**Software Requirements Specification**

March 1, 2017

Team MangoWEB

Caleb Warren

Collin Epstein

Julian Bertini

**1. Introduction:**

This document covers all Software Requirements for the development of a 3D puzzle-type game for iOS and Android handheld mobile devices.

**1.1. Purpose**

The purpose of creating a game can be somewhat challenging, considering the main motivation of gaming is entertainment. However, from a programming perspective, the purpose of such a task can be more narrowly defined. We decided to create a game because doing so involves aspects of software design that no one in our group has explored in depth before. It also requires many different processes (graphics, controls, Game State, interaction, etc.) to be executed simultaneously with as little error as possible. A game was most desirable for us to create because we enjoy playing video games ourselves, we wanted to gain the experience of creating a game, and to extend this exciting project to other players.

**1.2. Scope**

This document covers the requirements of creating our game in its entirety. All code will begin from these requirements. The graphics elements will be implemented in the Unity Video Game Engine using assets that other developers have created in the past. Requirements for the graphics of this game will be created with this in mind.

**1.3. Definitions, Acronyms, and Abbreviations**

Unity: Unity is a multiplatform video game engine that can build programs to multiple hardware platforms. It also has a large database of user-submitted libraries of graphics packages, models, animations, scripts, materials, shaders, sprites, textures, maps and other assets for free or paid use.

FPS: frames per second. The number of graphics frames rendered and displayed per second while game is running. Higher FPS results in smoother experience, but is also more computationally intensive.

**1.4. References**

* Unity Documentation: <https://docs.unity3d.com/550/Documentation/Manual/>
* ISO C# Standards: <http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip>
* Apple iPhone Technical Specifications: <http://www.apple.com/iphone-7/specs/>
* iPhone 6 Plus V. Other SmartPhone Battery Specs: <http://www.trustedreviews.com/iphone-6-plus-review-battery-life-and-verdict-page-6>

**1.5. Overview**

This game will consist of three high level categories: Display, User interface, and Game State.

The Display of this game will be managed and connected to our Game State by the Unity engine. Using Unity’s Asset Store, we will find and implement appropriate graphics assets that fit our game’s internal environment. Our idea for the aesthetic of our game is a 3D world that has a 2D-like gameplay: a world in which objects appear to have three dimensions, but a scrolling screen that follows a player whose maneuvers are limited to two dimensions. This is achieved with and isometric camera view angle (such as the games *Bastion* and *Transistor* by Supergiant Games).

User Interface will define how the user may interact with the Game State. The user will observe this interaction with the Display associated with the current Game State, but the User Interface will allow the user to directly manipulate Game State. This will deem how the user moves, how to move objects that the user is interacting with (when available), camera manipulation, and the act of interacting with the game world and puzzles.

The Game State is the underlying code that everything else relies on. It consists of the game world and the current state of the player, the environment, and the puzzles. This should be the main focus of the game during the first iterations of our project. Constructing this environment will span a wide variety of things. For Display, the Game State will control where objects are located and object appearance. It will have key locations at which the user is allowed to interact with and solve the puzzles found throughout the game, and is in charge of changing what the user sees. For User Interface, the Game State will change according to user input through the User Interface. So the Game State will manage what operations a user is allowed, and deems when those actions are valid to the user.

**2. Overall Description**

**2.1. Product Perspective**

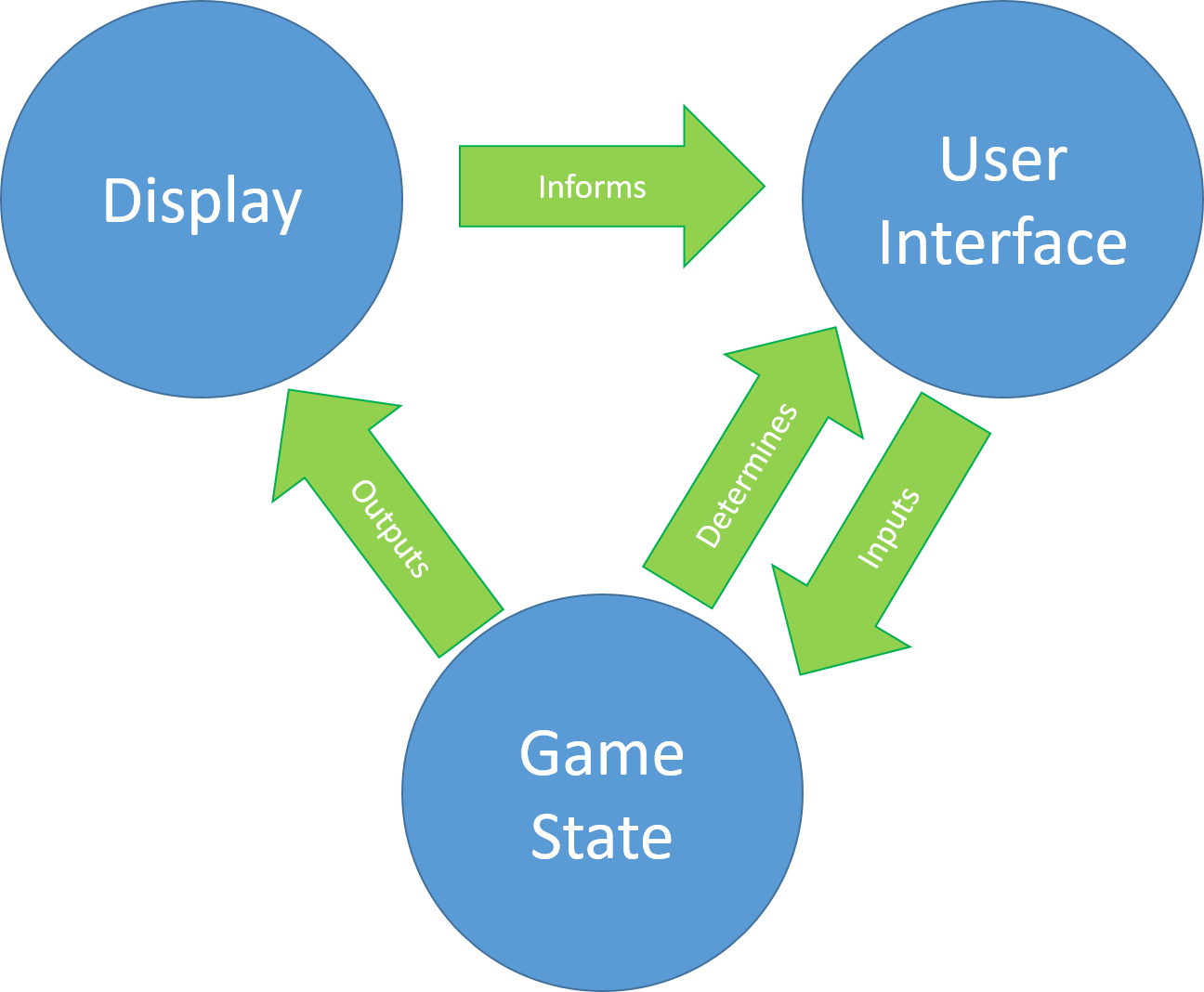
The game is designed for mobile handheld devices, specifically smartphones running the iOS or Android operating systems. We will program the game in the Unity game engine. Unity has multi-platform build capabilities, allowing us to design and program the game largely independent of operating system and hardware considerations.

**2.2. Product Functions**

Functions for the user are movement, looking, and interacting with the environment. Game State functions include coupling user controls to the movement of the screen (effecting what the user sees), when to load objects and when to hide them from the user’s environment (when scrolling, objects seen in the past should be hidden and new objects should become visible as the user moves), which objects to load and where on the screen they should appear (this brings the environment and the graphics together).

The major functions of the system are User Interface, Game State, and Display. These three functions are fundamentally interconnected.

Function Block Diagram



The game has an initial Game State. The Game State includes all the information concerning the current circumstances of the game and determines the current User Interface. The Game State then outputs information to the Display, which shows the current camera view, associated data about the game, and the current User Interface. The player observes the Display and then interacts with the User Interface based on the context the player observes. The User Interface determines the commands that the player can use. The User Interface then changes the Game State via inputs to the game from the user, and the cycle continues until the program terminates.

**2.3. User Characteristics**

Our target audience for this game is a subset of the audience for mobile games. The game is intended for individuals who play mobile games and enjoy puzzles based on spatial reasoning. These individuals likely play mobile games for short durations at a time, for instance on public transit or during a bathroom break. They will enjoy a challenging puzzle, and find pleasure upon completing small tasks. This audience will need a basic understanding of how to interact with menus and handheld devices. We expect an age range of 8-75+ will have the ability to understand and enjoy this game.

**2.4. General Constraints**

The game is subject to control and performance constraints inherent to applications on smartphones. All input to the program must come in the form of taps and swipes on the screen by the user’s finger(s). These rudimentary actions must be interpreted contextually to allow the user control over the game’s many functions.

Performance is limited to what a typical smartphone can accomplish. We use the technical specifications of the iPhone 7 as a reasonable benchmark[[1]](#footnote-0). Fortunately, modern smartphones are sufficiently powerful that we do not anticipate hardware to put meaningful limits on the performance of our game.

The only major limitations are screen size and battery life. We must take special care to ensure that we design a game world that can be reasonably observed, controlled, and enjoyed on a 4-6 inch screen. Additionally, the game ought not to require so much power as to drain a smartphone battery unreasonably quickly. Most 3D games will deplete a fully charged mobile battery in approximately 4 hours[[2]](#footnote-1). We expect to have the same power needs.

**2.5. Assumptions and Dependencies**

We make the following assumptions about this project:

* Dependent upon Unity’s graphics
* Dependent upon Unity’s ability to expand code to multiple languages/devices
* Assume user interface is sufficient for gameplay
* Assume user has general familiarity with smartphone operation.

**3. Detailed Requirements**

**3.1. External Interface Requirements**

**3.1.1. User Interfaces**

The user will be interacting with our game through taps and swipes on a screen. There will be a couple of clickable buttons that will produce a menu when tapped; Main Menu and Inventory.

The Main Menu will allow the user to save their current game state, load a previous game state, change sound levels/vibrations, view how to use controls, and review the tutorial, that can be closed by the user. Clicking on this button will pause the game.

The Inventory button will contain a list of items that the user has been allowed to collect throughout the game. These items will be useful to the user as the game progresses. This list, once opened, will also contain a way to close this list once opened. It will have a sub-menu at the top of the screen to manage sorting of the objects in the list.

Both of these will be available to the user at any point in the game. Both of these menus will appear on top of the game state. Once one of them is opened, it must be closed before the other can be opened. When neither menu is open, the user is free to move around and interact with objects when necessary.

The user will move by a touch-and-drag method on the screen. The user will tap on the character, and drag them the direction they want to move. When the user encounters a puzzle or movable object, they will be prompted to interact with the object. This prompt should be discrete yet obvious to the user. To interact with an object, the user will use the same touch-and-drag method. Touch the object on the screen, and drag it to a new position on the screen. A double touch and drag will allow the user to rotate an object when necessary. For certain objects this feature may be restricted. As mentioned before, certain objects in the game will be available to the user in their inventory. These objects, when touched, will be removed from the user’s screen and added to the inventory.

**3.1.2. Hardware Interfaces**

Must successfully install and run on mobile devices, specifically smartphones.

**3.1.3. Software Interfaces**

Our game will operate largely independent of other software. It must be compatible with iOS and Android mobile operating systems. Ideally, it will be compatible with Apple’s App Store and Google’s Play Store to facilitate distribution.

**3.1.4. Communication Interfaces**

We do not currently believe a communication interface to be necessary to our game. In the future, this may be included for a multiplayer feature. However for our first version of the game, we do not plan to need any communication interfaces.

**3.2. Functional Requirements**

**3.2.1. Mode 1 - Menu**

**3.2.1.1. Functional Requirement 1.1 - Save Game**

There must exist a button that will save the player’s current progress in the game (current Game State). Player will return to that Game State upon losing or choosing to load that state. May be coupled with an autosave feature. This submenu must include a button that returns the user to the main menu.

**3.2.1.2. Functional Requirement 1.2 - Load Game**

There must exist a button that will load a previously saved Game State. May save only one Game State at a time, or multiple. Once level is loaded, the game must transition to Player Movement Mode (Mode 2).

**3.2.1.3. Functional Requirement 1.3 - Close Menu**

There must exist a button that will close the menu. The game then must transitions to Player Movement Mode (Mode 2) or Puzzle Manipulation Mode (Mode 3), depending on which mode the menu was loaded from.

**3.2.1.4. Functional Requirement 1.4 - Load Level**

There should exist a button that will display a menu detailing the levels/puzzles in the game. The player may select any unlocked level, and that level will be loaded. Levels are unlocked by reaching the level through normal gameplay. Once level is loaded, the game must transition to Player Movement Mode (Mode 2).

**3.2.1.5. Functional Requirement 1.5 - Sound Options**

There should exist a button that will display a submenu where the audio options for the game may be edited. These audio options include background Music volume and event-based Sound Effects volume. This submenu must include a button that returns the user to the main menu.

**3.2.1.6. Functional Requirement 1.6 - Video Options**

There may exist a button that will display a menu where the video options for the game may be edited. This submenu must include a button that returns the user to the main menu.

**3.2.2. Mode 2 - Player in World**

**3.2.2.1. Functional Requirement 2.1 - Player Movement**

The player model must move about the environment in two dimensions according to player input.

**3.2.2.2. Functional Requirement 2.2 - Termination Condition**

The player must ‘lose’ the game if the player fulfils the termination condition - falling off the edge of the world. Upon losing, the player returns to the most recently saved Game State.

**3.2.2.3. Functional Requirement 2.3 - Transition to Puzzle Mode**

The game must transition to Puzzle Manipulation Mode (Mode 3) when a puzzle is encountered/selected in the game world.

**3.2.2.4. Functional Requirement 2.4 - Load Menu**

There must be a visible button that will transition the game to Menu Mode (Mode 1).

**3.2.2.5. Functional Requirement 2.5 - Item Acquisition**

The player must acquire items found in the environment if the player fulfils the acquisition condition - player model collides with item or user selects item. Item is added to player inventory.

**3.2.2.6. Functional Requirement 2.6 - View Inventory**

There must exist a button that brings up a view of the player’s current inventory, a list of all the items the player currently possesses. Selecting an item must allow the player to use it, if appropriate. In the inventory view, there must exist a button that returns the player to the current location in the game world.

**3.2.3. Mode 3 - Puzzle Manipulation**

**3.2.3.1. Functional Requirement 3.1 - Block Manipulation**

Player must be able to select, manipulate, and place blocks in the puzzle interface.

**3.2.3.2. Functional Requirement 3.2 - Victory Condition**

Game must recognize when the puzzle is completed, inform player, and return to Player Movement Mode (Mode 2). Completing a puzzle may also unlock it in the ‘Load Level’ submenu, if implemented.

**3.2.3.3. Functional Requirement 3.3 - Load Menu**

There must exist a visible button that will load the main menu and transition the game to Menu Mode (Mode 1).

**3.2.3.4. Functional Requirement 3.4 - View Inventory**

There must exist a button that brings up a view of the player’s current inventory, a list of all the items the player currently possesses. Selecting an item must allow the player to use it, if appropriate. In the inventory view, there must exist a button that returns the player to the current puzzle.

**3.2.3.5. Functional Requirement 3.5 - Help and Hints**

There may exist a help button that will give the user a hint on how to solve the current puzzle.

**3.3. Performance Requirements**

Game must to run smoothly. Otherwise, no one will be interested in playing it. Our goal is for the game to run at a minimum of 30 FPS. Program must also respond in a timely fashion to user input. Response time must be 150ms, with a standard deviation of 10ms.

Save and load times are more intensive processes and have a higher time tolerance. They should not exceed 2 seconds with a standard deviation of 0.5 seconds. The initial load time of the game when opened should not exceed 3 second with a standard deviation of 05. seconds.

**3.4. Design Constraints**

We are scripting in the C# programming language. For best performance and ease of collaboration, we ought to adhere to the most recent ISO C# Standard: ISO/IEC 23270:2006 (E)[[3]](#footnote-2).

Our team does not intend to build most of our graphical assets from scratch. Thus, the appearance of our game will be constrained by what is available for free use on the Unity Asset Store.

**3.5. Attributes**

In order to create a points-based game, we may need to implement some sort of method that allow us to keep track of multiple players and the statuses of their games. We won’t need to create persistent data, only keep track of their data temporarily.

**3.6. Other Requirements**

Once a successful single player version is developed, multiplayer interfaces should be implemented. With multiplayer versions, global scoreboards and time sheets would be necessary to create a competition-based environment.

1. http://www.apple.com/iphone-7/specs/ [↑](#footnote-ref-0)
2. http://www.trustedreviews.com/iphone-6-plus-review-battery-life-and-verdict-page-6 [↑](#footnote-ref-1)
3. http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926\_ISO\_IEC\_23270\_2006(E).zip [↑](#footnote-ref-2)