Landslider: A Depth Detection Game

Design Specification Document

Team 5
Michael Fadem
Cameron Chinena
Christopher Gonzales
Eduardo Lash
Jeremy Keys

Table of Contents

1 Introduction

This design document outlines the design for a gesture-based video game, called Landslider. The game will detect the user's hands and fingers by utilizing the intel RealSense F200 3D camera. The primary objective of this game is to keep a non-playable character (NPC) alive until they reach the end of the level. The player will accomplish this goal by using the hand and gesture recognition provided by the RealSense camera to grasp the virtual land tiles and moving the more dangerous tiles such as volcanoes further away from the NPC while bringing safer tiles such as flat land closer to the NPC. This document outlines the design decisions that will create this game.

1.1 Design Drivers

The most important factor in driving the design of the system is the requirement that the application interface directly with the Intel RealSense camera, rather than more traditional input methods. This constraint has driven the choice of design in several areas, most notably in the choice of game engine Unity 5.0.0 for the development of the game. Intel has currently released SDKs for both the Unity and Unreal game engines. The choice to use the Unity Game Engine is due in part to the majority of the team have previous experience with the engine, thereby speeding up development efforts. Additionally, Unity allows C#, and JavaScript as programing languages that can be utilized, whereas the Unreal Engine only supports C++. C# and JavaScript are both scripting languages making it easier to script additions that the game might need if any issues were to arise, also there are less limitations with being able to use two scripting languages. The primary motivation in using the RealSense cameras instead of a device such as the Microsoft Kinect, is that the RealSense has a lower setup requirement, as well as a higher fidelity in detecting user's hands, fingers, and gestures.

1.2 Acronyms, Abbreviations, Definitions

- Intel RealSense F200 3D Camera - Hardware developed by Intel that uses various sensing technologies, such as RGB and infrared sensors, to achieve depth detection, hand tracking, and 3D scanning. The player will interact with this camera during gameplay

- Unity Game Engine - Free-to-use video game creation engine and IDE, integratable with Intel RealSense cameras

- NPC - Non-Playable Character, this will be the character that is running indefinitely over the tiles that are being moved by the human player..

- SDK - Software Development Kit

- Microsoft Visual Studio - Microsoft developed integrated developer environment used for coding in a variety of programming languages including C# which is the language choice of the system.

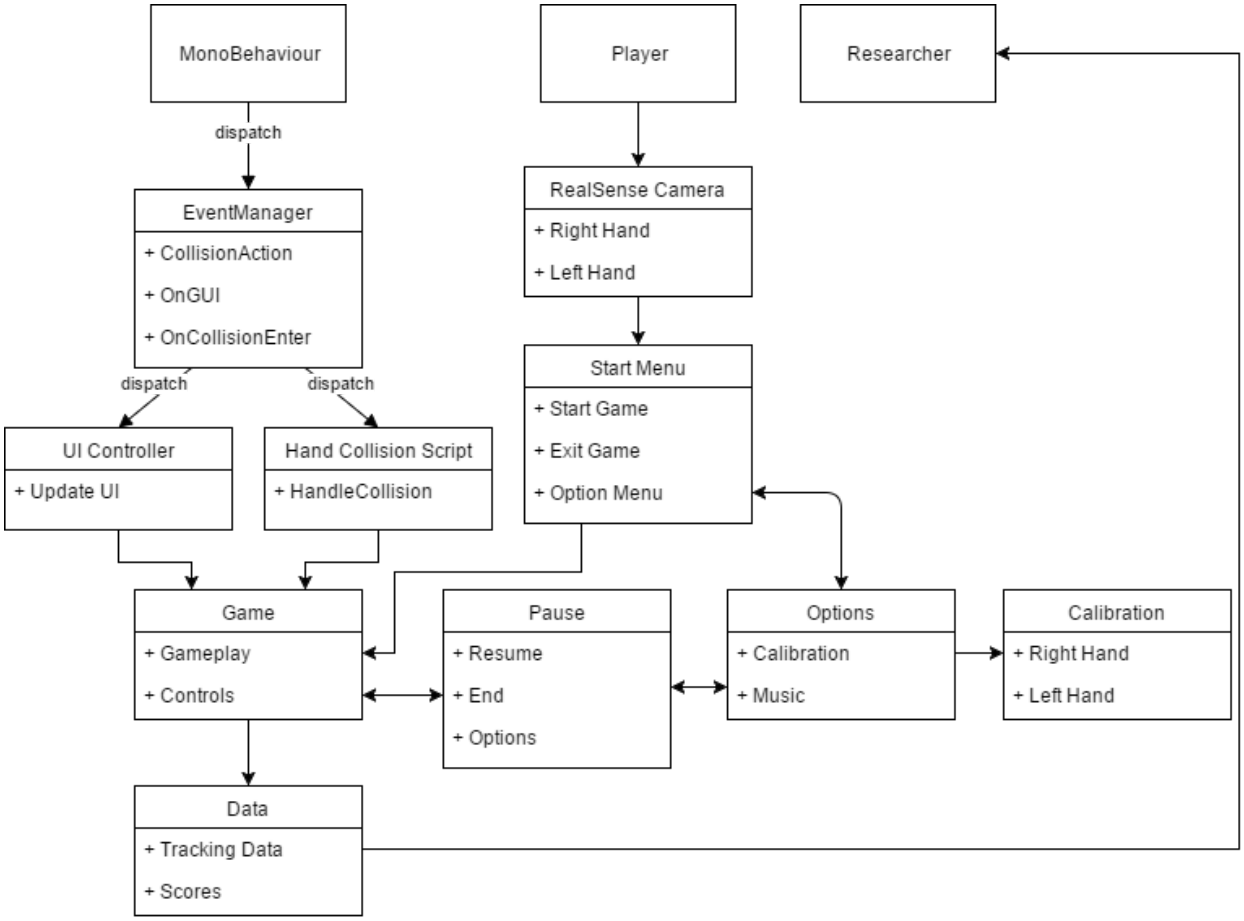
- ❏ Microsoft Kinect - Device developed by Microsoft, that uses motion based input provided by the human user, to be used with video games on Xbox and Windows PCs

- ❏ MonoBehaviour - The base class from which every Unity script derives

2 Platform, Languages, and Environment

The chosen platform that the game will be built for is a PC running Windows 8.1 and above with a minimum 4th generation Intel CPU per RealSense camera requirements. The development environment for the game is Unity 5.0.0 to allow for easier integration with the RealSense SDK and RealSense Unity Toolkit. The primary languages used within the Unity Game Engine is C# and Javascript.

3 System Architecture



The primary system architecture for this game is an event-driven architecture. Unity relies heavily on events, which allow objects in the game to either send events or respond to receiving events from other objects. All Unity scripts inherit from a class called MonoBehaviour, which has a variety of methods that can be implemented and then which will be called upon internal Unity events being triggered.

4 System Design

This section will discuss the low-level technical design of the game's system components.

4.1 Data Design

This video game can be naturally represented by using object oriented methods, including encapsulation, inheritance, and composition. The game will be comprised of classes and objects, specifically, object-scripts as the primary data structures.

4.1.1 NPC object

The game involves a NPC object running across movable tiles, which will be manipulated by the player using gesture input picked up by the RealSense cameras. The Player will have the following attributes:

1. **Speed** (integer) - the current scalar speed of the player object
2. **Health** (integer) - the current health of the NPC

4.1.2 Tile interface

One of the main game elements in this game is the "tile," individual blocks of land across which the NPC runs. These tile will be apart of a tile row, a strip of 5 connected tiles that the human user will move.

1. **TileModifier** (ObstacleModifier) - an object which contains code that modifies the current tile in some way, either by making it dangerous to the player (decrement player health), or slow the NPC (decrement speed), *et cetera*.

4.1.3 GameManager object

This is the object which coordinates most of the game's logic. This object will contain the following attributes:

1. **EndTile** (Tile) - the tile which represents the goal; the tile which the player wants to help the NPC arrive at
2. **Score** (integer) - the player's current score
3. **Time** (float) - the (current) amount of time in the current session/"life"

4.1.4 Player object

The player object, which primarily handles the gesture input coming from the RealSense cameras. This object will contain the following attributes:

1. **RightHandGesture** (string) - a string which corresponds to the right-hand gesture detected by the RealSense camera
2. **LeftHandGesture** (string) - a string which corresponds to the left-hand gesture detected by the RealSense camera
3. **RightHandPos** (Vector3) - a Vector3 containing the three-dimensional position of the player's right hand
4. **LeftHandPos** (Vector3) - a Vector3 containing the three-dimensional position of the player's left hand

4.1.5 ObstacleModifier interface

This interface, when overloaded, will allow us to easily give individual tiles modifying attributes. An implementing class should supply the following information:

1. **damageDealt** (integer) - the amount of damage that collision with this tile does; if none, should be equal to 0
2. **healthRestored** (integer) - the amount of health restored to the NPC that collision with this tile does; if none, should be equal to 0
3. **speedModifier** (integer) - the scalar amount of how much the NPC's velocity changes as a result of collision with this tile

4.1.6 GestureController object

This object will handle input coming from the RealSense cameras via the RealSense Unity toolkit.

4.1.7 DataLog object

This object will handle data collection and logging actions that will be exported for the researcher to analyze.

1. **realsenseVectors** (integer) - the x, y, and z coordinates of the user's virtual hands relative to the camera in the
2. **uiValues** (integer) - the values of the UI components within the game space, including health, score, and time

4.2 Architecture Component Interfaces

The event-driven architecture simplifies much of the program logic, but it is still worth enumerating the various objects and scripts which will handle the events.

4.2.1 EventManager object

The EventManager object has responsibility of detecting events triggered by the Unity game engine and handling them. It has the following attributes and methods:

1. **OnCollision** (event CollisionAction) - the reference to the CollisionAction delegate, which will be called by the OnCollisionEnter method

2. **CollisionAction(Collision)** (public delegate void function) - a delegate which is invoked upon events received from the Unity engine (a collision is detected)
3. **OnGUI()** (void function) - MonoBehaviour function which is called when a GUI action (such as a button click) is carried out
4. **OnCollisionEnter(Collision)** (void function) - MonoBehaviour function which is called when one rigid body collides with another rigid body or collider.

4.2.2 UI Controller object

This object will be tasked with updating the various UI elements, including the heads-up display elements such as the current health, score, and level.

1. **UpdateUI()** (void function) - a method which will be called upon a collision being detected, thus updating the score and health

4.2.3 Handle Collision Script

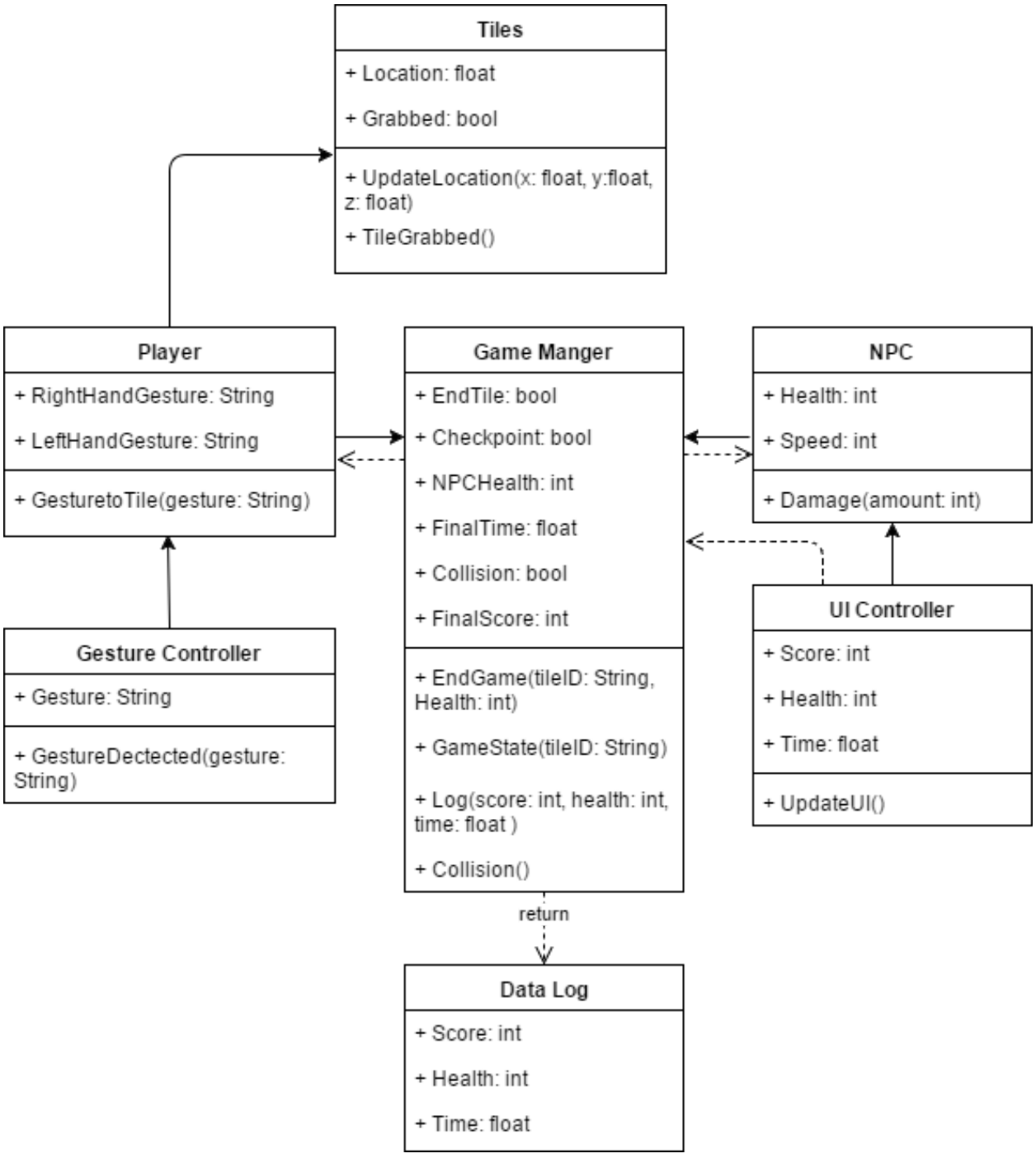
This object will be tasked with updating the various UI elements, including the heads-up display elements such as the current health, score, and level.

1. **HandleCollision(Collision)** (void function) - a method which will be called upon a collision being detected, thus affecting the NPC in various ways (lowering health, decreasing or increasing speed, resetting position, granting temporary invulnerability, *et cetera*).

4.3 Architecture Component Designs

This section will describe in detail the architecture of each component that exists in within the system.

4.3.1 Architecture



4.3.2 Game Manager

The game manager component of this system is most likely the most important component because it handles the background operations to determine if the game is still in a playable state or not.

This means that it is checking if the NPC has reached the end tile, checkpoint tile, or a depleted health bar. After the conclusion of the game the manager will output the results of the round to the data log file to be viewed by the researcher. .

4.3.3 Player

The player will not be represented in the game space except for the RealSense hand models so that the player will be able to accurately interact with the tiles. The player component will be the only factor that is influencing the location of tiles within the system which will result in the player being responsible for the action that affect the NPC's speed or health.

4.3.4 Tiles

The land tiles will be individual entities within the game with each tile having its own functionality and attributes that will affect the NPC is a positive or negative way. These attributes will affect the NPC's attributes such as health and speed and the player's overall score and time attributes.

4.3.5 NPC

The primary objective of the NPC is to reach the level end tile in the fastest time possible. In order to obtain this goal the NPC needs to maintain a health amount that is greater than zero, which is relayed to the UI indicating to the player if the NPC can still continue if they receive damage from any of the tiles meant to impair the NPC.

4.3.6 Gesture Controller

The gesture controller is tasked with maintaining which gesture the player is currently performing so that that data can be passed to the either the game manager or tile object so that the appropriate action can occur that relates the that gesture such as grabbing and releasing a time.

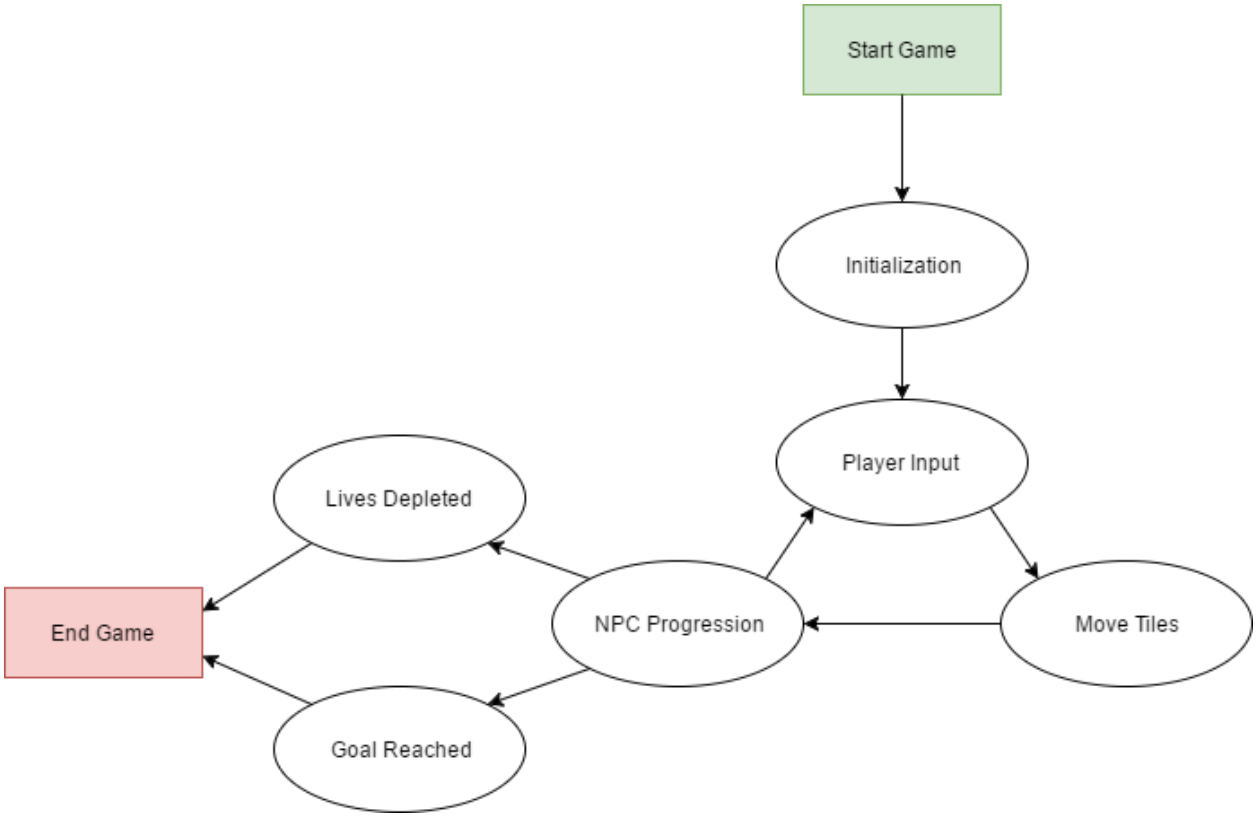
4.3.7 UI Controller

The UI controller is tasked with maintaining the current score, health, and time of the game state. It will also handle updates to its attributes if any of the data experience a change such as a decrease in player score or health.

4.3.8 Data Log

The most important component for the researcher is the data file which will contain the final values output from the UI controller so that the researcher can see how the player progressed during the current level of play.

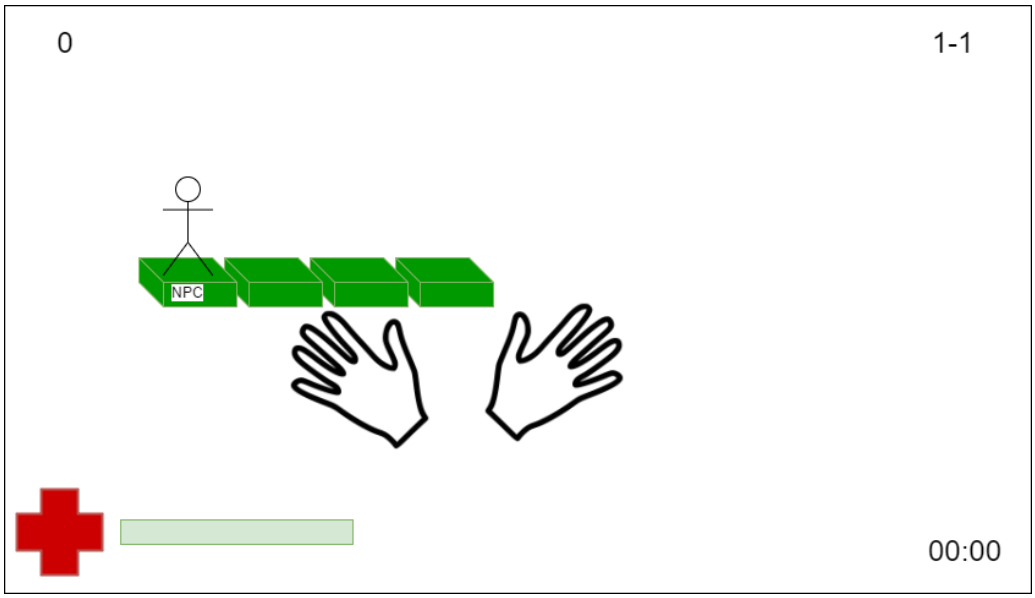
4.4 Important Object State Diagrams



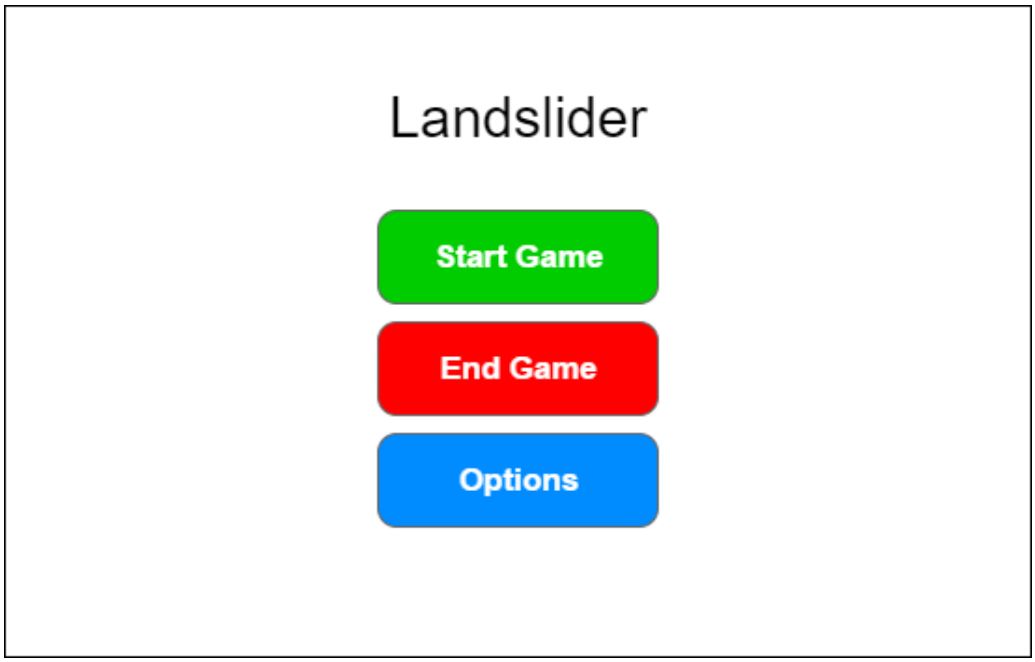
5 User Interface Design

- Player health or the amount of “hits” the player can endure will be shown towards the bottom left of the screen, any damage taken will be indicated as the game progresses.
- A score indicator is shown in the top left corner of the screen. This will show the player their current score.
- A left-hand and right-hand object will be shown on the screen indicating where the user has their hands placed, to better show which object the user is currently holding or grabbing.
- A level indicator will be displayed on the top right corner of the screen, indicating what level the user is currently on.
- The main menu will contain three separate buttons that will allow the player to start the game, exit the game, or go to the options menu.
- The pause menu will allow for the user to resume playing the game, exit to the main menu, and enter the options menu.
- The options menu will allow for the user to modify game objects such as game music, player speed, and to check the calibration of the RealSense camera.

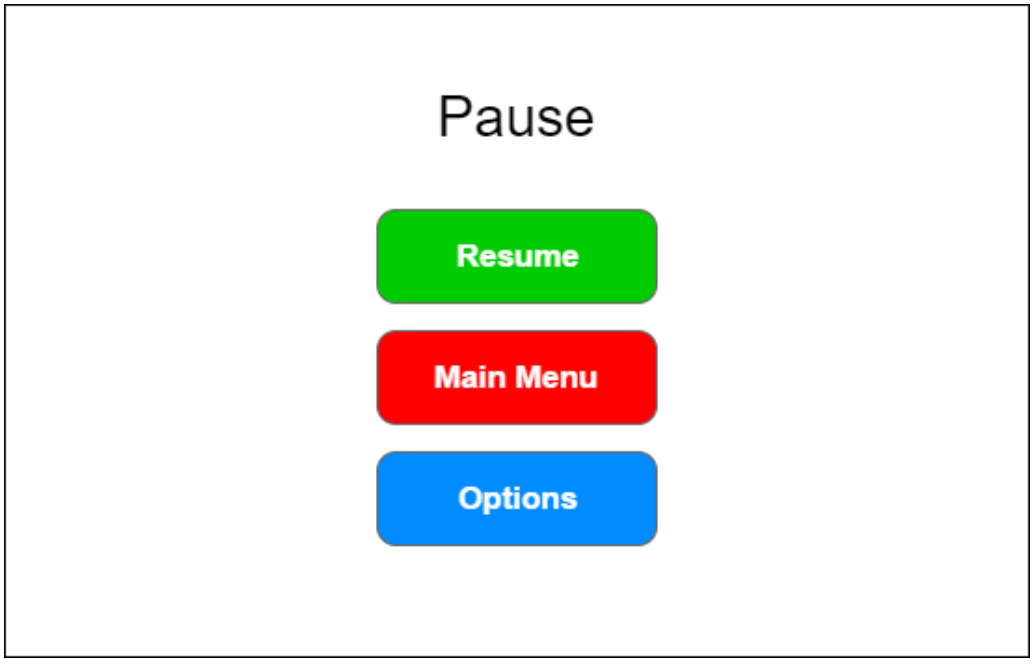
Main Game Screen



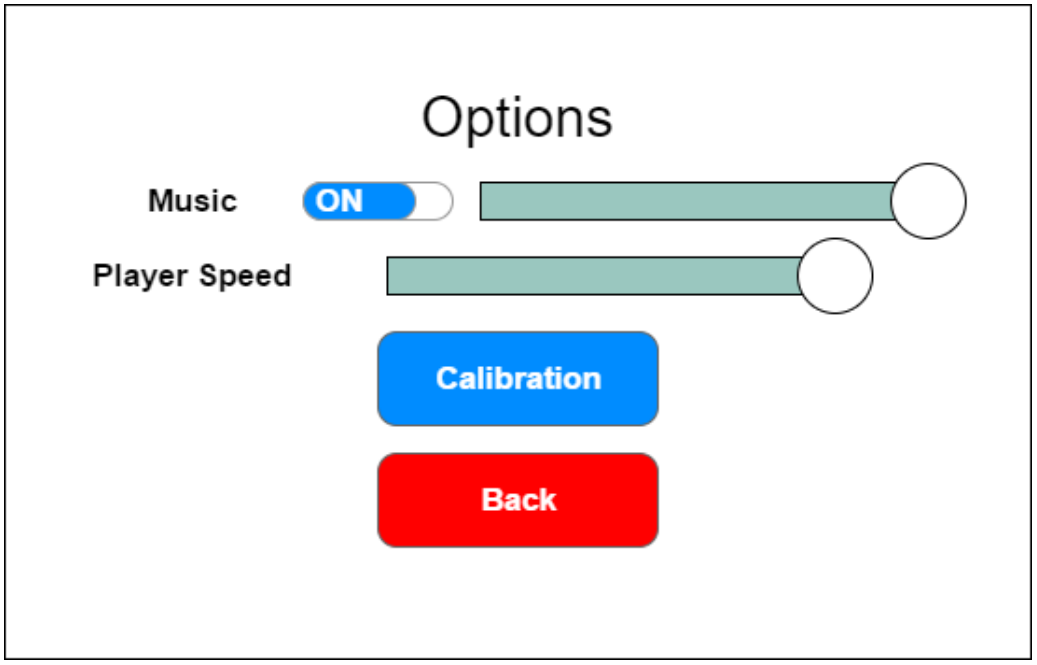
Main Menu



Pause Menu



Options Menu



6 Deployment Design

The Unity Game Engine does not need to be installed on end users systems because Unity builds create self contained executables for deployment, as long as the proper files are present it can run on an machine that the game was built for. This means that the final build of the game can be deployed through any form of file transfer, whether it be email, USB, etc.

The application will be delivered as a runnable executable file that can be executed on any Windows PC running at least Windows 8.1. In order to execute the game the executable needs to be located in the same directory as the assets folder to ensure the game will execute properly. The executable will have a small prompt before launching the game that will let the user customize the resolution and screen options.

7 Coding Standards

Basic coding conventions will be used throughout the project, following Microsoft's basic C# coding standards.

- Layout Conventions
    - Write only one statement per line
    - Add at least one blank line between method definitions and property definitions
- Commenting Conventions
    - Place comments on a separate line, not at the end of a line of code.
    - End comment text with periods.
    - Do not create formatted blocks of asterisks around comments.
- Language Guidelines
    - Strings
    - Arrays
    - && and ||
    - Event Handling
- Naming Conventions
    - Meaningful names
    - Descriptive
    - Camel Case

For more information: https://msdn.microsoft.com/en-us/library/ff926074.aspx