**Software Design Document**

March 31, 2017

MangoWEB

Julian Bertini

Collin Epstein

Caleb Warren

**Introduction:**

This document covers all Software Design aspects 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 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 design of our game in its entirety.

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.

1.4. References

* Unity Documentation:<https://docs.unity3d.com/550/Documentation/Manual/>
* ISO C# Standard: <http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip>

1.5. Overview

This project is the development of a single-player puzzle-based video game application for mobile devices, specifically Android and iOs. The game will have two main states, the World view and the Puzzle view, and a series of menus.

In the World view, the player will navigate an avatar through a 3D world. However, gameplay will be limited to two dimensions on the plane of the ground, and will be rendered in an isometric view. The player will acquire and use Items and encounter puzzles. Items are consumable objects that provide situational bonuses and assistance to the player when used. Upon encountering a puzzle, the game will shift to the Puzzle view.

In the Puzzle view, the player will use a series of provided Blocks of different shapes to fill a chasm. The Blocks are puzzle-specific objects with varying properties which the player uses to solve the puzzle(s). Filling the chasm completes the spatial puzzle and allows the player to cross the chasm into new sections of the World. The user may use certain Items to help solve the puzzle.

Menus will be accessible at all times in both views of the game. These menus will include settings, save/load game, and inventory. Closing the menus will return the game to the view and state that it was in before the menu was opened.

2. Software Design Description

2.1. **Game Controller**

2.1.1. Design Overview

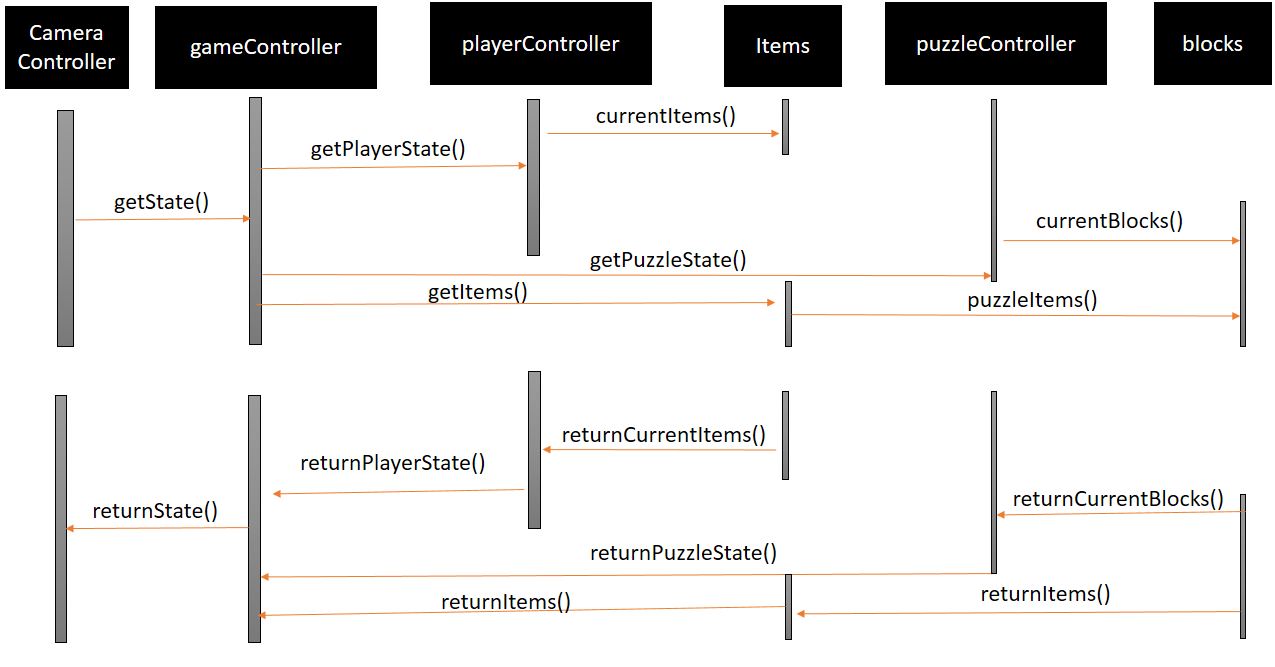
Using object-oriented design, the function of the game controller’s function is to direct which operations should execute, and when. It will manage all other subsystems.

2.1.2. Language and Infrastructure

This script will be written in the C# programming language. It will be handled within the Unity game engine, like all other assets for this game.

2.1.3. Class diagram or data structure diagram

2.1.4. Sequence diagram for retrieval of state and population of menu ‘items’



2.1.5. Detailed Design

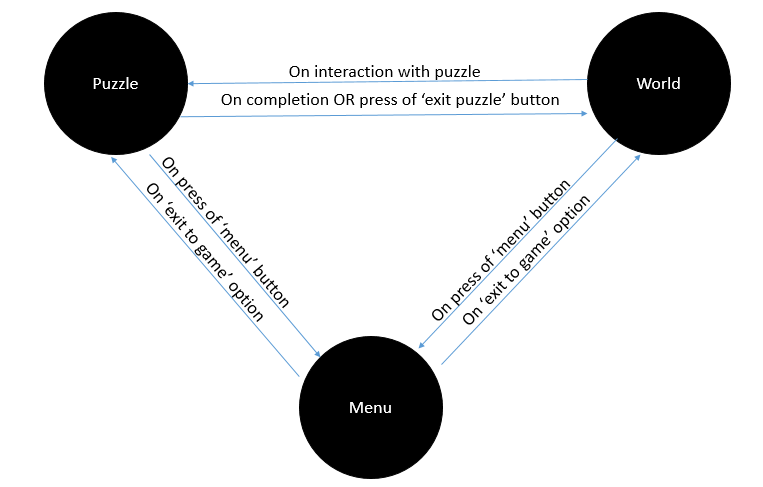
2.1.5.1. Logic/Algorithm Design

Retrieve states from puzzle and player. If the player is close to the puzzle, return the puzzle state. Otherwise, return the player’s current state. Parses user input to decide which component to apply each input to.

Internal operations:

Compare current item list with returned item list. If they are different, update the menu ‘inventory’ appropriately.

2.1.5.2. State Diagram



2.2. **Player Controller**

2.2.1. Design Overview

This component uses object-oriented design. Its function is to manage the behavior of the player avatar in the game world. This includes interpreting user input as it relates to manipulating the avatar and controlling the interactions between the avatar and other objects in the game world.

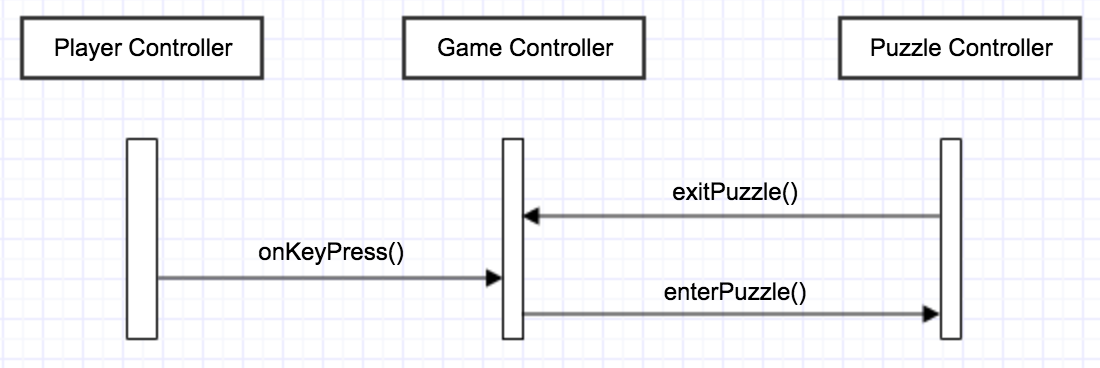
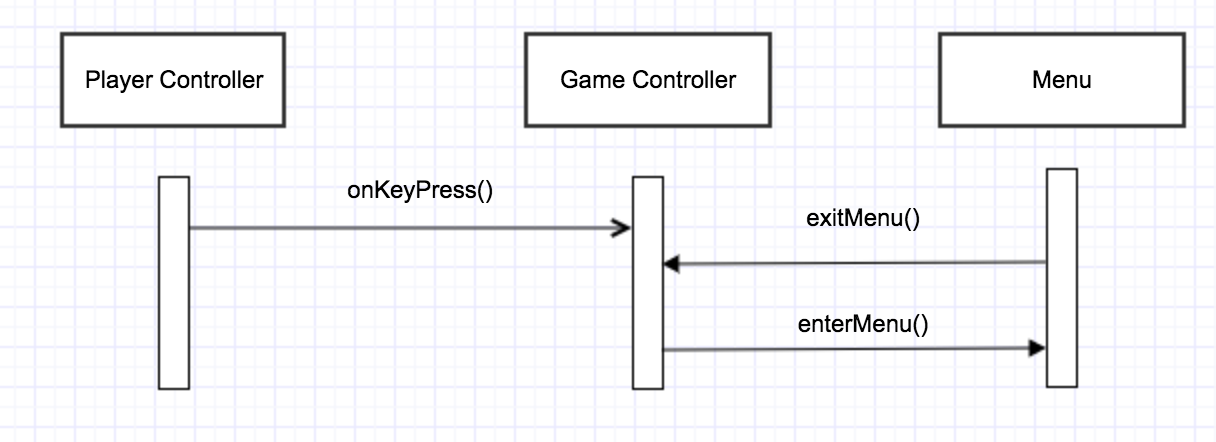
2.2.2. Language and Infrastructure

This script will be written in the C# programming language. It will be handled within the Unity game engine, like all other assets for this game.

2.2.3. Class diagram or data structure diagram

2.2.4. Sequence Diagram

Refer to diagram in section 2.1.4.



2.2.5. Detailed Design

2.2.5.1. Logic/Algorithm Design

Receive parsed input from the game controller. Handle these inputs appropriately (move where user directs movement, pick up items, etc.)

2.2.5.2. State Diagram (if applicable)

2.3. **Puzzle Controller**

2.3.1. Design Overview

This component uses object-oriented design. Its function is to manage the puzzle(s) that the player actively engages with. This includes detecting when a puzzle is active, determining how puzzle components interact, interpreting player input as it relates to manipulating the puzzle and its components, and determining if/when the puzzle has been solved.

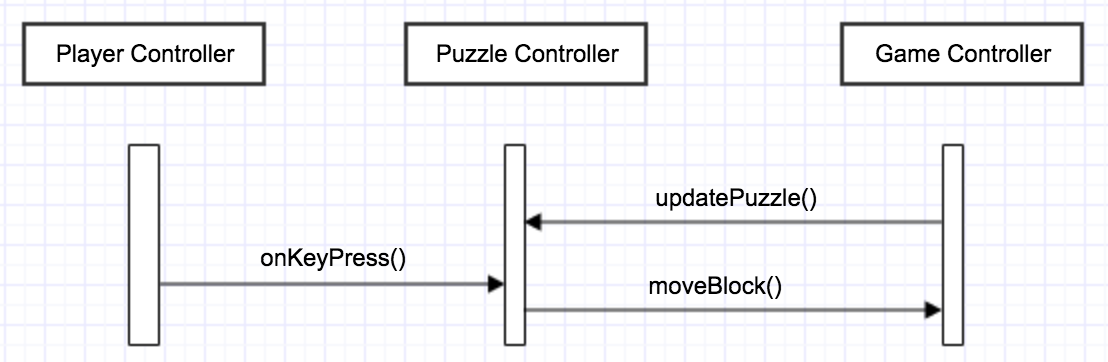
2.3.2. Language and Infrastructure

This script will be written in the C# programming language. It will be handled within the Unity game engine, like all other assets for this game.

2.3.3. Class diagram or data structure diagram

2.3.4. Sequence Diagram

Refer to diagram in section 2.1.4.



2.3.5. Detailed Design

2.3.5.1. Logic/Algorithm Design

OO design is common practice; since we are all familiar with it, we decided to implement it in the coding of this game.

2.4. **Camera Controller**

2.4.1. Design Overview

This component uses object-oriented design. It determines the position and orientation of the camera component that determines the view rendered for the player.

2.4.2. Language and Infrastructure

This script will be written in the C# programming language. It will be handled within the Unity game engine, like all other assets for this game.

2.4.3. Class diagram or data structure diagram

2.4.4. Sequence Diagram

Refer to diagram in section 2.1.4.

2.4.5. Detailed Design

2.4.5.1. Logic/Algorithm Design

Communicates with unity via script: if in the player state, camera follows player movements. Otherwise, the camera will remain stationary focused on the puzzle until completion or termination of the puzzle progress by the user.

2.4.5.2. State Diagram: refer to 2.1.4

2.5. **Items**

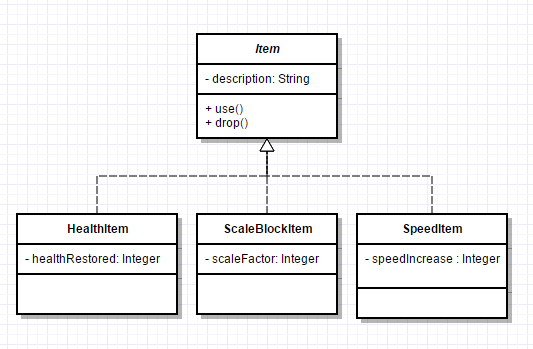
2.5.1. Design Overview

This component uses object-oriented design. It determines the behavior of the various items and consumables that the player can acquire and use in the game world and while solving puzzles.

2.5.2. Language and Infrastructure

This script will be written in the C# programming language. It will be handled within the Unity game engine, like all other assets for this game.

2.5.3. Class Diagram



2.5.4. Sequence diagram for key use cases

2.5.5. Detailed Design

2.5.5.1. Logic/Algorithm Design

Objects that the user can pick up; should lie within the environment. The user will encounter an item via unity’s graphics engine. Upon pickup, the item will be removed from the environment and stored in a list available to the user. These items will have special times to be used (i.e.hatchet to make ‘blocks’ for puzzles, or some other permanent item type for the user to use in special cases)

2.5.5.2. State Diagram: refer to 2.1.4

2.6. **Block**

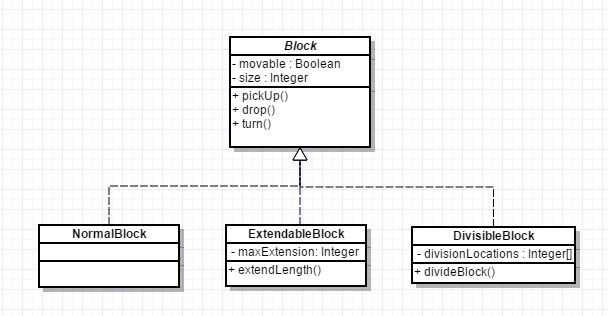
2.6.1. Design Overview

This component uses object-oriented design. It determines the behavior of the puzzle components that the player can use to solve the various puzzles presented in the game.

2.6.2. Language and Infrastructure

This script will be written in the C# programming language. It will be handled within the Unity game engine, like all other assets for this game.

2.6.3. Class Diagram



2.6.4. Sequence diagram for key use cases

2.6.5. Detailed Design

2.6.5.1. Logic/Algorithm Design

Blocks will be accessible by puzzles. These will be pieces that are movable within a puzzle, and should consist of pieces that can be manipulated in an obvious way to complete a pattern.

2.6.5.2. State Diagram: refer to 2.1.4

3. Inter-component communications

3.1. Graphics

3.1.1. An interface managed by Unity controlling what graphics are displayed and how they are oriented. Response times must be quick enough so as not to cause lag between frames. These will be managed by Unity.

4. Metrics

Since this project will be using object-oriented design, object-oriented metrics are most appropriate for verification and validation. Upon analysis, these numbers were not deemed to represent a program of complexity that would be unattainable in the amount of time we have to complete the project. These numbers, although not to scale, represent an approximation of the overall complexity of this project.

4.1. Size information

Size of total program should not exceed 200MB, including graphics packages once formatted into SWIFT (aka, the app should not exceed this limit)

4.2. Complexity

Total Complexity = 9+34+22+27+7+5 = 104

4.2.1. Network Metrics

n/a

4.2.2. Information Flow Metrics

Format: x - #calls(in)\*#calls(out)+#data(in)\*#data(out)

Camera- 0 +3\*3 = 9

GameController- 2\*2+10\*3 = 34

PlayerController- 1\*2+5\*4 = 22

PuzzleController- 1\*2\*5\*5 = 27

Items- 2\*3+1\*1 = 7

Blocks- 2\*2+1\*1 = 5