Think ‘N’ Slide the Box

Concept Overview

Written By

Nathaniel Marshall

v. 1.3

Table of Contents

About this Document

This is a living document; it will be updated as new ideas, concepts, and theories as they are suggested, as goals are met or failed, and as the project progresses.

At its most basic level, this document is to accomplish the following:

1. The One-Pager (High Concept Overview)
2. Outline a basic overview of the project including:
    a. Tools to be Used
    b. Target Platform
    c. Target Audience
    d. Means and Processes used to complete the project.
3. Define the Scope and immediate Primary Goals including:
    a. Gameplay Requirements
4. Strengths, Weaknesses, Risks, Opportunities
5. Define the Unity Engine Conventions of:
    a. Folder and File Organization
    b. Prefabs (saved object to be created dynamically)
    c. Scenes (levels)
6. Define the C# Coding Naming Conventions of:
    a. Class Script Names
        i. Archetype Names
    b. Functions
    c. Variables

The One Pager (High Concept)

High Concept:

Players control a lost and pragmatic Character, using their ability to Kick the Walls of the level to slide the entire world around to reach the Exit of the level. Along the way, Players must be aware of the level’s geometry, including objects that do not move when the Walls are and sliding boxes, as well as Upper and Lower levels that may limit movement in both the Vertical and Horizontal.

Description:

Setting:

A pixelated tower, stylized in the veins of the classic, non-colored Gameboy. Players are climbing a tower, or adventuring through a series of towers/ruins, in search of untold treasure.

Gameplay:

Players control a character whose only action is to Kick objects. These objects include the level’s walls, sliding Boxes, switches, and other usually non-moving objects. Scattered throughout the level are collectables, including keys and one Challenge collectable that requires a certain positioning of the level’s walls for the player to obtain it. Certain levels require the player to Jump up/down Floors using special Jump Tiles. Each Floor is moveable and may have Geometry that extends above/below to interact with the respective floor. Players can only Jump if the Jump Tiles are unobstructed. The player’s goal is to find their way to the exit and collect as much treasure along the way.

Target Audience:

Puzzle-Lovers and those looking for an intriguing take on the classic Sliding Puzzle game.

Target Platforms:

PC, Mobile

Key Features:

- Unique Take on Sliding Puzzle Games
- 'Challenge Gems’, collectables that require an outside-the-box solution on how to reach them.

Genre/ Rating:

Puzzle, Adventure. E Rating (E10+ max).

Overview

At its most basic level, Think ‘N’ Slide the Box is a game where the level’s walls move around the player and the gameplay area is constantly shifting. Within each level, there are smaller puzzles within the walls of the level and interact accordingly as the walls are shifted.

For example, two boxes may be put next to each other and an immobile object so that the boxes stop the moving Wall, providing a two-tile gap when the boxes are again moved. Alternatively, free-moving boxes are shifted as the wall collides with them.

The Tools

Unity3D

Pros:

1. It’s free to work in.
2. Exports to Web Player or stand-alone executable, great for playtesting.’
3. Superb Online/Offline Documentation.
4. If you have a problem, chances are someone else had it. Google is your friend.
5. Component-Based Architecture
6. Personal Familiarity
7. Visual Studio (now free) has a Plugin that allows Debugging with Unity.
    a. No using MonoDevelop!

Cons:

1. GUI, while massively improved, is still best created from scratch.
2. Research needed on Publishing with Unity.

C#

Far more powerful and versatile than Javascript or Boo. Personal Familiarity as well.

The Scope and Primary Goal

The current Primary Goal is to have a working Prototype ready to showcase and be used for formal playtesting (beyond the ad-hoc playtesting using individual levels).

The Minimum requirements of the Game Prototype are:

1. Two Levels
    a. One with 1 Floor
    b. One with 2+ Floors
2. Menu System
3. Credits
4. Options

Gameplay Requirements

For the Prototype, Players and the Game MUST be able to do the following:

MOVEMENT

- Move Up/Down Left/Right on a 2D Plane
- Detect collision properly and *not* walk into Objects/Walls
- When colliding with a Moving Object, said Object Moves the player at speed.
- When Walking into Objects/Walls, they cannot move.
- Kick
    - Moveable Object/Wall, said objects move appropriately
    - Immobile Object/Wall, Player is stunned for a short amount of time.
- Jump
    - When on Jump Tile and Jump Tile is not blocked, kick switch to be launched up/down one level.

WALLS

- When Kicked, Walls move until they run into an Immobile Object.
    - Alternatively, Walls stopped when colliding with Moveable Objects stacked in such a way that they themselves cannot move in the desired direction.
        - Example, having two Boxes against an immovable object will stop the Wall.
- Walls are not affected by the movement of the walls Above/Below unless explicitly depicted through a special warning tile at the entry of the level.
- Walls can Crush the Player if an Moving Wall meets an Immobile Wall with the player in between.
    - This restarts the level.
    - Alternatively, respawns player at a checkpoint.

Game Manager Requirements

The Game Manager must be able to do the following:

LEVEL MANAGER

- Detect Win State
- Detect Lose State
- Keep track of Collectables acquired
- Keep track if ‘Challenge Gem’ is acquired.
- Keep track of Time.

HUD

- Display Time Elapsed Correctly
- Display Collected Collectables correctly
- Display ‘Challenge Gem’ Collected State correctly
- Contain a button that opens the Menu for Mobile

MENUS

- Must be able to work in a LOGICAL matter in terms of UX.
- Main Menu:
    - Choose Level
    - Options
    - Credits
    - Exit (/w confirmation)
- Pause Menu:
    - Resume
    - Restart (/w confirmation)
    - Choose Level (/w confirmation)
    - Options
    - To Main Menu (/w confirmation)

OPTIONS

- Sound/Music Mute
- Key Rebinds (PC)
- Resolution Selector (PC)
- DELETE ALL DATA (/w 2 levels of Confirmation)
- Back

Strengths, Weaknesses, Risks, Opportunities

Strengths

Internal Positives of the Project

1. Personal Strengths and extreme familiarity with Unity3D
2. Very unique and intriguing main mechanic
3. Nostalgic feel of the planned Art Aesthetic
4. Extremely Abstract Art Style might prove be sufficient
5. The most immediate difficult parts of the project is getting the mechanics working, the rest is Level Design

Weaknesses

Internal negatives to the project.

1. Shifting the Walls might be extremely confusing and hard to comprehend
2. Art/Animations will need to be researched/completed
3. While simple in paper, implementation of the sliding walls is temperamental
4. Level Design and Tutorials for such an obtuse Mechanic is something to be carefully constructed

Risks

External issues the project might face.

1. Personal Responsibilities (work, job hunting, etc)
2. Time Constraints/lack of Time to be made available
3. Need to Research Publishing Unity Games, may run into unforeseen complications
4. Finding people to Playtest the game
5. Marketing: Research and begin getting the name out.

Opportunities

External things the project can take advantage of.

1. Currently learning Unreal, Possibility that Project might be a perfect fit for skill level
2. Unity Asset store
3. Finding a Freelance Artist available to create quick Art.
4. Borrow friends and peers, use online Game Development forums for playtesting
5. Marketing: Game Dev forums may help, contact YouTube and Twitch personalities.

Unity Engine Naming Conventions

From the start, it is wise to know how to name things so you can stick to that naming convention throughout the project.This section deals with how the project is set up in Unity proper.

Folders and File Organization

Files will be sorted by either Type or Utility. This will be moved to the TDD.

NOTE: The '_' is to set the folder at the TOP of the Asset list as it will be frequently used.

Notable folders

_ProjectFolder – Contains everything but items from the Asset Store. If we are to use anything from the Asset store, they will be stored in the top-level Asset folder and, no doubt, will get lost in the muddle of the many files that are to come. Asset Store files shouldn't be moved due to how they update, so we lump everything into the Project folder.

- _2DMaterials – Contains the Materials created from the 2DTextures folder.
- _3DMaterials – Contains the Materials created from the 3DTextures folder.
- _Audio – Contains all the sound the game will use
    - Audio_SFX – Contains the Sound Effects the Project will use.
    - Audio_BGM – Contains the Music the game will use.
- _Prefab – Contains objects saved into the Assets (Prefabs)
    - Prefab_PickUp – Storage place for the Collectables of the game
- _Scene – The folder where all Scenes will be stored.
- _Scripts – The folder where all Scripts will be stored and further categorized on utility.
    - Scripts_MANAGER – Contains all the managers used in the game.
    - Scripts_PickUp – Contains Scripts for the Collectables for the game.
    - Scripts_UTILITY – Contains Scripts with Global functions/Variables.
- 2DTextures – Folder for 2D Images such as the HUD, and further categorized on context.
- 3DTextures – Folder for anything to be placed on a 3D object, and further categorized on context.

General Naming Convention

Along with the sorting into the correct folder, files will be named to be easily understood what they are and where they came from using the following:

[SUBTYPE]_[NAME]_v[VERSION NUMBER].

The version number is optional.

Examples:

The second iteration of a prefab in the Unit folder: UNIT_Char_v2

A Level Manager Script: MANAGER_Level

A Test Prototype level: LEVEL_TestPrototype

Coding Naming Conventions

Class Script Names

Script names follow the above convention.

[SUBTYPE]_[NAME]_v[VERSION NUMBER]

The sole exception is for Utility Scripts, where they are named after their Utility.

Archetype Names

Archetypes are Scripts that provide the basic foundation that other scripts will derive from. These have their own distinct naming convention.

ARCHETYPE_[Name]

Function Names

Function names will be categorized by their role in the script and named appropriately.

Example:

A function in the Unit Class that modifies the HP Value. As HP is a Stat, the function goes

public void STAT_Damage()

Variables

Similarly, Variables are sorted by their role, then by their name.

Examples:

STAT_Def

STAT_HP_Current

STAT_HP_Maximum