Comp 8051 Final Report for IOS game: Mr Marble’s Maze

By Nathaniel Zeeman, Paul McCarlie, Jason Hutton

# Our Team

Team members and their respective roles:

* Nathaniel Zeeman: Architecture, physics, optimization
* Paul McCarlie: UI, Music, Graphics
* Jason Hutton: Sound effects, settings, bug fixing, level node construction

# Game Overview

## Story

Your name is Mr Marble and you are a marble trying to roll away from falling lava

## Gameplay

Our app is a falling marble game in which you guide a marble down a vertical maze by tilting the phone left and right.

Players can also tap the screen to invert the gravity.

There is a descending lava cube of doom that falls downward at an accelerating rate, so the player must guide the marble downwards through an infinite series of platforms in order to stay alive for as long as possible.

## Score

Score is determined by how long the player stays alive.

# Game Features

Some of our game features include:

* 3D rendered, textured game environment
* Box2D physics and collision detection
* Ball rolling animation
* Saveable highscores: the game will save your 3 highest scores
* Mute-able music and sounds
* Sound effects when the ball hits a surface (Context-sensitive)
* Randomly generated nodes of predefined groups of platforms

# Player Profile

Casual player with time to kill, e.g. on public transit. Not too committed or hardcore, but still wants a reasonable level of challenge. Similar to the “endless running” games demographic

Ages: 12+

Both male and female

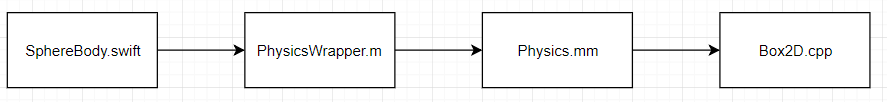
# Game Architecture

## Entity Component System

* Our game makes use of an Entity Component System, with interfaces based on Unity’s
* Entities, or game objects, exist in a hierarchy - each game object has a parent and a collection of children game objects
* Game objects also hold a collection of components that perform specific behaviour
* Components have several lifetime functions that are automatically called, ie. onEnable, onDisable, update
* Components can be increasingly specialized through inheritance

## Box2D

* Our game also makes use of the Box2D physics system, which interfaces with the swift code using 2 (two) wrapper classes:



* Interface of the physics and physics wrapper classes is simplified and abstracted from the actual Box2D calls, for the sake of sanity

## Level Generation

* Levels are infinite and randomly generated
* Our level loader takes in several JSON files known as nodes, these contain information on where platforms will be placed for that particular section
* Nodes are randomly selected, and stacked vertically
* Nodes are generated a fixed distance below the player - this is never visible
* Nodes are cleared away by the “lava death wall” that follows the player

# Alpha accomplishments

* Create a proof of concept
  + Create game architecture (Entity Component System, basic game loop)
  + Load and render a ball and a game environment

# Beta accomplishments

* Get physics system implemented (this was the major goal)
* Basic main menu
* Implement music
* Implement graphics i.e. textures, shaders, lighting

# Final Goals

For our final release, we had the following features planned:

* Sound Effects *(met)*
* Implement a score system *(met)*
* Score saving  *(met)*
* Game Over screen *(met)*
* Way to quit and restart the game *(met)*
* Implement a falling wall of doom that makes the player lose *(met)*
  + Balance falling wall speed to make it fair *(met)*
  + Create some kind of telegraphing to warn the player it is near *(met)*
* Load levels  *(Goal changed: Game is one single unending level.)*
* Bug fixes and optimizations *(met)*
* Background image for game  *(not met)*

# Testers

* Andrew Busto
* Chaela Chalmers

# Testing feedback

Of the little feedback we were able to obtain, here’s what was recommended:

* Change the mute button to make it more clear when it is muted and not
* Change the Death Wall texture to something more noticeable
* Make it more obvious to the player how to control things (give some instructions)
* Make it clear to players what the goal is
* Game should better utilize the gravity flip
* The Death Wall is a little overbearing and feels unfair when it accelerates and kills the player without warning
* Have a separate mute for music and sound effects
* A background image would be distracting and make it difficult to see
* Music is enjoyable and suits the game.
* Game is fun, especially in the more complex areas, very playable.

# Things that went well

* Group members (of those present) were willing and eager to work
* Gitflow in general went well; all group members were careful to not override others work during merges
* Problems and progress (or lack thereof) was communicated clearly and effectively between group members, either face to face or over Slack
* Game idea was flexible which made it easy to to reduce the scope
* Entity Component System allowed for easy implementation of behaviour for individual game objects and memory management
* Node level generation came together quickly, allowing for parts of levels to be designed using JSON

# Things that did not go so well

* One group member dropped the course
* Documentation was occasionally difficult to find, out of date, or unclear
* Some libraries (Box2D) were tricky to implement due to not working natively with Swift
* Initial game structure setup (Box2D, shaders, ECS) took more time to implement than anticipated
* Group members did not manage their time well, lot of work was done too late
* Working on a Mac OS Emulator often slowed progress
* XCode caused issues.
* Mac Simulator was of limited use, due to inclusion of motion controls.
* Provided Apple iPod Touches were quite slow, also slowing progress

# Things we would do differently next time

* Use fewer languages, as the choice to use Swift introduced complications even though it seemed beneficial on the surface.
* Look into the technology we want to use (Box2D) more carefully before choosing - we ran into a lot of compatibility issues which took a lot of time to resolve
* Create more realistic expectations for what our final app will look like
* Maybe choose a less difficult idea (that doesn’t require too many external libraries)
* Better manage our time

# Things we’d improve or expand upon

If we were to continue this project, some things we might do are:

* Make more sections with greater variety
* Greater variety of platform patterns
* More complex areas
* Different types of platforms with different properties
* Implement a separate game mode made up of levels
* Utilize a greater variety of sound effects
* Improve the art: better menu artwork, subtle background image for the game
* Add enemies that shoot at the player
* Add collectables for extra points
* Test more comprehensively and fix bugs

# Screens

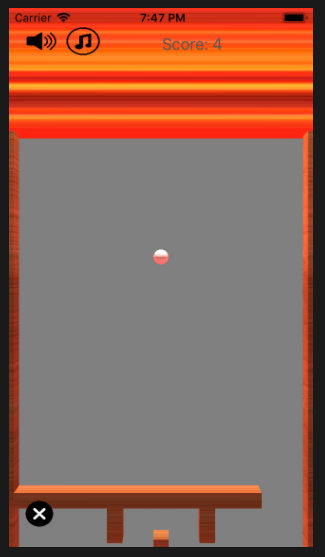
## Main Menu:



## Instruction screen:

## 

## Game Screen:



## Game Over Screen:

