApp:**This object instantiates the **Controller** and **Simon**.

**Controller:**   
This object dictates what happens on the start screen. It also instantiates **Simon** when the player selects “Start Game”.

**Simon:**   
This object controls the game logic for Simon. It accepts user input and controls the finite state machine for Simon. It also instantiates the **Colour Pattern** which is a Linked List of colours created by the finite state machine.

**Colour Pattern**:  
This object contains a Linked List of colour patterns.

## Game Flow and Mechanics

**Controller:**The Player is presented with two options: Start Game and Exit Game. If the player selects Start Game then the object **Simon** will be instantiated.

The controller also checks whether **Simon** has entered the Restart State.

**Simon:**A Finite State Machine is launched based on an enumerator sequence.

* **New Pattern State:** The FSM generates a new pattern and stores it in the **Colour Pattern**. After the pattern is generated the FSM changes to the Flashing State.
* **Flashing State:** The FSM flashes the new pattern to the screen. After the new pattern has been flashed the FSM changes to the User Input State.
* **User Input State:** The FSM accepts input from the player and checks this input against the last pattern generated. If the user input is correct, Simon changes to the Pause State. If the user input is incorrect, Simon changes to the Lose State.
* **Pause State:** The FSM sits in the pause state for a short period of time. This state exists purely to ensure that the player experiences a short break between entering the correct pattern and seeing a new pattern. Once the Pause State ends the FSM changes to the New Pattern State.
* **Lose State:** The FSM sits in the lose state for a short period of time. This state exists purely to ensure that the player experiences a short break between realizing they lost and exiting the current game session. Once the Lose State ends the FSM changes to the Restart State.
* **Restart State:** The FSM sits in the restart state. It remains in this state until the Controller deletes the Finite State Machine.

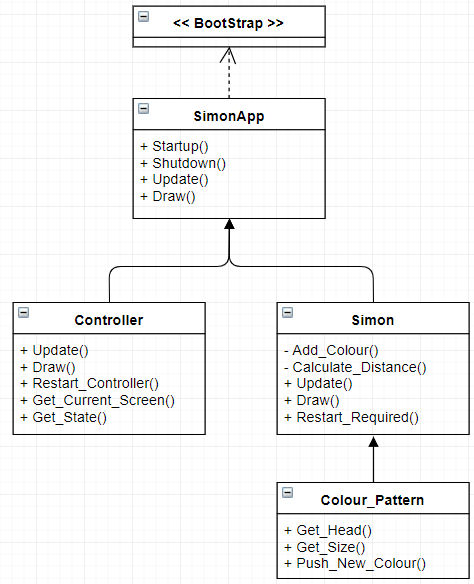
**Lose State:**The player can lose the game only by entering the incorrect colour pattern.

**Win State:**The player can’t win the game. The player can only win each subsequent round until they can’t remember any more colours.

# Game Architecture

Core Data Structures

Classes



Algorithms

There are no specific algorithms used by the game.

# UI and HUD

On this screen the player can select “Start Game” and “Exit Game”.



On this screen the player plays the game, Simon. The note book acts as an interface to show the current status of the game. The red square (on the right) shows a countdown of how long before the next pattern.



Play Testing

This section is for recording the results of play testing.

|  |  |
| --- | --- |
| Name: |  |
| Date: |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Steps to perform | Expected Outcome | Pass or fail | Comments |
| Launch the game. | Game will launch. |  |  |
| Test all options on the main screen. | All buttons will function as expected. |  |  |
| Attempted to win the game. | Game can’t be won, but you should be able to successfully win a few patterns. |  |  |
| Purposely lose the game. | The game should inform you that you’ve lost and exit to the main screen. |  |  |
| Attempt to break the game. | There should be no bugs. However, please attempt any combination to try break the game. |  |  |