imt3673 Project, Gravity Ball - Report

Adam Jammary
Magnus W. Enggrav
Michael Bråten
Martin Bjerknes

04.05.2018

Contents

List of Figures

1 Introduction

Our project is an expansion of lab 3. We had to make a game around the mechanic of using gravity to control a ball. We also had to integrate that game with google play services. We choose this as our project because we liked the idea of a sensor based game.

2 Development Process

We used an agile development process. The tools used in our process were all provided by github. Issues were used for tracking features and bugs, we organized issues in a sprint board under the projects tab in the repo. By using github for our sprint board in this way we could reference each issue using smart commits. In practice not all team members remembered to use smart commits for every commit, in which case we might compensate by referencing the issue in the resulting PR.

All of us developed in feature branches to keep master clean of bugs and partial features. When a feature was completed we created a pull request. Pull requests need 1 review from another team member to be merged into master, this is a form of quality control for us. Having another team member review your PR usually makes it so that the code/feature gets looked at from a different perspective. Having someone else review also makes it so that the feature gets tested on different hardware. This is helpful because android is a rich ecosystem with a lot of different phones whit different screen resolutions, different hardware specifications and different versions of android. With that being the case, the fact that something works on your phone is no guarantee that it will work on other peoples phones as well.

3 Design

3.1 Game Design

3.1.1 Intro

The game is about guiding a ball towards a goal on obstacle filled levels using gravity. The completion time of a level is timed, the achieved time is the score. There are 3 tiers of times to beat on each level, meant to challenge the player. Scores are added to a local leaderboard for each level and an opt in online leaderboard.

3.1.2 Ball

The player controls the ball by rotating their phone. The ball should always fall towards the center of gravity in real life. The degree of the tilt on the phone effects how quickly or slowly you accelerate the ball. This allows the player to apply fine control to the ball or apply a lot of speed by using an aggressive tilt. When the ball hits an obstacle it should bounce and loose velocity. The loss of velocity prevents the ball from accelerating forever, making it manageable for the player.

3.1.3 Levels

The levels are linear and 1 dimensional in the horizontal direction. This gives the player a clear overview of the level and a clear path towards progression. The player always starts at the left side of the level and has to reach a goal on the right side. The levels are filled with 3 obstacles that the player can interact with:

1. Wall: Static objects acting as obstacles, hitting them causes a collision.
2. Destructible Crate: Acts the same as walls except that they will break when hit with sufficient force. Breaking is communicated to the player through a change in texture and by a transfer of momentum causing the crate to slide away. After sliding for a small period the crate disappears from the game.
3. Hole: They act as triggers, hitting them does not cause a collision, it instead teleports the player back to the spawn. These are meant to challenge the player in fine control, they have to be careful and pay attention to avoid them.

3.1.4 Score

The score in this game is the time it takes the player to complete a level, so lower is better. Every level has 3 tiers of score goals for the player to beat. This is meant to give the player an incentive to replay the same level in order to improve their score. The 3 best scores of the player on a level are also stored

as further incentive for the player to beat themselves. We also have a social incentive for the player to improve their score through an online leaderboard, with the leaderboard they can compete globally against other players.

3.2 User Interface

3.2.1 Start Menu

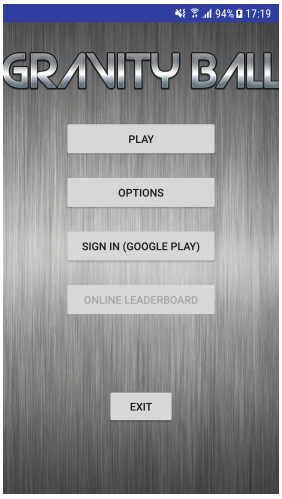

Figure 1: Start Menu UI.

This is the first menu you see when lunching the application. It contains five buttons:

- **Play** will navigate to the level chooser.
- **Options** will navigate to the options menu.
- **Sign in (Google play)** will sign the user in with Google play.
- **Online leaderboard** will show the online leaderboard for selected level.
- **Exit** will exit the application.

3.2.2 Options Menu

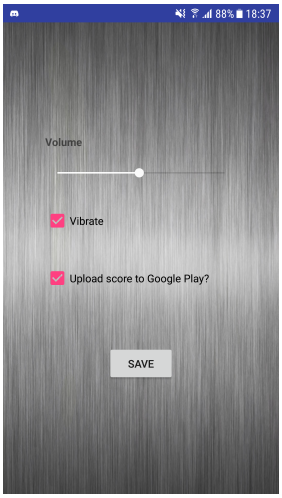

Figure 2: Options Menu UI.

In the options you can control the volume (not yet implemented), turn on or off vibration and choose if you want to upload your score to Google Play.

3.2.3 Level Chooser

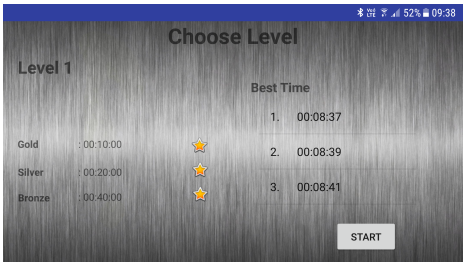

Figure 3: Level choosing UI.

We wanted the player to be able to choose which level they wanted to play and at the same time give them a challenge to beat different times, beating the times meant earning stars which would also give the player an urge to beat the level as fast as they could. If the player had beat all the level times, displaying a local scoreboard of the top three best times, could also give the player motivation to compete with themselves and beat their own time.

3.2.4 Win Screen

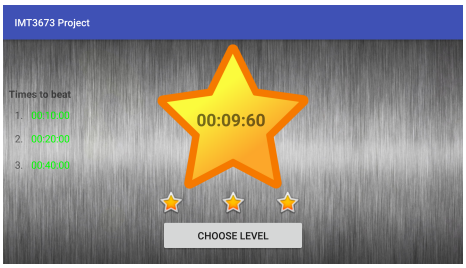

Figure 4: Win Screen UI

After a player reaches the goal a win screen will be displayed (see Figure 4). The win screen will display the time used on this level, the times the player needed to beat, and how many stars the player earned through this level. If the new time is better than the times to beat, the times to beat will be marked in green. When the win screen starts, the stars earned will do a rotation animation to make it more visual how many stars they got, the big star in the middle will continuously do a scaling up and down animation so the win screen looks a bit more alive. The player can press a button to go back to the Level Chooser.

4 Features

4.1 Google Play Games Services

4.1.1 Authentication

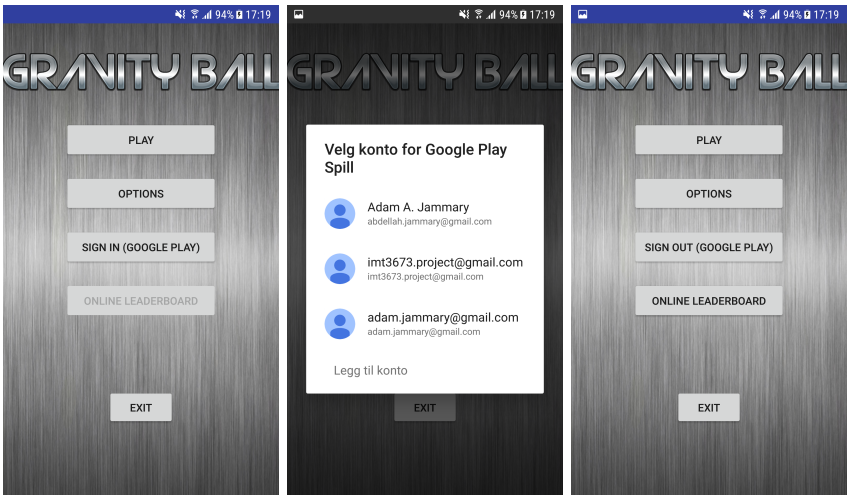

Figure 5: Left to right: Sign-In flow

1. The user is not signed in by default. As you can see in Figure 5, the user has to manually choose to sign in to Google Play from the main menu.

2. The sign-in process will first check if the Google Play Games app is installed, the app is required to link a gmail account to any Google Play Games services like Leaderboards.

3. If the app is not installed, the user will be taken to the app store, then requested to install the app, and finally open it so they can set up which account or accounts they want to use for Google Play Games Services.

4. The above mentioned process will only happen the first time the user signs in. Every other time the sign in process will always try to re-use an already authenticated user silently without showing any login UI.

5. But if no authenticated google account is found, the user will be presented with a UI which is completely handled by the Google Play Services. The user will then be asked to pick an account and sign in.

6. If the user was authenticated successfully, the sign in button will be changed to a sign out button, and the online Leaderboards button will be enabled and functional.

4.1.2 Online Leaderboards

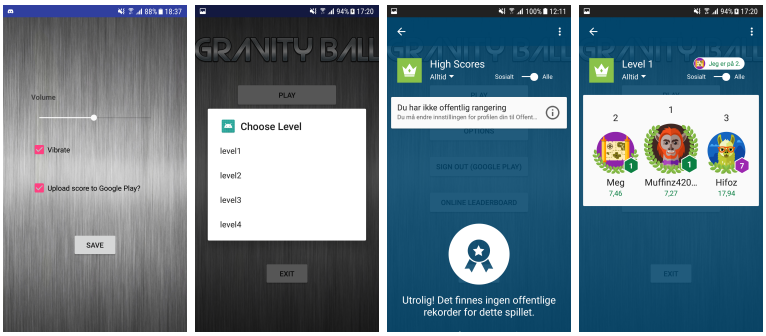

Figure 6: Left to right: Options, Leaderboards: Choose, Empty, Scores

The user needs to sign in to Google Play with special permissions to access Games services like the online leaderboards etc. But the user does have the option of not submitting their score to Google Play. This is necessary, because we could implement features in the future that requires user authentication with Google Play but maybe not any access to Games services. The user can easily opt-out of submitting their score in the Options menu as shown in Figure 6. We have linked each leaderboard to a level, and we have currently implemented four levels, so the user can choose between four leaderboards.

4.2 Level Creator

To create levels we draw them in paint, and use the drawn bitmap to build a level in game. The class responsible for building a level from a bitmap is `Level`. We start by creating an instance of `Level`, then we call `Level.buildFromPNG(...)`. The levels created are scaled with the height of the phone, so that the entire height of the drawn level spans the height of the screen. This means that we can draw levels of any resolution.

The different colors used when drawing the level represents different entities in the game. Black pixels are walls, red pixels are destructible boxes, cyan pixels are holes, green pixels are goal blocks and blue pixels define the spawn point. The level editor uses these pixels to build instances of `RectF`, which become the in-game level. When building a level from a bitmap the level builder iterates through every pixel, when it encounters a pixel with one of the defined colors it calls `createRect(...)` on that pixel. `createRect(...)` searches for more pixels of the same type as the input in the x and y direction to work out what the biggest `RectF` it can make is. Once it finds the width and height of the `RectF` through the search it clears all included pixels to prevent them from being included in other rectangles and creates a `RectF` from the data. Creating rectangles in this way, instead of creating a rectangle for every pixel helps with performance. When doing collision detection it does not matter how

big a rectangle is, the amount of rectangles you have to check against however impacts performance. See Figure 7 for an image of a bitmap and the resulting in-game level.

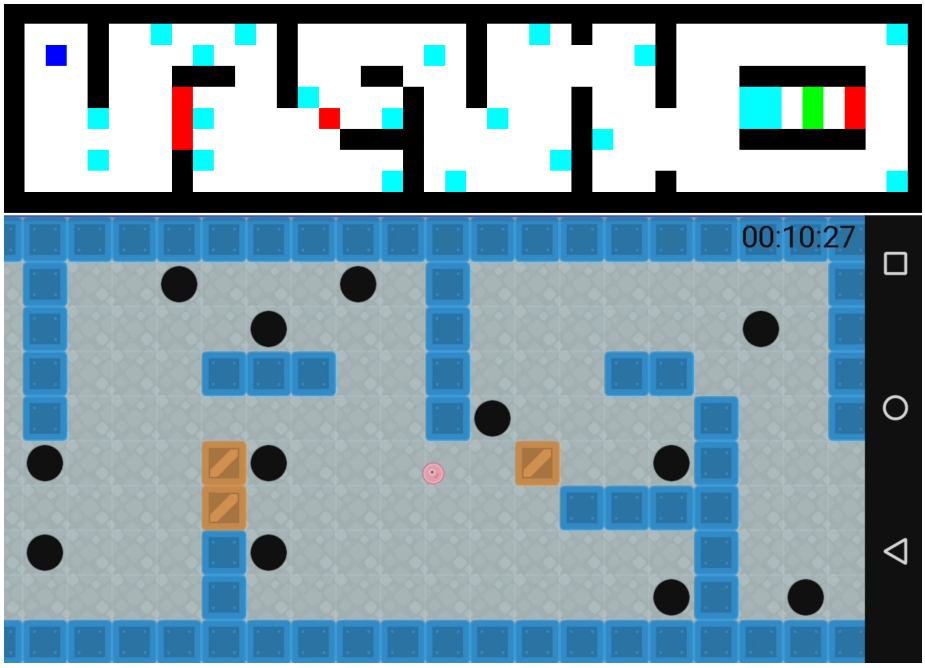

Figure 7: From top to bottom: Source bitmap of a level and the resulting level in-game.

Rectangles in the level are also grouped together into collision groups. A collision group is a 20 pixel horizontal segment of the original bitmap of the level. So a 10x60 bitmap level would have 3 collision groups. When creating rectangles you also add that rectangle to every collision group that it intersects. This helps us scale up levels by only doing collision detection for the collision groups that the ball intersects. This approach is inspired by quadtree collision detection, we did not need to implement quadtrees as our levels are mostly horizontal, so doing horizontal grouping is good enough.

4.3 Physics

The ball physics update has 2 steps:

1. Update velocity with data from accelerometer.
2. Update position with velocity.

4.3.1 Velocity update

Updating the velocity is fairly straightforward, we add the acceleration from the accelerometer to the velocity. We also use two scaling factors for the update: `accelDelta` and `zFactor`. `accelDelta` is just a const value used to bring the acceleration down to a manageable level for the player. `zFactor` is calculated and used to simulate table friction, it approaches zero as the Z component of the acceleration data increases. This makes it so that you can more easily control the ball when holding the phone level.

4.3.2 Position update

When doing the position update we update the X and Y components independently. This is so that we know which component to blame in collisions. To update one component of the position we add velocity to that component. After the position update we check if it caused any collisions. We check for collisions against every rectangle inside the collision groups that the ball intersects, the collision check used is a circle-rectangle intersection function [1]. If a collision does occur we resolve it by reverting the position component value back to its previous value. We then flip the velocity of the component and multiply it by a constant smaller then 1 to simulate loss of velocity caused by the impact.

4.4 Local Leaderboard

To store local highscores we created a RoomDatabase by using the room persistence library [2], the database consists of a single entity called `HighScore`, this table contains a `levelName` and a `levelTime`. The `HighScoreDao` has the method `insertAll(HighScore)` that will save an HighScore entity to the database. This is used in the `MainActivity` when the “Win screen” is displayed to the player. `HighScoreDao` also have a method called `getTopScoresFromLevelSorted(levelName)` that will return a list with the top three highscores for the specific `levelName`, sorted with the smallest first. The `LevelChooserListAdapter` will display all the items in the list and will be shown in the LevelChooser 3 under ”Best Time”. The database itself is implemented using a singleton pattern since it is costly to build each time we would like to access the database [3].

4.5 Planned Features

4.5.1 Google+ Friends

We planned to implement more social features like being able to compare your scores and rankings with your friends. The leaderboards do have a social switch you can turn on, so you would expect that Google Play Games Services used that feature to show only your Google+ friends instead of showing all registered high scores globally.

But adding Google+ friends via the Google Play Games app is not possible anymore. We tried following each other on Google+, but there is no Friends or Players list available in the app anymore. If you do some Google searches you will find some tutorials and videos showing you how to add friends via the app, but they are old versions of the Games app.

It seems that Google has dropped Google+ integration with the Games Service all together [4] [5]. Google still has multiplayer support in Android games [6], but no support for adding friends and other social aspects in the Games Service.

5 Experiences

5.1 Google Play Games Services

Google Play and Google Play Games services are extremely nice features provided by Google, but at the same time they are very poorly documented. By poorly, we don't mean that there is a lack of documentation, because there is a lot. We mean there are so many, and all of them only mention parts of what you need to know, so you need to go through all of them to get it right.

First of all, we started implementing sign-in using GoogleApiClient, and then when we followed other documentation they used a completely other type of sign-in flow. So it seems that GoogleApiClient is now deprecated, so we had to change everything to use the new Sign-in flow instead as recommended. The problem is that most tutorials and documentation still reference the deprecated way of signing in, so they become completely useless during troubleshooting.

Then you need to learn the differences between Google Play and Google Play Games, even though the are two completely different services, they are also very tightly integrated as the Games service uses the Play service for all authentication processes.

After we finally got everything to work in one developer PC, we realized during the Pull Request that no one else on the team could get it to work. That's when we learned that if you want to use Google Sign-In for game features, you need to register the game or project on Google Play Console, and to do that, you need a paid developer account.

And if you want other people on your team to test it, they need to be added as testers using their Google accounts. Then you need to link the SHA-1 key on their machine which is used to sign the debug APK, to get an oAuth2 client link between the app (APK) and the project (package name). You need to do the linking in two different places in Google Play Console, one in the project and the second place is in Google Cloud (APIs & Services).

None of this is clearly defined in any one of the documentations, they are all over the place and very inconsistent, so you need to glue small pieces of information from many places together and make some implicit assumptions. Also console errors and activity results will not help you at all, because if all the above mentioned requirements are not set, you will just get generic errors saying the activity was cancelled, or that the user account was not properly authenticated even though they are.

5.2 Level Creator Old API Bug

We encountered an unusual bug with the level creator affecting older APIs. The levels would be built correctly and rendered correctly in nougat, but in lollipop the player was faced with a blank screen. Android studio gave no errors so we had no obvious clues. After looking at the android documentation for `RectF` [7] we saw the note: "*Note: most methods do not check to see that the coordinates are sorted correctly (i.e. left ¡= right and top ¡= bottom)*". Even

the documentation for the constructor states that it does not perform range checks. We did after some time actually check if we were calling the constructor with the parameters in the wrong order, which we were. After fixing that the level was built and rendered correctly across all API versions. It seems like they made the constructor more permissive in nougat without documenting it.

References

[1] e.James. Circle-rectangle collision detection (intersection). https://stackoverflow.com/questions/401847/circle-rectangle-collision-detection-intersection, 2008. (Visited 2018-04-13).

[2] Google. Room persistence library. https://developer.android.com/topic/libraries/architecture/room, April 25, 2018. (Visited 2018-05-04).

[3] Ajay. Building database with room persistence library. https://medium.com/@ajaysaini.official/building-database-with-room-persistence-library-ecf7d0b8f3e9, May 22, 2017. (Visited 2018-05-04).

[4] Mishaal Rahman. Google+ integration will finally be removed from google play games in 2017. https://www.xda-developers.com/google-integration-will-be-removed-from-google-play-games-in-2017/, 2016. (Visited 2018-05-01).

[5] Clayton Wilkinson. Games authentication adopting google sign-in api. https://android-developers.googleblog.com/2016/12/games-authentication-adopting-google.html, 2016. (Visited 2018-05-01).

[6] Google. Real-time multiplayer support in android games. https://developers.google.com/games/services/android/realtimeMultiplayer, 2017. (Visited 2018-05-01).

[7] Google. Rectf. https://developer.android.com/reference/android/graphics/RectF, 2018. (Visited 2018-05-04).