University of Arkansas – CSCE Department
Capstone II – Final Report – Fall 2019

Tilted

David Green, William Harris, Drew Rudasill, Connor Dean

Abstract

The objective of this project is to give all of us more experience in programming. It will help expose us to more tools that are popular to use in game development today. Our approach is to use the Unity Engine in order to create a mobile game that is able to take advantage of a phone's gyroscopic capabilities.

1.0 Problem

Developing mobile games may not provide the world a solution to their dire problems, but what they can provide still serves to improve the quality of life for all around us. There can be many problems tied to the purpose for a mobile game; such as location, or accessibility. For instance, people that don't have access to their consoles or computers on campus. Or consider people that don't have any consoles at all. And, perhaps the most common, something to pass the time. Mobile games provide something to people that is perhaps overlooked; entertainment. Many people use entertainment as a method of temporary freedom from their hardships, and it takes many forms. It could be from work, school, relationships, and many more things. The lack of entertainment would bring down the overall quality of life by a measurable amount, even if it is small.

2.0 Objective

The objective of the project is to implement a "marble platformer" in Android Studio. The player will control a marble and must navigate it through a course to reach the goal. There will also be a time limit they must overcome on top of other challenges. However, there are other sub-goals that we'd like to reach in doing this project. The first step is to decide how we wish to build our application and from there we can carry out our other objectives such as implementing a leaderboard system, creating a database schema for keeping track of users, and the game mechanics itself. Another primary objective of ours is to implement both gyro and touch controls.

3.0 Background

3.1 Key Concepts

Unity

- Unity is the most popular game engine there is for developers. It's not only a framework with which we run our code on, but it also acts as an edit tool. Unity utilizes Views that allows us to manipulate the objects as we are building. It also allows us to view our game in real time while making changes in the code. This being supported on Android is what makes the project feasible.

Database

- A database is how we will store information from our users such as their ID. We need this because there are certain features that we want to implement and require a table to pull information from. For instance, if we want to implement a high-score leaderboard for a certain level, we can pull from each user their best times of that level and display the top ten.

3.2 Related Work

Other mobile games, that involve a marble specifically, have a similar problem in that there is no actual control for the marble. In Marble Run, users are allowed to design their own levels where they watch marbles race on the track, but no interactive controls for the users, with a terrible user interface. Marble It Up is a great example of what we want to accomplish. It features platforming and free movement over the marble, with difficult levels. However, its only available on PC and thus, being mobile will provide a better gaming experience. Additionally, we will provide two different methods of control, gyro and touch.

4.0 Design

4.1 Requirements and/or Use Cases and/or Design Goals

What we want by the end of this project is a competent, fully functional, basic android mobile game, which provides reasonable challenge, can be played offline, but implements some network features, and has at least two unique control schemes. Network features include things like high scores and other user information. The game's UI should be fairly basic, and made of only a few easy to navigate menus, which can be interacted with by touch.

The gameplay will be simple. You must navigate a ball to an end goal, using either tilt controls, or touch controls. Levels should be designed in such a way that they are challenging, but not overly punishing. Scores and other user information should be saved offline locally upon completion of a stage If the user is online, those scores will be submitted to a database, which the game will query to fill some sort of table for high scores, or best times. Control schemes *must include* both a tilt control option, and also some sort of touch based control. Mobile games can be played in transit, while on a bus, in an uber, etc. Gyroscopic functionality may be impaired while in transit.

4.2.a Detailed Architecture - Assets Overview

Using Unity's entity component architecture, our project and its assets are broken up into a number of folders.

- Scenes
    - These represent our various levels and menus. In Android studio, these would be our Activities.
- Animations
    - Animations and Animation Controllers live here. All of our custom model animation are controlled from here.
- Audio Mixers
    - Not used as often, but still important. The game's audio is split into several channels, representing timescaled, and unscaled sound effects, as well as channels dedicated to menu sounds, and in-game sounds.
- Custom Models
    - All of our assets in this game are entirely our own work, and that includes the handful of custom 3D models we've created for the game, including the goal flag, bumper, and gimmick pads.
- Materials - Physics
    - Physics materials determine object friction, weight, and handle collision data.
- Materials - Textures
    - Our textures have to be applied to materials in order to use them in a 3D space. We have a number of colors for specific objects saved here, as well as the animated version of the boost pad texture, among other things.
- Prefabs
    - Everything that can be used across scenes is saved here, including pre-built platform configurations, custom models sized to fit in with the rest of the world, and even menu elements.
- Raw Textures
    - Before a material is created, we have to import all of our custom textures into the raw textures folder.
- Scripts
    - All of our code is located here. A section farther down will go into detail on code functionality.
- Shaders
    - In order for an object to appear in the game world, a shader must be applied to it. Shaders influence how light reacts on a surface, and how textures are rendered on objects. The game only really has three shaders. One for the marble, which keeps it from becoming obscured by objects. Another exists for all other 3D environmental objects, and gives the game more of a bright, hard shadowed look. The last is the default shader used for anything else not covered by the first two.

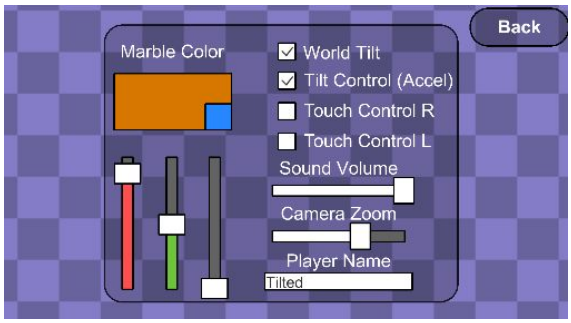
- Sound Effects
    - The sound effect folder is dedicated to raw sounds. These raw sounds are pushed through the appropriate audio channels mentioned above, to keep things from playing over one another.
- Sprites
    - This folder contains data pertaining to non-ui elements used in a 3D context. They aren't considered textures, or materials, and are kept separate.

4.2.b Detailed Architecture - Game Flow

When a user starts the game for the first time, they will be greeted with a player name entry screen. This name can be changed at any time -- even in the middle of a level.

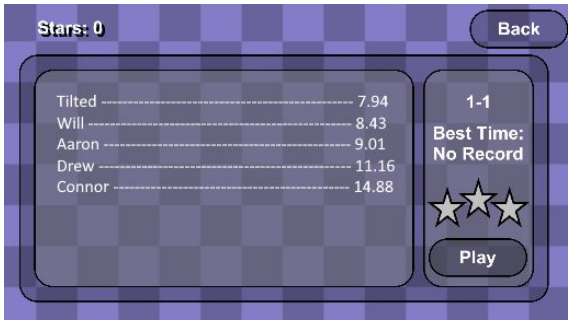
After confirming a name, the user is greeted with the title screen. There are three options; Play, Options, and Quit. Selecting Play will continue to the Stage Selection screen. Selecting Options will bring you to the Options screen, and Quit will close the program.

On the Options screen, you may change the game's camera settings, control scheme, marble color, audio volume level, and modify player customization settings including name and marble color. Hitting back will take you to the main menu again.

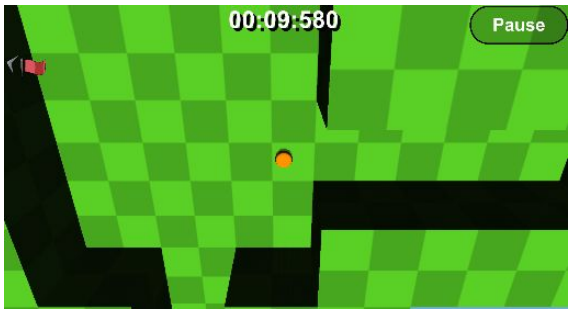


The Stage Selection screen contains two lists. One holds worlds, and the other holds stages. In the top left, a counter lists how many Stars you have. Worlds 2 and 3 are locked currently, and require the amount of Stars listed on their lock open them. To obtain Stars, you must clear levels as quickly as possible. The game grades out of 3 Stars per level. To select a world and stage, simply tap.

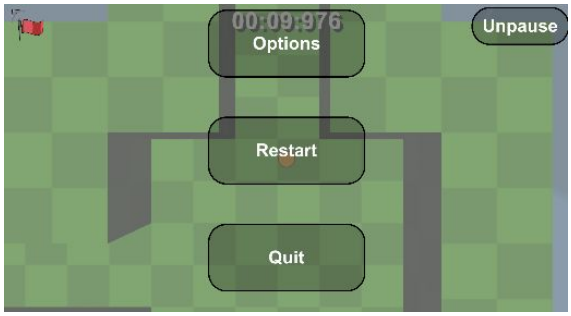
After selecting a level, a new screen displays information about your best time, and current Star Score per the given level. Leaderboard information is shown to the left of your personal information, which will list the top 10 scores, if they are available.



Upon hitting the play button, the selected level will be loaded and the user will begin play. A short countdown will display, and the level will begin. Depending on what control scheme the player has selected, the game may or may not show the analog input graphic. The top counter is your current level time, and the flag indicator on the left is the flag guide.



The player may pause at any time (unless waiting for a countdown to complete). Upon pausing, the game freezes, and the player is given four options. Unpause, Options, Restart, and Quit. Unpause will unpause the game. Options will open the Options screen. Restart restarts the level from the beginning, and resets the timer. Quit sends the user back to the leaderboard stage screen.



4.2.c Detailed Architecture - Data and Structures

Player Input:

Player input is handled in one of three ways.

- Motion Input (Accelerometer)
- Motion Input (Gyroscope)
- Touch Input (Virtual Analog)

When the game first boots, Motion Input is selected by default. The game then checks whether or not the device it is being played on has a Gyroscope or not. If it does, it automatically selects the Gyroscope as the Motion Input method. If it does not have one, it will check to see if an Accelerometer is present, and select it if it is. Gyroscope is prioritized as it is more accurate, and can be calibrated at any angle, where an Accelerometer must be flat for it to work. If no motion devices are present, the option will be disabled.

At the beginning of every level, a short countdown will display. The beginning of this short countdown also acts as a calibration period for Motion Input, so that the player can begin tilting the device before the countdown is over, and immediately begin moving in the direction they wish, as fast as they can.

All input methods are normalized so that no particular input method grants greater speed than any other

Game Physics:

The physics in this project are mostly based on the default 3D physics system built into Unity3D. Originally we tested trying to tilt the entire world to make the marble move, but it was considerably more resource heavy, and also afforded us much less control of the ball. As it stands now, the camera moves around the ball to create an illusion of a shifting world. A few small tweaks to the physics engine have been made to give greater control of the marble to the player. Several custom materials have been added to the project as well, for greater collision diversity.

Game Gimmicks:

Shifting Platforms - Shifting platforms are those which translate themselves on an axis, by way of adjusting their transform.position through script, or animation. These platforms require the use of rigidbodies for proper collision.

Rotating Platforms - Rotating platforms rotate about an axis uniformly at a set speed by adjusting the transform.rotation. These platforms do *not* require rigidbodies for proper collision.

Invisible Platforms - Platforms which toggle between visible and invisible on a timer, by disabling their meshrenderers. They can also be turned on or off via a linked button.

Intangible Platforms - Like the Invisible Platforms, Intangible Platforms will toggle between visible and invisible, but will not only have its meshrenderer disabled, it will also have its colliders disabled as well. Similarly to the Invisible Platforms, it can be linked to a button for manual transitions.

Bounce Pad - Bounce Pads are objects with colliders which abruptly launch the marble into the air at a set speed.

Boost Pad - Boost Pads are similar to Bounce Pads. Instead of launching upwards, they launch in whichever direction they are facing at the time, at a set speed. The texture for this object required a unique shader, which would shift the set material in the direction the object was facing.

Cone - A simple physics object affected by the built in gravity. It doesn't do much, and isn't animated, but was the first thing we built in Blender.

Bumper - A somewhat complex physics object, built in Blender. Consists of two parts. A non-animated base, and an outer ring which is animated. When the player collides with the trigger collider, the bumper rings shoot outward rapidly. The player is launched away, and the bumper's sound plays.

Button - The Button is a multi purpose… button. Virtually any scripted event can be linked to it and run when the player collides with the Button.

Saving and Loading:

For saving and loading our game's data and player preferences, we used Unity's built in PlayerPrefs, but with a slight twist. Early in our development we decided to make the game entirely functional offline, which means all data has to be saved locally. This caused issues with leaderboard functionality, because we'd reference the locally saved data for the leaderboard. This would be easy to cheat without an extra layer of security. The security isn't foolproof, but it is at least hopefully discouraging to anyone who might attempt to cheat the game's saved data. Everything is encrypted with really basic DES security, and a basic key.

Leaderboards:

The game assigns a Unique Device ID to a player name, so that if they choose to change their name their scores will still remain. Our ideal leaderboard is for every stage, the top 10 fastest times will be recorded and displayed. To make this simple and easy, we use SecurePlayerPrefs, which is a modification of the class PlayerPrefs in the Unity Engine. Using SecurePlayerPrefs, we assign a string of the form <World,Stage> to a float value <Best Time>. This makes uploading and downloading scores to the database very easy. We constantly update "Current World" and "Current Stage" based on where the player is in the menu, and we simply query the database with these values to obtain the highest scores from the respective

world and stage. The same applies to uploading high scores. Our logic for uploading scores is as follows: We check if the player has a time on the current level on the client, if not we do nothing. If there is a score, we then check to see if the player, with their Unique Device ID, as a score for the level already, if there is then we compare the scores to see which is faster. To prevent from constantly querying the database if the player has a score, we keep a boolean variable that only queries the database if true. After the first query, we set it to false until the player has obtained a faster time locally. Then repeat the process. For downloading scores, we query the database the first time a player selects a stage in the menu. To prevent querying the database multiple times, another boolean is used. We only want to query the database again if they have returned to the world select and selected another level. We chose to use dreamlo as our backend, because it was free and easy to use. It imposed limitations that made implementing it for our purposes difficult. In the future, we may stand up our own backend to further specialize the schema to better support our game.

4.3 Risks

Risk	Risk Reduction
Designing games is a potentially challenging task. It all depends what's in it. For instance, building your own game engine could be its own capstone project.	Instead of designing our own, we're using Unity; a well known, and *super* well documented engine.
Connection to the internet can cause all kinds of security issues. People finding information they aren't supposed to find and using it with malicious intent.	Secure messages through encryption and decryption when internet is required for a task.

4.4 Tasks

These tasks are more or less in the order they need to be completed, but there may be some performed out of order as needed.

1. First of all: Gain a better understanding of how Unity works. Run through some tutorials.
2. Design in detail game flow, as in, how menus are connected to levels, how they function, and in what order, etc.
3. Program basic game physics
4. Program basic UI interaction (basic menus, control options)
5. Initial device testing

6. Level design
7. Level implementation
8. Decide on and design network features
9. Implement network features
10. Secondary device testing
11. Bug fixing
12. Prepare app store version, and associated materials for storefront
13. Deploy to app store
14. Document

4.5 Schedule

The needed tasks can more or less be broken down into weekly goals, where some tasks may be given more time, and others less. This timeline fits almost perfectly into one semester.

Tasks	Dates
1. Learn Unity's ins and outs.	8/19-8/25
2. Design "Game flow"	8/26-9/1
3. Program game physics	9/2-9/8
4. Program UI interface	9/9-9/15
5. Test on actual android devices	9/16-9/22
6. Design actual levels	9/23-9/29
7. Build actual levels	9/30-10/6
8. Design network features	10/7-10/13
9. Implement network features	10/14-10/20
10. Secondary device testing	10/21-10/27
11. Fix anything that's still broken	10/28-11/03
12. Prep app & storefront materials	11/04-11/10
13. Deploy app to store	11/11-11/17
14. Document our process	11/18-11/24

4.6 Deliverables

The final product should have these associated deliverable items:

- A final, full app to be visible on the app store.
- If necessary, a design document visualizing work done in Unity.
- Full game code in C#.
- Final Report

5.0 Key Personnel

David Green- Green is a Senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas.
He has completed: Software Engineering, Operating Systems, Database Management, Programming Paradigms, Networking,
Has experience with: Machine Learning, Android Studio, C++, Java, SQL, Hadoop, Spark
Is responsible for: Leaderboard and schema logic, UI logic.

William Harris- Harris is a Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas.
He has completed: Database management, computer graphics, programming paradigms.
Has experience with: C++, Java, SQL, Managing databases, game design, UI design.
Is responsible for: Custom models, UI elements, sound, and level design, level scripting, shaders.

Drew Rudasill- Rudasill is a Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas.
He has completed: Software Engineering, Database Management Systems, Operating Systems, Programming Paradigms
Has experience with: C++, Java, SQL, Python
Is responsible for: Documentation, control and input logic, saving and loading.

Connor Dean- Dean is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas.
He has completed: Programming Foundations I & II, Computer Paradigms
Has experience with: C++, Python, Java, JavaScript, C, artificial intelligence, game logic, and game design.
Is responsible for: Helping to make the game work.

6.0 Facilities and Equipment

- Unity software
- Emulated/Physical Android phones
- Database host