



Deep linking in mobile Unity game

Jeremias Kuitunen

Bachelor's thesis

December 2017

School of Technology, Communication and Transport

Degree Programme in Media Engineering

Author(s) Kuitunen, Jeremias	Type of publication Bachelor's thesis	Date 10.12.2017 Language of publication: English
	Number of pages 51	Permission for web publication: x
Title of publication Deep linking in mobile Unity game		
Degree programme Media Engineering		
Supervisor(s) Manninen, Pasi		
Assigned by Traplight Ltd.		
Abstract <p>The thesis was assigned by Traplight Ltd., a mobile game development company that focuses on creating high quality user-generated content games (UGC). Traplight strives for being the industry leader in free mobile games where users generate the game content.</p> <p>The goal of the thesis was to develop a deep linking solution and deploy it in a game called Big Bang Racing prior to its launch in July 2016. In Big Bang Racing, deep linking capabilities allow users to share user-created levels with web URLs outside the game boundaries including social media platforms. Opening a deep link takes the user to the shared content or to a platform specific app store page in case the application was not preinstalled.</p> <p>A deep linking plugin was created for Unity game engine, and it supports iOS and Android platforms. The Unity plugin consists of procedures that enable communication between platform native code and Unity code. This allows presenting deep linked content in the game and sharing of deep links with system native sharing dialog. Providing deep link metadata for web crawlers allows rich and user friendly posts to social media channels.</p> <p>The result was a working deep linking solution still in use with Big Bang Racing. The solution offers basic deep linking functionality with a good foundation to be further developed to fit the requirements of a modern deep linked application.</p>		
Keywords/tags (subjects) User-generated content, deep link, mobile game, plugin		
Miscellaneous		

Tekijä(t) Kuitunen, Jeremias	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä 10.12.2017
	Sivumäärä 51	Julkaisun kieli Englanti
		Verkojulkaisulupa myönnetty: x
Työn nimi Syvälinkitys Unity-mobiilipelissä		
Tutkinto-ohjelma Mediatekniikka		
Työn ohjaaja(t) Pasi Manninen		
Toimeksiantaja(t) Traplight Oy		
Tiivistelmä <p>Mobiilipeliyhtiö Traplight keskittyy korkealaatuisten pelien tekemiseen. Traplight pyrkii olemaan edelläkävijä ilmaisissa mobiilipeleissä, joissa käyttäjät luovat pelin sisällön.</p> <p>Opinnäytetyön tavoitteena oli luoda ratkaisu, joka mahdollistaa syvälinkityksen pelin sisältöön. Syvälinkitysominaisuudet tuli olla Big Bang Racing -nimisessä pelissä ennen sen maailmanlaajuista heinäkuun 2016 julkaisua. Big Bang Racingissa syvälinkitys antaa pelaajille mahdollisuuden jakaa käyttäjien luomia kenttiä pelin rajojen ulkopuolelle kuten sosiaaliseen mediaan. Kenttien jakaminen tapahtuu käyttäen web-osoitteita. Syvälinkin avaaminen vie käyttäjän pelin sisälle jaettuun kenttään. Jos peliä ei ole valmiiksi asennettu, vie syvälinkin avaaminen käyttäjän pelin alustakohtaiselle kauppasivulle.</p> <p>Syvälinkkiiliitännäinen tehtiin Unity-pelimoottorille tukien Android ja iOS -alustoja. Liitännäinen koostuu menetelmistä, jotka mahdollistavat kommunikoinnin alustojen natiivin ja Unity-koodin välillä. Tämän avulla voidaan käyttäjille tarjota syvälinkitetty sisältö, kun syvälinkkiä on painettu. Liitännäinen mahdollistaa myös linkkien jakamisen käyttäen alustakohtaista jakodialogia. Syvälinkkien sisällöstä palvelin tarjoaa hakuroboteille metatietoja, joiden avulla sosiaalisen median alustat pystyvät luomaan syvälinkistä havainnollisen julkaisun.</p> <p>Opinnäytetyön tuloksena saatiin toimiva syvälinkityskokonaisuus, joka on edelleen käytössä Big Bang Racing -pelissä. Toiminnallisuudeltaan ratkaisu tarjoaa perustavanlaatuisen pohjan syvälinkitysominaisuuksien hyödyntämiseen ja jatkokehittämiseen vastaavissa tuotteissa.</p>		
Avainsanat (asiasanat) Käyttäjien luoma sisältö, syvälinkki, mobiilipeli, liitännäinen		
Muut tiedot		

Contents

1	Introduction.....	4
1.1	Background.....	4
1.2	Traplight Ltd.	4
1.3	Objectives	5
2	Big Bang Racing.....	5
2.1	Gameplay.....	5
2.2	User-generated content in Big Bang Racing	6
3	Mobile deep linking	8
3.1	What is mobile deep linking?	8
3.2	Deep linking features in a user-generated content game	10
3.3	The deep link solution for Big Bang Racing.....	11
4	Implementing deep linking in a mobile game	12
4.1	Client development	12
4.1.1	Structure of the DeepLink plugin	12
4.1.2	Launching the application from a deep link.....	13
4.1.3	Passing the URI to Unity	16
4.1.4	Presenting the deep linked content in Unity	19
4.1.5	Sharing a deep link.....	20
4.2	Server development	25
4.2.1	Why to have a deep link landing page?	25
4.2.2	Deep link landing page.....	27
4.2.3	Rich social media posts	32
4.3	Facebook’s App Links for Android	35
4.4	GIF sharing with deep linking	36

5	Results and discussion	40
5.1	Deep linking in Big Bang Racing.....	40
5.2	Further improvements for DeepLink solution	43
5.3	Development of Unity plugins.....	47
5.4	Other deep linking solutions for Unity in 2017.....	47
5.5	Current state of mobile deep linking.....	48
	References	50

Figures

Figure 1.	Racing against others in Big Bang Racing	6
Figure 2.	Level editor in Big Bang Racing.....	7
Figure 3.	Loading screen while loading a level from server	8
Figure 4.	Flow of opening a deep link	9
Figure 5.	iOS and Android specific sharing buttons on Imgur application	9
Figure 6.	Reasons mobile gamers share gameplay experiences (Mobile Gaming: Social Motivations 2014)	10
Figure 7.	DeepLink -plugin files in a Unity file hierarchy	12
Figure 8.	Opening a deep link with the game.....	13
Figure 9.	Xcode entitlements	14
Figure 10.	Visualization of how deep link plugin passes an URI to Unity	16
Figure 11.	Native Android sharing dialog for text	22
Figure 12.	Native iOS text sharing dialog	23
Figure 13.	A view after publishing a level.....	24
Figure 14.	Level cards on iOS and Android.....	24
Figure 15.	Deep link is opened in a webview and fails.....	25
Figure 16.	Deep link experience with a landing page through a webview.....	26
Figure 17.	Deep link flow with a landing page	27

Figure 18. Deep link landing page for a level	28
Figure 19. Deep link logic flow from server's perspective.....	31
Figure 20. Facebook constructed rich snippet of a post containing a deep link	33
Figure 21. Twitter's generated <i>Summary card with large image</i>	34
Figure 22. A post containing a deep link on different social media platforms.....	35
Figure 23. Preview of GIF	37
Figure 24. GIF deep link posted to Facebook	39
Figure 25. Chart of deep link features used by users	40
Figure 26. Sharing of levels by player type and users sharing own levels.....	41
Figure 27. Comparison of deep links opening the application versus total share events (GIFs and levels)	41
Figure 28. Deferred deep linking.....	44
Figure 29. More personalized deep link sharing	46

1 Introduction

1.1 Background

The consumption of online content has changed a great deal in the past decades. The amount and variety of online content has increased greatly with not only traditional services, such as newspapers, Netflix and Kela, moving to web but also with new web-only services being born, e.g. different social media platforms.

The popularity of smartphones keeps increasing and nowadays more than half of the world's population uses a smartphone. Of all web traffic, the majority comes from mobile phones. (Kemp 2017.)

Mobile web browsers have gotten better in the sense of taking advantage of native mobile features more widely, such as 3D accelerated graphics and location services. However, according to eMarketer (2016), the most of mobile device usage comes from apps due to the fact that apps are seen to provide more immersive experience with functionalities taking better advantage of the native features of the platform, such as push message capability and access to contact list.

The traditional way of sharing content online is by sharing a link by copy-pasting a web URL from a web browser's address bar to the recipient. With the rise of mobile apps, and since mobile apps do not have address bars apart from web browser apps, the ability to link content has required new practices to enable sharing of online content.

1.2 Traplight Ltd.

Traplight Ltd., a Tampere-based game development studio in Finland, focuses on creating high quality free to play user-generated content (UGC) games for mobile platforms. Traplight employs 25 people. (Traplight n.d.)

The author has worked at Traplight as a programmer since May 2015. Currently Traplight develops its titles with Unity game engine. Unity is the one of the most popular public game engines with its easy-to-use multiplatform developing tools alongside with a big community of game developers and a great deal of third party content, e.g. plugins and other assets (Murphy n.d.).

1.3 Objectives

The goal of the thesis was to deploy deep linking capabilities in a Unity game that works on both iOS and Android platforms. The thesis consists of following objectives:

- Choosing the most suitable way of having deep linking features in a Unity game for iOS and Android.
- Implementing the features in Big Bang Racing game prior to its scheduled global release date of July 2016.
- Deep linking features:
 - Game launches from a deep link and presents the associated content to the user.
 - Users of Big Bang Racing are able to share deep links to others easily.
 - The shared GIFs have a deep linking functionality.
- Analysis of having the deep linking features in an UGC game.

2 Big Bang Racing

2.1 Gameplay

Big Bang Racing (BBR) is a 2D physics based racing game where the player races against other players around the world (See Figure 1). The core gameplay consists of driving a vehicle by accelerating, breaking and leaning to reach the goal. Players compete to gain trophies to advance into higher leagues and to climb up the leaderboards.



Figure 1. Racing against others in Big Bang Racing

Aside from racing, players in Big Bang Racing can play adventure levels. Adventure mode is more casual and relaxed compared to racing. In the adventure levels the goal is to survive through the level and collect map pieces.

2.2 User-generated content in Big Bang Racing

UGC is the core of Big Bang Racing as every level is user-made, which means that the game's levels are as unique as the different players creating those. Basically, if the game has players, the amount of playable content increases. The game has a powerful level editor allowing players to create diverse, exciting and artistic levels. Levels are created by drawing shapes with different ground materials, placing physical objects and decorating with decorative items as illustrated in Figure 2.



Figure 2. Level editor in Big Bang Racing

When a player creates a level and publishes it, the level is sent to the server where it is stored in a database to be served for other players. Players have the possibility to play any live level by using a search tool in the game or by entering a level through its creator's profile page. However, the user progression is tied to playing levels that are being distributed by the server for the player. The distribution algorithm works based on parameters such as quality, difficulty, newness, player's vehicle statistics and possible previous experience of the player on this level. The quality is determined by other players' rating of it.

When a player chooses to play a level, a loading screen is presented while the client downloads the level from the server. The loading screen contains information about the level including the creator's name, the creator's profile picture, name of the level and a screenshot of the level as seen in Figure 3.

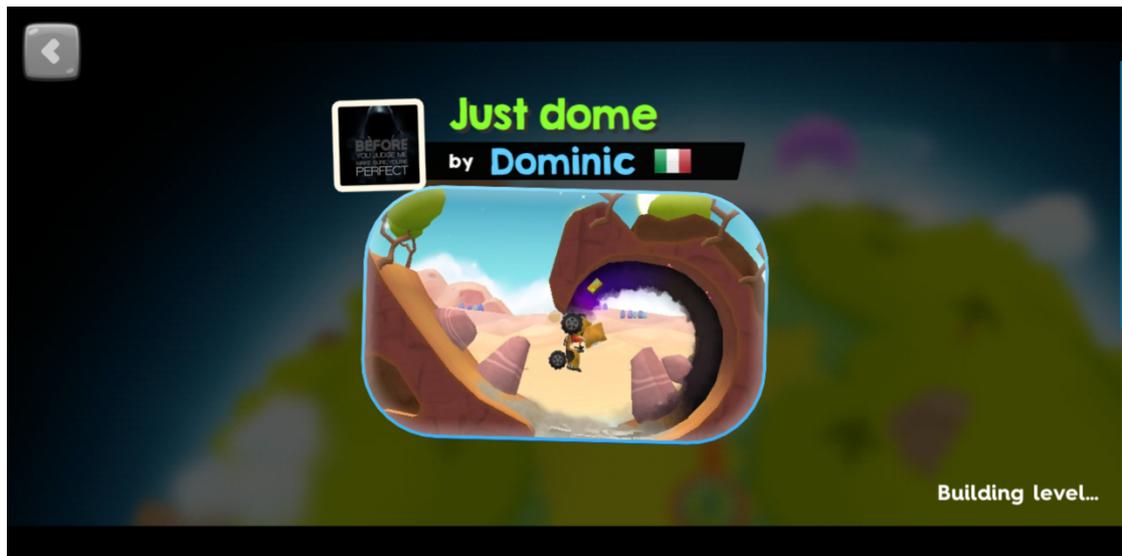


Figure 3. Loading screen while loading a level from server

3 Mobile deep linking

3.1 What is mobile deep linking?

Deep link in the context of the web is a hyperlink linking to a specific content instead of a homepage of a website. The term mobile deep linking stands for the ability to link specific content of a mobile app with the use of URIs, which allows users to share content outside the traditional boundaries of the application. Tapping a deep link on a mobile device will open the application in question and show the linked content to the user. If the application was not already installed on the system, app store page will be presented to the user as illustrated in Figure 4.

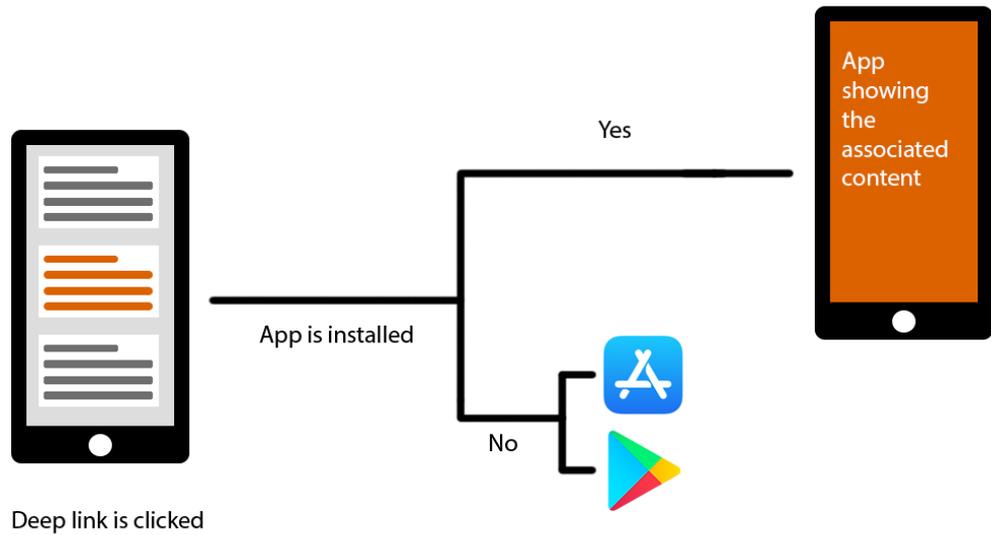


Figure 4. Flow of opening a deep link

Deep linked content is shared by pressing a distinct sharing button, as seen with Imgur application in Figure 5. Pressing the button gives the user a link to the content chosen to be shared. The link can be shared to any appropriate application installed on the device using a platform native share dialog.

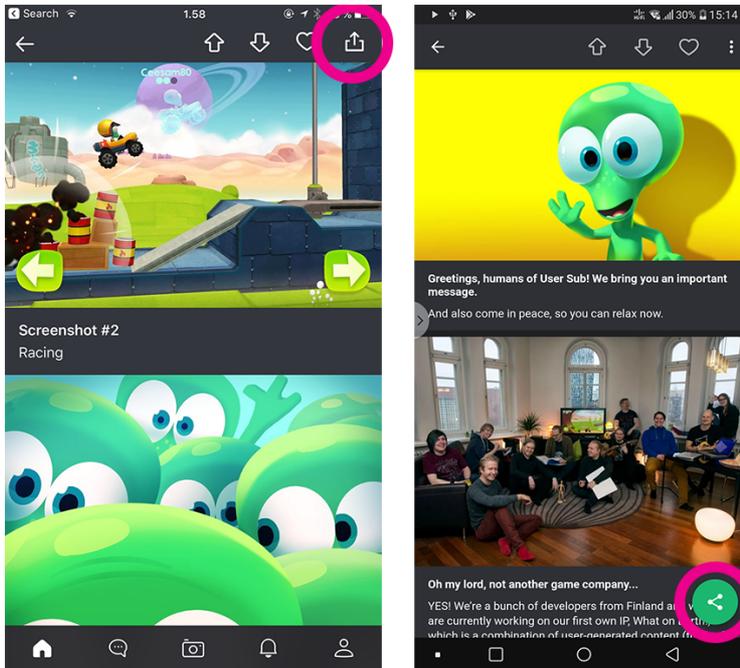


Figure 5. iOS and Android specific sharing buttons on Imgur application

3.2 Deep linking features in a user-generated content game

Players, as any other group of people, enjoy talking about their hobbies. In mobile games, this can translate into players sharing gameplay experiences online. The reasons for sharing in mobile games can be further broken down to understanding the motivation behind the behaviour (Figure 6).

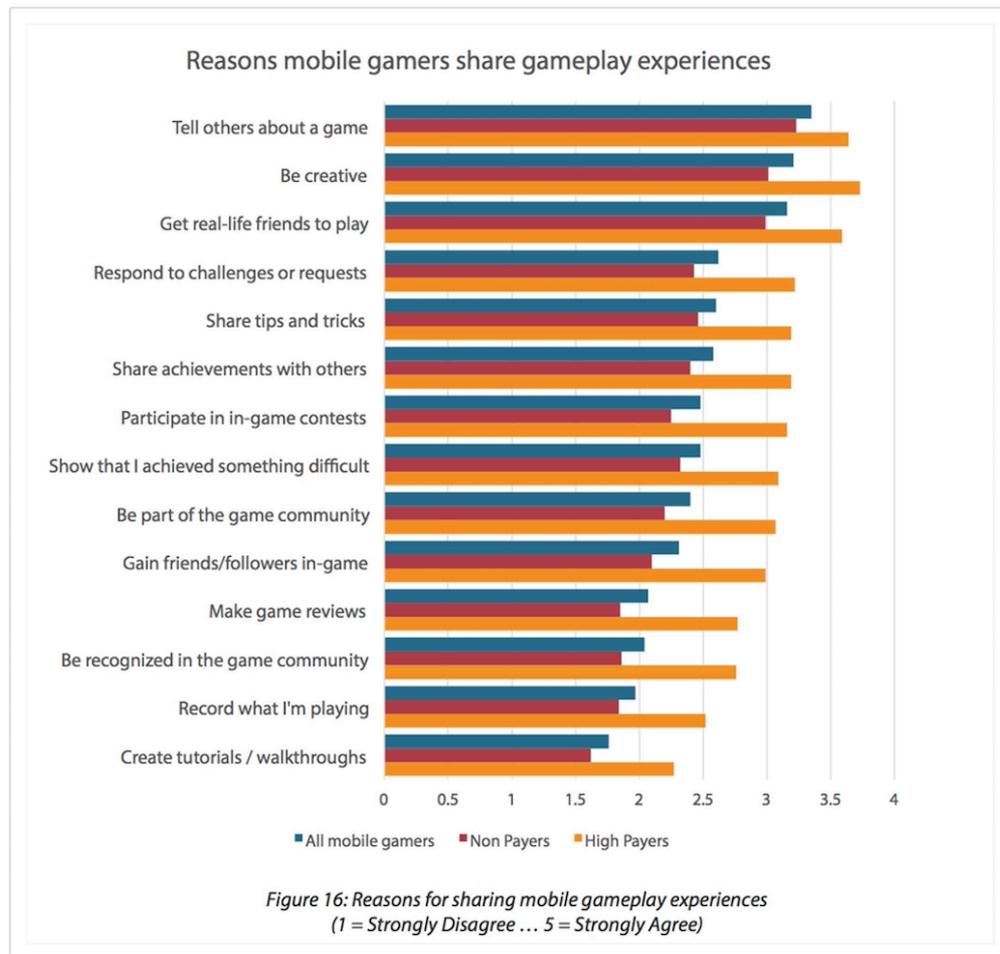


Figure 6. Reasons mobile gamers share gameplay experiences (Mobile Gaming: Social Motivations 2014)

Many mobile games offer the option to share a screenshot or a high score to social media channels. In user-generated content games such as Big Bang Racing, creating and sharing content is a core part of the gaming experience. Deep linking not only provides a natural way for the players to share experiences outside of the game but

also makes sharing meaningful by allowing others to interact with the deep linked content.

The motive for having deep linking features in a user-generated content game can be seen from two different aspects, player's and the product's point of view. Deep linking features can be used with the product's marketing strategy. According to Mavrck (2017), user-generated content drove over six times higher engagement than brand-generated posts on Facebook. Promoting game's user-generated content can also feel less intrusive than traditional advertisements. When players share links, they are not only introducing the game to potential users but also make the game gain more media coverage and making it more viral. Virality is important since it helps getting youtubers and other social media influencers interested in the game.

Players sharing deep links makes them more engaged with the game by providing a convenient way of exploring the game's content with others. The ability to share the content can also help building player communities around the game.

3.3 The deep link solution for Big Bang Racing

When deciding whether to use an existing solution or developing one from the ground up, different factors needs to be assessed. The aspects include the required setup workload for a new project, total development time, feature set, user experience, control over the features, compatibility with other plugins and cost of use.

When developing a solution from the ground up, the initial development time will naturally be longer.

At the time the work of bringing a deep linking solution to the game began, no complete deep linking solutions were available for Unity. Researched existing deep linking client plugins were inadequate as they only supported one platform. Developing a solution from ground up was easily arguable as it was the only reasonable way to achieve unified deep linking features across both platforms.

4 Implementing deep linking in a mobile game

4.1 Client development

4.1.1 Structure of the DeepLink plugin

Client development consists of developing DeepLink plugin for Unity and deploying deep linking features to Big Bang Racing with the plugin. The plugin is responsible for providing the context of the opened deep link to the Unity game and presenting the platform specific sharing dialog for sharing deep links. Client development is partly divided for each mobile platform.

The plugin consists of iOS and Android specific files as well as a C# class that the developer uses to handle sharing of deep links and getting the URL that launched the game as shown in Figure 7. The existing folders are a part of a Unity project template. The new files represent the files that are part of the DeepLink plugin. Modified file represents that the file is part of a Unity project template, however, it needs to be modified to enable deep linking features.

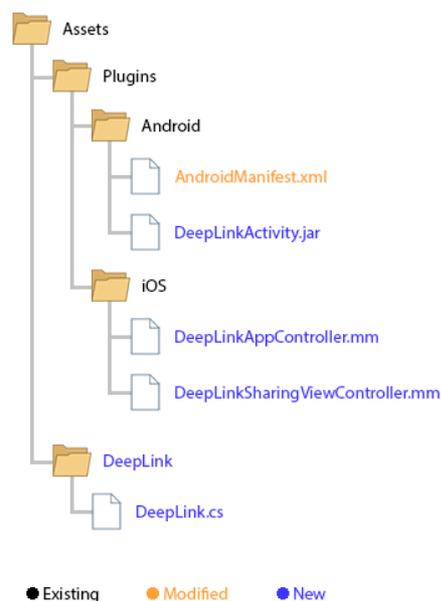


Figure 7. DeepLink -plugin files in a Unity file hierarchy

4.1.2 Launching the application from a deep link

Opening a deep link initiates a sequence of required steps to present the associated content to the user (Figure 8). Steps 1 and 4 are app specific and need to be done to each project individually. Steps 2 and 3 are plugin specific and provide plug-n-play functionality when integrated into new projects.

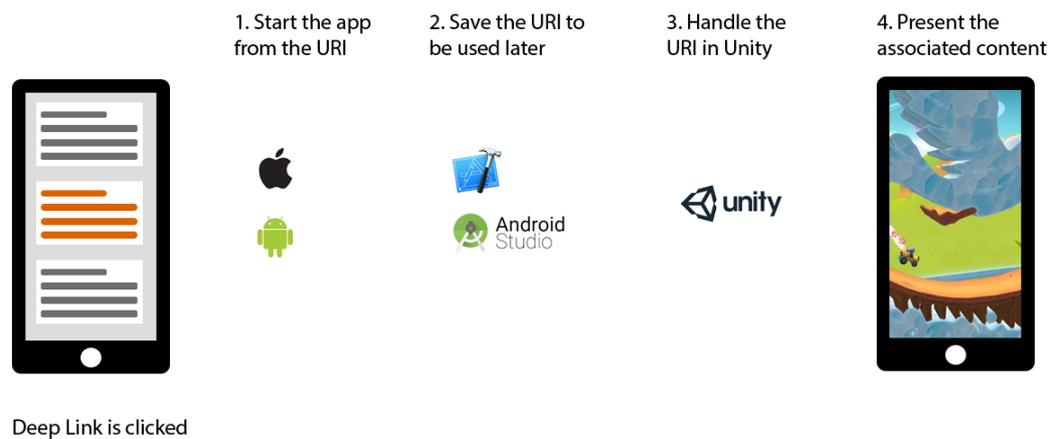


Figure 8. Opening a deep link with the game

iOS and Android have their own setup steps for the operating system to be able to launch the application when the user taps a predefined type of URI. Both platforms support launching an application from a custom URI scheme. Below is an example of a link using custom bigbangracing URI scheme:

```
bigbangracing://playlevel/588d12123000005b001b93af
```

The path `playlevel/588d12123000005b001b93af` has the information that the application will use in order to present the content to the user, the `playlevel` being the action and `588d12123000005b001b93af` being the target. If an application has registered an URI scheme to the operating system, it can be launched by opening a link with the associated URI scheme. On iOS and Android, the registering takes place when the application is installed. Other popular URI schemes include HTTP, HTTPS, mailto, SMS, FTP and SSH. URI scheme is simple to setup and good for prototyping, however, since the URI scheme is customized, the operating systems are not able to

handle it when the application is not preinstalled. When the user opens a custom URI that is not recognized by the operating system, it will present an error to the user.

Using a web URL to launch the application requires more steps to setup on iOS, however it allows having control what is presented to the user if the operating system is not supported or the application is not preinstalled. The example below is a web URL containing the same information for Big Bang Racing to handle as the preceding custom URI scheme example. The difference is that the operating system will load the URL in a web browser rather than prompting the user with error if the application was not installed; which allows redirecting the user to a platform specific App Store of the game through the web site.

`https://playbbr.traplightgames.com/playlevel/588d12123000005b001b93af`

With iOS 9, Apple introduced a feature called Universal Links which allows launching an application from a web URL (iOS 9 n.d.). Prior to iOS 9, applications could only be launched with custom URI schemes. iOS 9 or newer iOS versions are supported by Apple mobile devices from 2011 onward (iOS Support Matrix n.d.). Having full deep link support for older iOS versions was not relevant and was dropped to save development time.

Setting up Universal Links requires three steps in order to work. From Apple Developer web portal, Associated Domains setting needs to be enabled for the AppID. In Xcode project, Associated Domains entitlement must be enabled under the Capabilities tab and an entry containing the web URL must be included as seen in Figure 9. (Support Universal Links n.d.)

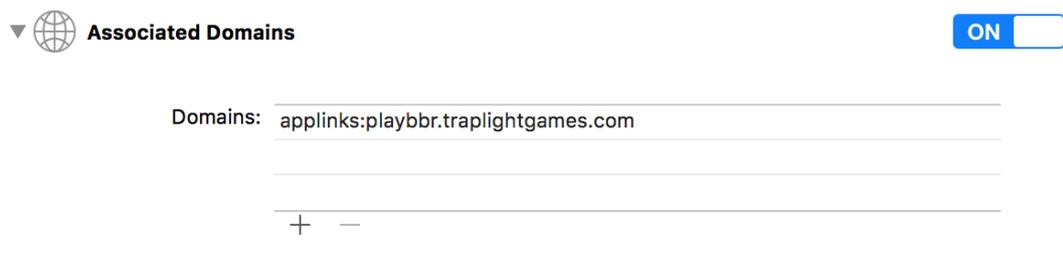


Figure 9. Xcode entitlements

A file named apple-app-site-association needs to be accessible from the root of the domain that can launch the application. The content of the file is in JSON format and contains the information of the application as well as the paths which will launch the application. (Support Universal Links n.d.)

```
{
  "applinks": {
    "apps": [],
    "details": [{
      "appID": "TEAMD_ID.APP_ID ",
      "paths": ["*"]
    }]
  }
}
```

When an application with associated domains is installed, the operating system fetches the JSON formatted file and is able to launch the app when the predefined conditions are met when clicking a link.

On Android, the AndroidManifest.xml file must include the information of the URIs launching the application. The data belongs inside the activity it needs to launch, which in this case is the UnityPlayerActivity, the main activity of the application.

```
<activity android:name="com.unity3d.player.UnityPlayerActivity"
  android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="http" />
    <data android:scheme="https" />
    <data android:host="playbbr.traplightgames.com" />
    <data android:pathPattern=".*" />
  </intent-filter>
  <!-- other main activity related data -->
</activity>
```

The tags inform the operating system that it can display the data referenced by a link and specify the kind of links it is able to handle.

4.1.3 Passing the URI to Unity

In order for the game to present the content in question to the user, the URI used to launch the application must be accessible within the Unity code. There are two possible ways for passing data to Unity from the native code of the platform: either saving the data to a location that Unity can access or calling Unity methods from the native code with the data as a parameter.

Calling Unity method from native code has a delay and requires a gameobject with a script in the scene receiving the method call. PlayerPrefs is a place to save data on Unity using key-value pairs. Saving the data from platform's native code to Unity's PlayerPrefs makes accessing the data effortless and therefore is the chosen way to pass the information from native code to Unity code (Figure 10). Internally Unity uses NSUserDefaults class for PlayerPrefs on iOS and on Android the equivalent class is SharedPreferences. Using these classes from plugin's native code makes it possible to store the deep link URI and access it from Unity's side with ease.

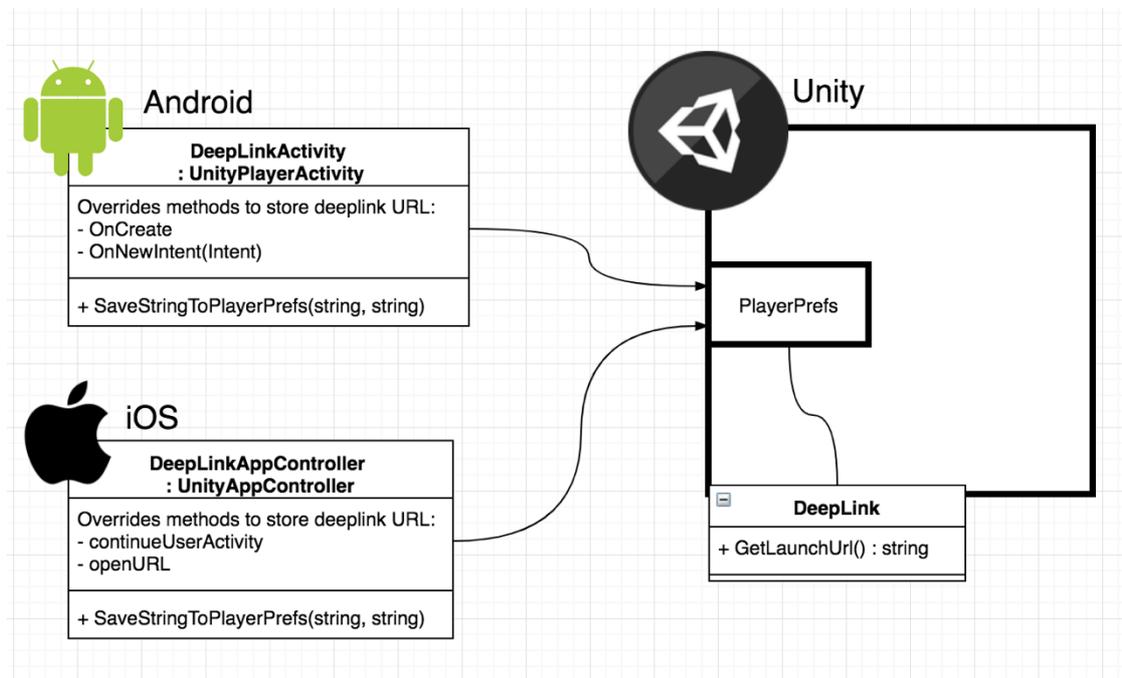


Figure 10. Visualization of how deep link plugin passes an URI to Unity

On iOS, the native code is Objective-C. UnityAppController class must be extended with a custom AppController to get the URI that started the applications activity. To extend the default AppController, the extending class must have the following line:

```
IMPL_APP_CONTROLLER_SUBCLASS(DeepLinkAppController)
```

In this class, it is required to override the method that is being called by the operating system when launching or continuing the application from a Universal Link.

```
- (BOOL)application:(UIApplication *)application
continueUserActivity:(NSUserActivity *)userActivity
restorationHandler:(void (^)(NSArray
*restorableObjects))restorationHandler
{
    if ([userActivity.activityType
        isEqualToString:NSUserActivityTypeBrowsingWeb])
    {
        NSString *URLString = [userActivity.webpageURL absoluteString];
        [self SaveStringToPlayerPrefs:URLString key:deepLinkUr1Key];
    }
    return YES;
}
```

The method checks if the activity has been activated from a web URL and saves the URL by calling a method SaveStringToPlayerPrefs with the URL as a parameter. This method takes another parameter which is the key for the key-value pair. Accessing Unity's PlayerPrefs is possible within native iOS code with UserDefaults class:

```
-(void)SaveStringToPlayerPrefs:(NSString *) URLString key:(NSString
*)key
{
    NSLog(@"-> BBR Saving url to playerprefs: %@", URLString);
    [[NSUserDefaults standardUserDefaults] setObject:URLString
    forKey:key];
    [[NSUserDefaults standardUserDefaults] synchronize];
}
```

To make Unity include the extending AppController in Xcode project automatically when building for iOS, the file (DeepLinkAppController.mm) has to be located in /Assets/Plugins/iOS/ folder inside the Unity project (Figure 7).

On Android, the native code is Java. To get access to the web URL that started the activity, UnityPlayerActivity class needs to be extended with the methods that are being called when starting the activity or continuing the activity. When Android

launches an application that was not already running in the background, it calls onCreate method:

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    Intent intent =
    com.unity3d.player.UnityPlayer.currentActivity.getIntent();
    if(intent != null && intent.getAction() == Intent.ACTION_VIEW)
    {
        SaveUrl(intent);
    }
}
```

In the override onCreate method, the activity's intent is passed to SaveUrl to store the URI. Also to store the URL when opening a link that continues the application from background, onNewIntent(Intent intent) needs to be overridden to call SaveUrl-method.

```
private void SaveUrl(Intent intent)
{
    Uri uri = intent.getData();
    if(uri != null)
        SaveStringToPlayerPrefs(uri.toString(), DEEP_LINK_URL_KEY);
}
```

SaveStringToPlayerPrefs-method accesses the same file that Unity uses internally for PlayerPrefs on Android and stores the key-value pair there for easy usage from Unity's side.

```
public static void SaveStringToPlayerPrefs(String value, String key)
{
    Context appContext =
    com.unity3d.player.UnityPlayer.currentActivity.getApplicationContext();
    SharedPreferences prefs =
    appContext.getSharedPreferences(appContext.getPackageName() +
    ".v2.playerprefs", Context.MODE_PRIVATE);
    Editor prefEditor = prefs.edit();
    prefEditor.putString(key, value);
    prefEditor.commit();
}
```

To compile a Java class that has references to or extends Unity classes, Unity's classes.jar needs to be added to the classpath that is used to compile the library. Unity's

classes.jar can be found inside Unity's installation folder Unity/PlaybackEngines/AndroidPlayer/Variations/mono/Release/Classes/. The compiled Android deep link library is then copied to Unity project's Android plugins folder (see Figure 7).

When extending the UnityPlayerActivity with another class, the AndroidManifest.xml needs to be modified to reflect that change. The main activity of the application, which is also launchable from web URLs, is now the newly created DeepLinkActivity:

```
<activity android:name=".DeepLinkActivity"
    android:label="@string/app_name">
```

On Unity, the programming language is C#. The URL that launched the application can be read on Unity's side by calling DeepLink.GetLaunchUrl method.

```
static public string GetLaunchUrl()
{
    string url = PlayerPrefs.GetString(LAUNCH_URL);
    PlayerPrefs.SetString(LAUNCH_URL, "");
    return url;
}
```

Returned empty value means that the application was not activated by a deep link. After the URL has been read from PlayerPrefs, it is set to be empty and therefore the same URL will not be used more than once. Both native and Unity code are set to use the same key in the key-value pair in storing and accessing the deep link URL.

Having ability to store data in PlayerPrefs from platform's native code with a key-value pair (method SaveStringToPlayerPrefs, see Figure 10) enables an effortless way of passing data from native code to Unity code, which can be utilized to be used with possible future solutions.

4.1.4 Presenting the deep linked content in Unity

Presenting deep linked content is project specific and is not a part of the deep link plugin. In a basic Unity project, there are two cases when checking for a deep link URL should take place:

- After initialization of the game to see if it was launched from a deep link
- In a script that inherits from MonoBehaviour and implements a method OnApplicationPause(bool) to handle if the game was already running and activated with a deep link

In an online game such as Big Bang Racing, the conventional place to check for a deep link is after server authentication since that is made to occur every time the application is launched or returned from background.

After retrieving the URI, it needs to be parsed in order to get the relevant data. In Big Bang Racing, the deep link that is from a level looks like this:

```
https://playbbr.traplightgames.com/playlevel/588d12123000005b001b93af
```

The content after the host `playbbr.traplightgames.com` is the path. The first part of the path `/playlevel/` is the action which tells what to do. The latter part of the path is the unique identifier of the shared level. That identifier is then used to get the information about the level from the server to be presented to the user.

It is worth taking all the possible states of the game into consideration when the user opens a deep link so it can be handled without losing of unsaved progress. In Big Bang Racing, handling of different states includes:

- Main menu -> User is taken to the content
- Inside a level -> Prompt is showed to the player to confirm the transition to the content
- Creating a level -> Prompt is showed to the player to confirm the transition to the opened content.
- Loading level scene -> Prompt is showed the after loading has been finished
- Loading menu scene -> User is taken to the shared content after the menu has loaded

4.1.5 Sharing a deep link

The shareable content can be anything with a unique identifier. Depending on the application, it can be user-made content, in-game events, high scores, or anything in between. In Big Bang Racing, users are able to share levels.

On Android, it is possible to share data by instantiating an Intent, setting the shareable to it and starting an activity with that intent. Deep links are shared by sharing a text containing the link:

```
Intent deepLinkIntent = new Intent();
deepLinkIntent.setAction(Intent.ACTION_SEND);
```

```

deepLinkIntent.putExtra(Intent.EXTRA_TEXT, _deepLinkUrl);
deepLinkIntent.setType("text/plain");
currentActivity.startActivity(deepLinkIntent);

```

Even though the shared deep link content can be practically anything from the game, the actual sharing always contains the deep link URL. In other words, instead of sharing an actual screenshot file as a binary, the sharing occurs by sharing a deep link URL leading to that particular screenshot. Since it is not possible to have plain Java code in Unity, it is required to use `AndroidJavaObject` and `AndroidJavaClass` in order to call native Android methods from Unity's C# (`AndroidJavaObject` n.d.; `AndroidJavaClass` n.d.). The C# example below has the same functionality as the preceding Java example.

```

private static void ShareTextOnAndroid(string _deepLinkUrl)
{
    AndroidJavaClass intentClass = new
    AndroidJavaClass("android.content.Intent");
    AndroidJavaObject deepLinkIntent = new
    AndroidJavaObject("android.content.Intent");

    deepLinkIntent.Call<AndroidJavaObject>("setAction",
    intentClass.GetStatic<string>("ACTION_SEND"));

    deepLinkIntent.Call<AndroidJavaObject>("putExtra",
    intentClass.GetStatic<string>("EXTRA_TEXT"), _deepLinkUrl);

    deepLinkIntent.Call<AndroidJavaObject>("setType", "text/plain");

    AndroidJavaClass unity = new AndroidJavaClass
    ("com.unity3d.player.UnityPlayer");

    AndroidJavaObject currentActivity =
    unity.GetStatic<AndroidJavaObject>("currentActivity");

    currentActivity.Call ("startActivity", deepLinkIntent);
}

```

Calling `ShareTextOnAndroid` method will bring up the Android's native sharing dialog, where the user has the option to share the link with another app or copy it to clipboard as shown in Figure 11.

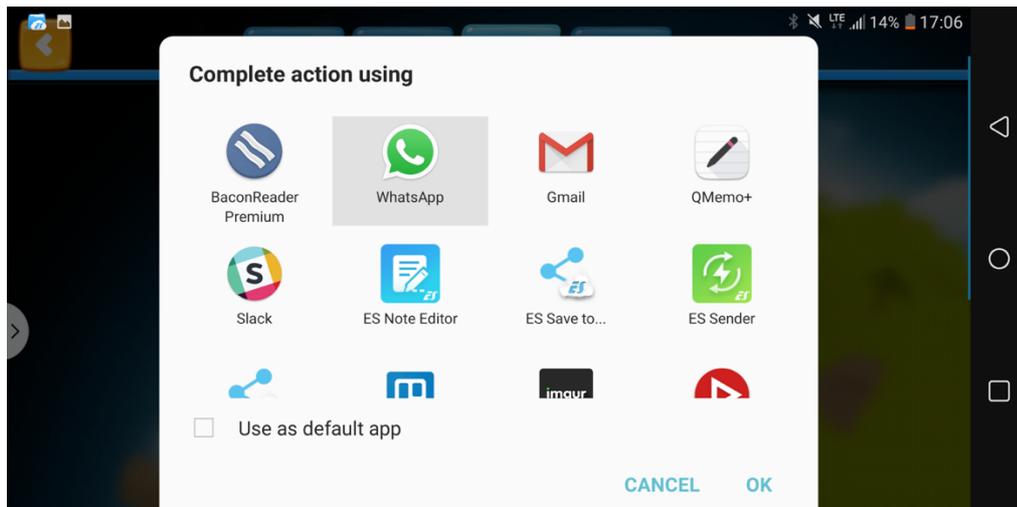


Figure 11. Native Android sharing dialog for text

On iOS, text sharing is developed with the guidance of The AppGuruz blog post (Khira 2015) by having a native iOS class with a globally declared text sharing method. The Objective-C class file `SharingViewController.mm` is placed into iOS plugins folder and the text sharing method `ShareTextOnIOS` is imported in C#:

```
#if UNITY_IPHONE && !UNITY_EDITOR
[DllImport("__Internal")]
private static extern void ShareTextOnIOS (string _sharedText);
#endif
```

Calling this method from C# presents the user with an iOS native sharing dialog where the user has the option to share the link to different applications or to copy it to the clipboard (Figure 12).

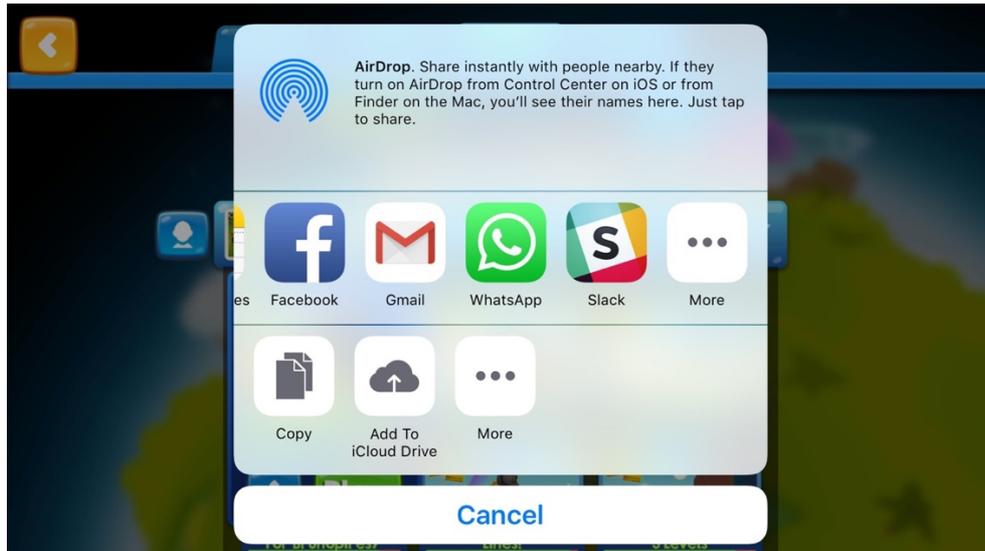


Figure 12. Native iOS text sharing dialog

In the C# DeepLink class in Unity, the platform specific code and methods are separated with preprocessor directives so that the project compiles for each platform:

```
static public void ShareTextOnPlatform(string _text)
{
    #if UNITY_IPHONE && !UNITY_EDITOR
    ShareTextOnIOS(_text);
    #elif UNITY_ANDROID && !UNITY_EDITOR
    ShareTextOnAndroid(_text);
    #else
    Debug.Log("Text sharing not possible on this platform: " + _text);
    #endif
}
```

A deep link from the game can be shared by calling `DeepLink.ShareTextOnPlatform(URLstring)` regardless of the platform, and the platform specific text sharing dialog will be presented to the user.

In Big Bang Racing, there are multiple occasions where users have the option to share a level. After publishing the level, the player is encouraged to share the level to others (Figure 13). The search results of the levels are presented to the user as level cards containing a sharing button (Figure 14). The level cards are also used to when viewing a user's levels under their player profile. When a user exits a level, the user is presented with a rating dialog, which also contains a share button.



Figure 13. A view after publishing a level

To have a consistent experience with the platform's sharing conventions, the share button's icon is made to resemble the sharing icon used on that platform as seen in Figure 14.

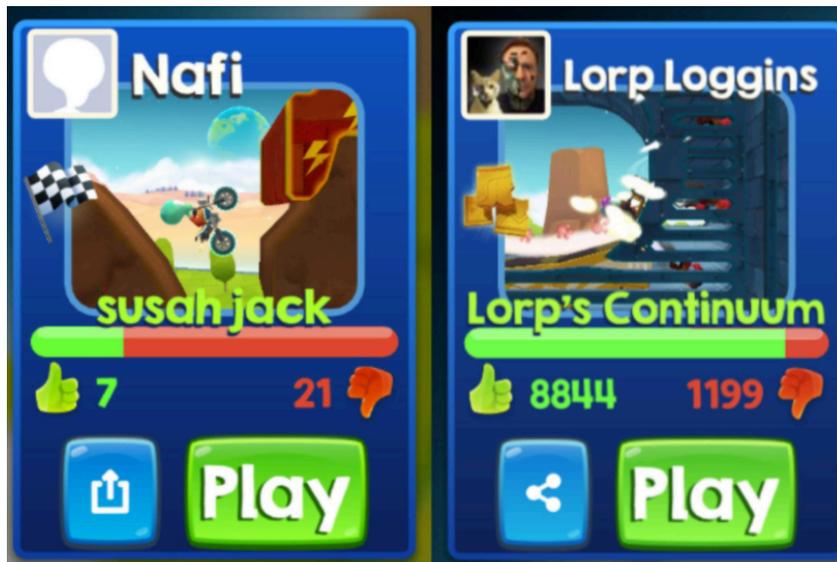


Figure 14. Level cards on iOS and Android

4.2 Server development

4.2.1 Why to have a deep link landing page?

Sometimes tapping a deep link fails to open the associated application even though it has been set up correctly from the platform's point of view. Ideally, the application would always open when pressing a supported deep link. The operability is dependent on the application from which the deep link was tried to be opened. Mobile web browsers and system applications such as SMS-app, notes, always work. Some popular applications, e.g. Twitter and Facebook, will instead open the link in a webview (WebView on Android, WKWebView on iOS) inside the application, and the deep link fails (Figure 15). Webviews are used on both platforms to display web pages within the application, rather than launching a separate web browser.

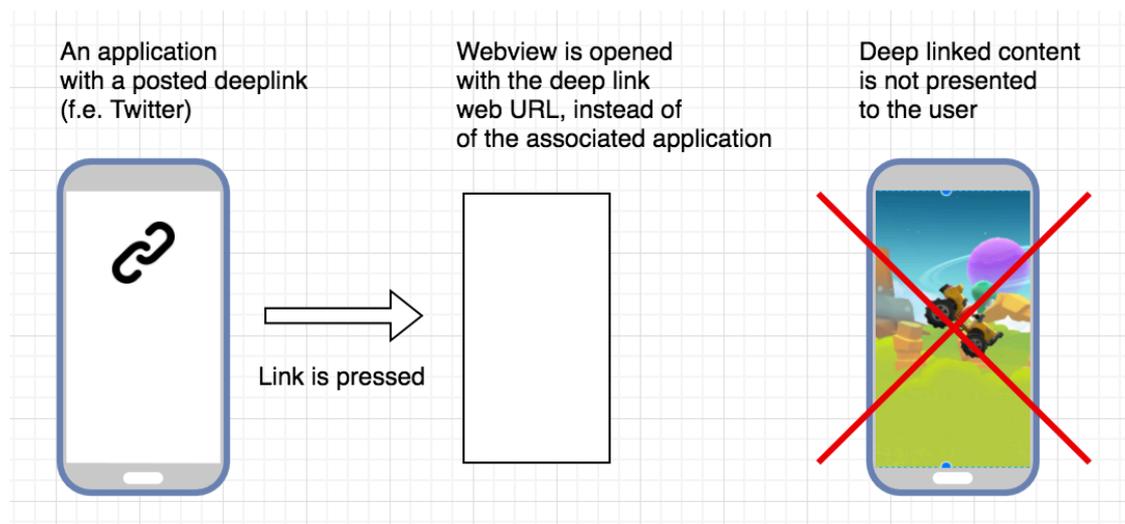


Figure 15. Deep link is opened in a webview and fails

It is not clear if this is an intended behaviour in those applications to prevent users from leaving to another application. Applications with a webview for opening web URLs will most probably open it in the webview and fail to launch the deep linked application. However, for example WhatsApp uses a webview to present web URL but is able to open a web URL in an associated application in case of a deep link.

To have a way around this, a landing web page for a deep linked content is needed. If a link supposed to open an application is pressed within an application's webview, the associated application will open in almost every case and the deep linked content can be presented to the user (Figure 16). The landing page is essentially a necessary layer in the flow of presenting the deep linked content, which is also used to show relevant information about the content to the user. The landing page has a Play button linking to the URL that is capable of launching the deep linked application. The reason for using a button to launch the application is because iOS will not launch the associated application if the Universal Link was not triggered by the user. In other words, JavaScript or server redirect do not work.

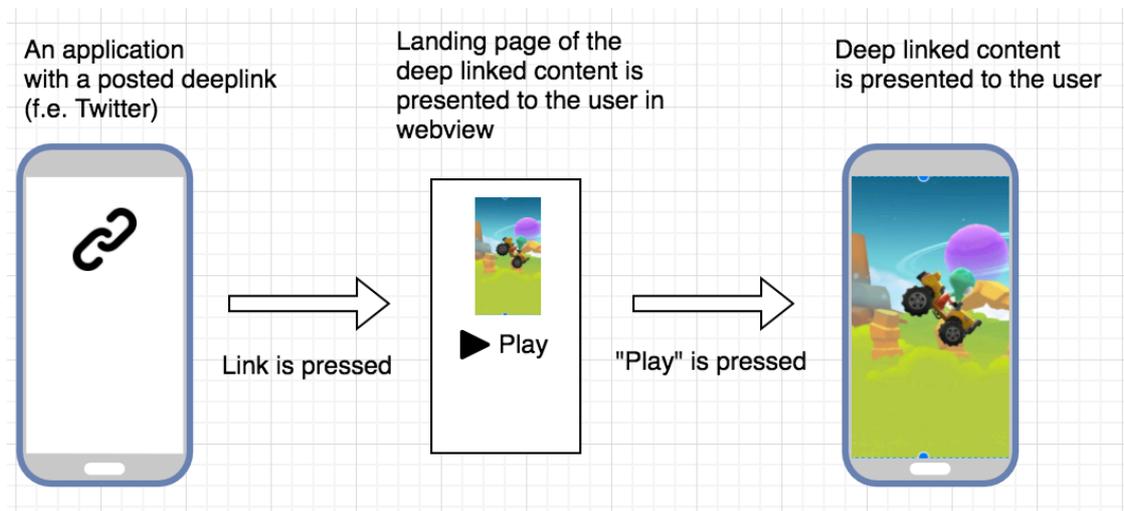


Figure 16. Deep link experience with a landing page through a webview

With deep link landing pages, users share an URL to the landing page instead of the URL able to open the application. This way it is secured that the deep link flow does not break with webview applications such as Twitter. When pressing the Play button, and the operating system does not launch the associated application (the user does not have the application installed), the client will instead load in the URL and will be redirected to the platform specific app store by the server. The landing page URL's domain needs to be different from the Universal Links domain for the application to launch on iOS. (Figure 17)

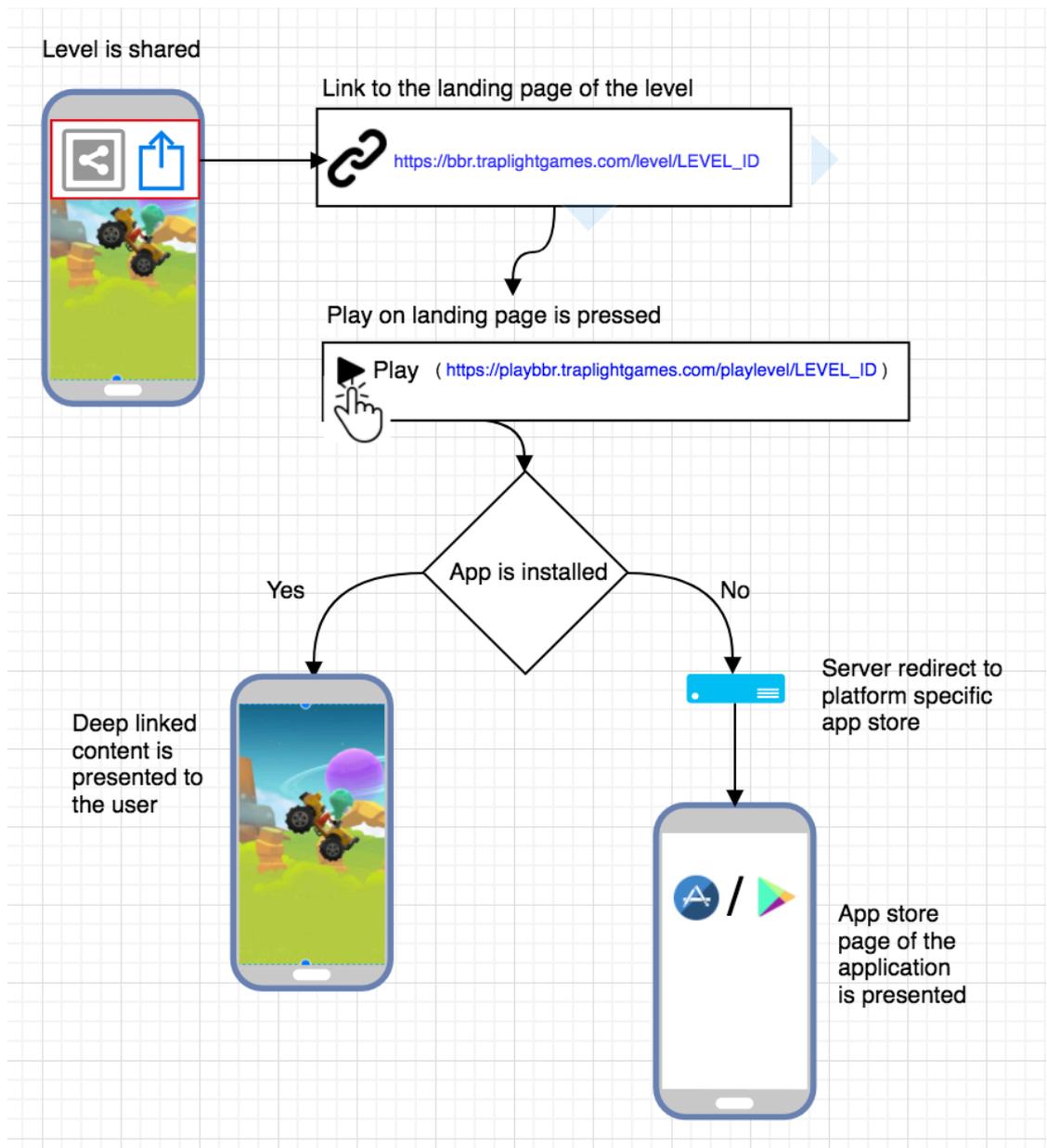


Figure 17. Deep link flow with a landing page

4.2.2 Deep link landing page

A level's deep link landing page is made to look familiar to the user with a similar visual style as the game's level loading screen with the same information about the level (Figure 18).



Figure 18. Deep link landing page for a level

In Big Bang Racing, the backend is using Play Framework and the programming language is Scala. The used database is MongoDB. The routes for HTTP requests are defined in the routes file:

```
GET /level/:id/      controllers.DeepLink.levelLandingPage(id)
GET /level/:id      controllers.DeepLink.levelLandingPage(id)
GET /playlevel/*id  controllers.DeepLink.redirectToStore(id)
```

A route consists of the HTTP method (GET / POST), request path and an action in which the request translates to. For deep linking, there are actions for showing a level landing page and a redirect to the user's app store. The unique identifier of the level is passed as a parameter to the action that handles creating the landing page.

```
def levelLandingPage(id: String) = Action.async {
  implicit request => getLevel(id)
}
```

The function `levelLandingPage` returns the result from `getLevel` which takes the level id as a parameter.

```

def getLevel(id: String) : Future[Result] = {
  IdValidator.validate(id) match {
    case Valid =>
      MiniGameMetaDao.get(id) map { _ match {
        case Some(game) => Ok(html.deeplink.main(game,
          html.deeplink.meta(game)))
        case None => Ok(html.deeplink.deeplinkErrorPage(id))
      }}
    case _ => Future(Ok(html.deeplink.deeplinkErrorPage(id)))
  }
}

```

The id parameter is a MongoDB's ObjectId in string format. First, the function checks if the id is a valid ObjectId. If the id is valid, a search for a level with that id is made. If a matching document is found, an HTML deep link landing page with that level's information will be returned to the client. In other cases, the result is an error page. (Figure 19)

Play Framework comes with Twirl web template engine used to create HTML documents dynamically. The template engine allows the use of Scala code inside a template. Templates act as functions and can therefore take parameters. If a template takes parameters, they must be declared on the top of the template file:

```
@(game: model.MiniGameMetaModel, meta: Html)
```

The deeplink.main template takes two parameters. The first is MiniGameMetaModel class instance, which is used to get information about the level. The second parameter meta is an HTML block containing the needed meta tags for different web crawlers. The template uses @ as a special character indicating the beginning of a dynamic statement. The syntax for using the dynamic statements is as follows:

```

<div id="creatorimage" class="creatorarea">
  @imageUri = @{
    game.creatorFacebookId match{
      case Some(id) => "https://graph.facebook.com/" + id +
        "/picture?width=200&height=200"
      case None =>
        routes.Assets.at("images/levelpage/profilepic.png")
    }
  }
  
</div>

```

The preceding example declares an expression `imageUri`, which is used to display the creator's profile image in the HTML document. If the creator has a `facebookId`, in other words the creator has connected the game to Facebook, the `facebookId` is used to get the creator's Facebook profile picture. If the creator does not have a `facebookId`, a default profile picture will be used. Other information such as the name of the level and the creator's name are stated in a similar fashion:

```
<div id="levelname">
  <h2>@game.name</h2>
</div>
<div id="creatorname">
  <h2 id="creatorNameFont">@game.creatorName</h2>
</div>
```

The play button on the landing page is a link to a web URL that has been set up to launch the application on each platform.

```
<a href="https://playbbr.traplightgames.com/playlevel/@game.id"
id="levellink">
  <button class="btn" type="button">Play level</button>
</a>
```

However, if the application is not installed, the web URL will be loaded. The HTTP request is translated into `DeepLink.redirectToStore` action, which will redirect the client to the platform specific app store (Figure 19). If the game is not available for the user's platform, a redirect to the home page of Big Bang Racing will be made:

```
val patterniOS = "(iPhone|iPad)".r.unanchored
val patternAndroid = "(Android|SAMSUNG)".r.unanchored

val storeAndroid =
  "https://play.google.com/store/apps/details?id=com.traplight.bigbangracing"
val storeiOS = "https://itunes.apple.com/us/app/big-bang-racing/id1048896882"

def redirectToStore(id: String) = Action { implicit request =>
  request.headers.get("User-Agent") match {
    case Some(patternAndroid(a)) => Redirect(storeAndroid)
    case Some(patterniOS(a)) => Redirect(storeiOS)
    case _ => Redirect("http://www.traplightgames.com/big-bang-racing/")
  }
}
```

The redirect is made by pattern matching the user agent information included in the HTTP request's header.

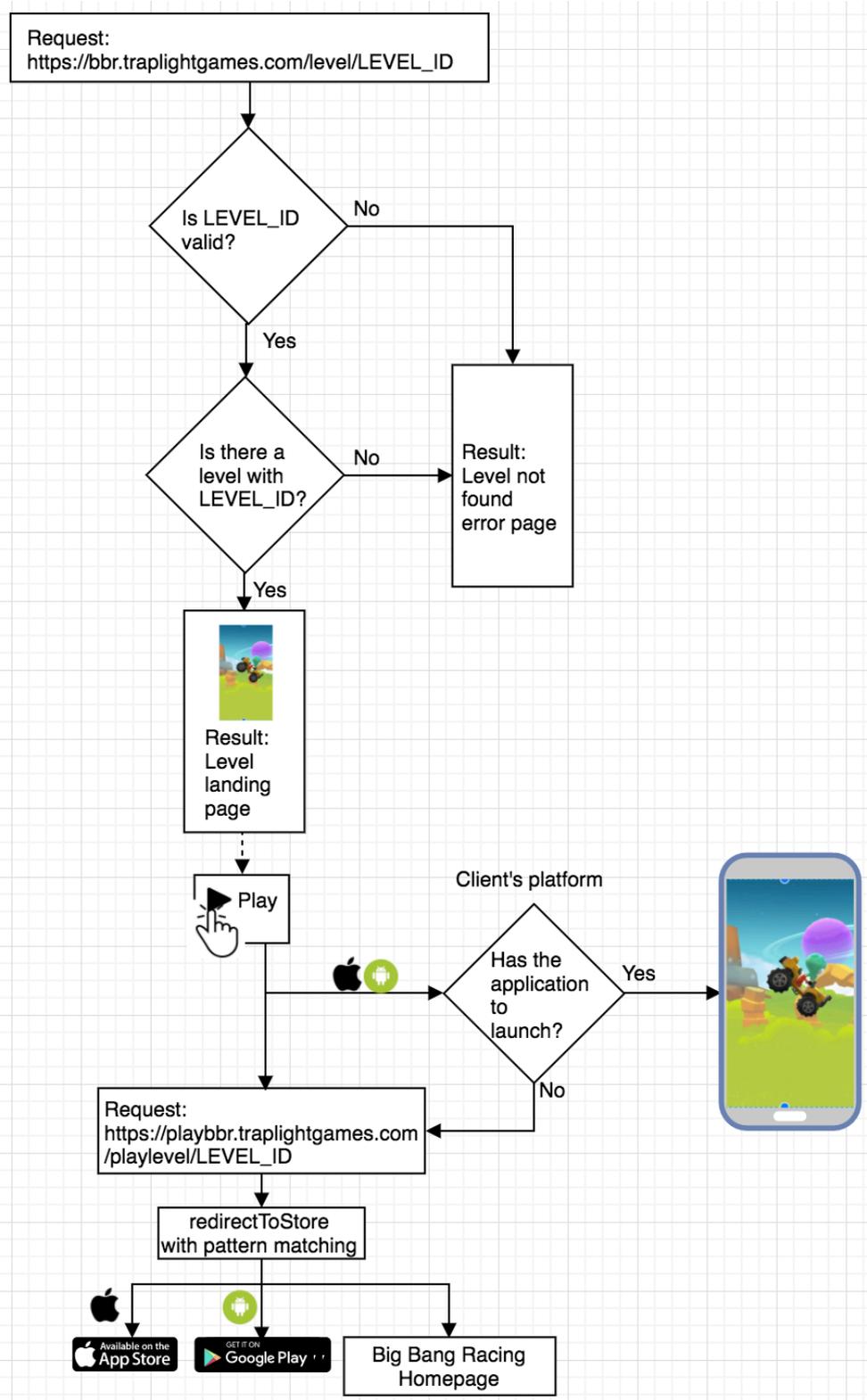


Figure 19. Deep link logic flow from server's perspective

4.2.3 Rich social media posts

The appearance of a posted deep link does not affect the functionality of the deep link itself from a technical standpoint. However, from a marketing perspective, it is crucial to consider the appearance of deep links and to make them easily approachable. A deep link posted to a social media channel can reach plenty of potential users, who will make their first impressions of the product based on the link and possibly download it. Also, having user-friendly deep links can encourage users to share more.

Rich posts to social media channels are possible when proper metadata about the link's content is provided for web crawlers to scrape. In HTML, metadata is provided in the form of meta tags. The metadata is dynamically created and given to the main template as a HTML parameter.

```
html.deeplink.main(game, html.deeplink.meta(game)))
```

In the main template, the metadata is placed to the document's head container. The Facebook Crawler scrapes metadata using the Open Graph Protocol.

```
<meta property="fb:app_id" content="1374618442808673" />
<meta property="og:type" content="bigbangracing:level" />
<meta property="og:url"
content="https://bbr.traplightgames.com/level/@game.id" />
<meta property="og:title" content="@game.name"/>
<meta property="og:description" content="@game.creatorName created
level '@game.name' ">
<meta property="og:image"
content="https://bbr.traplightgames.com@routes.Minigame.findScreensho
t(game.id)"/>
```

To have valid metadata for Facebook Crawler, at least the required tags `og:type` and `og:title` need to be included. By having `fb:app_id` allows the Facebook Crawler to associate the data with the application registered to Facebook, which lets the admins of the application view sharing insights at developers.facebook portal. Tag `og:url` indicates the canonical URL for the linked content. By providing a title, a description and an image allows Facebook to construct a rich snippet of a post containing a deep linked level (Figure 20).

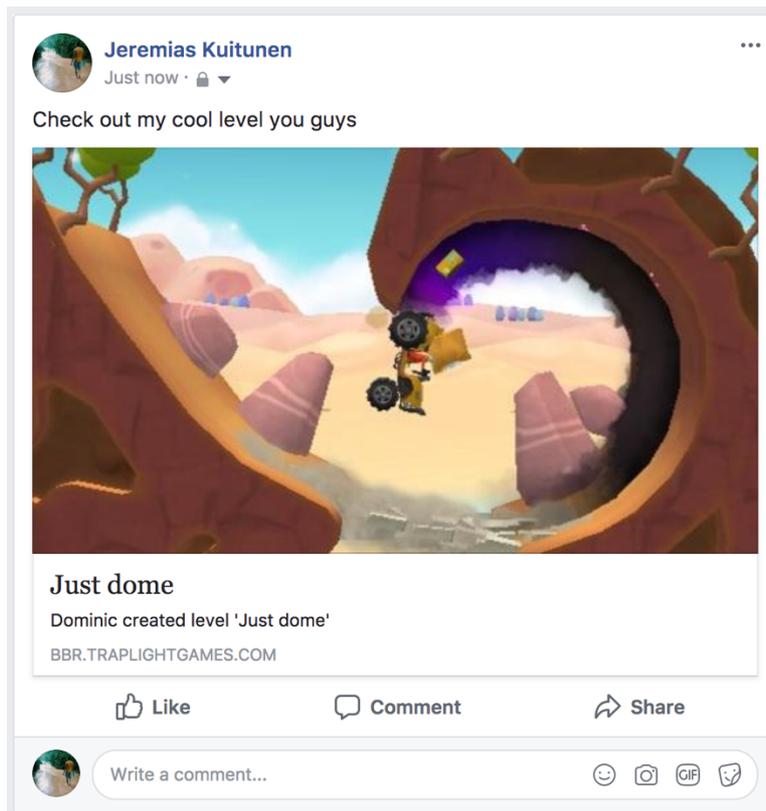


Figure 20. Facebook constructed rich snippet of a post containing a deep link

Twitter constructs a Twitter card (Figure 21) of a tweet containing a deep link if the necessary metadata is provided. The metadata for Twitter is defined in a similar way as for the Open Graph Protocol, containing title, description and image:

```
<meta name="twitter:card" content="summary_large_image">
<meta name="twitter:site" content="@BBangRacing">
<meta name="twitter:title" content="@game.name">
<meta name="twitter:description" content="@game.creatorName created
level '@game.name'">
<meta name="twitter:image"
content="https://bbr.traplightgames.com@routes.MiniGame.findScreensho
t(game.id)">
```

For Twitter, the type of the card needs to be specified. For deep linked levels, a Summary card with large image is used. Other options are Player card, App card and Summary card. To associate the card with Big Bang Racing's Twitter account, twitter:site is defined with the account name.

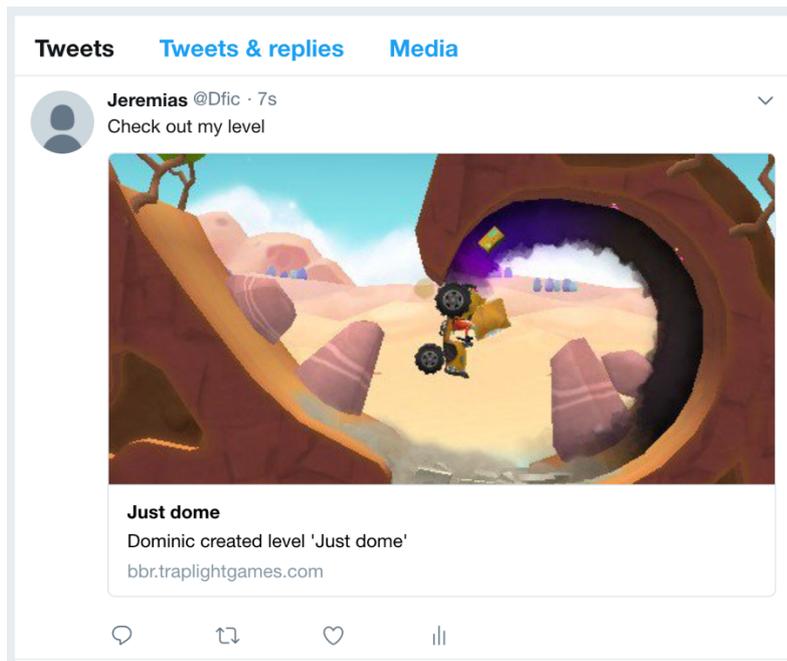


Figure 21. Twitter's generated *Summary card with large image*

By providing metadata using the Open Graph Protocol also allows many other social media platforms' web crawlers to scrape the necessary information for creating an informative presentation of the link. The supported platforms include Facebook messenger, Google+, Slack, LinkedIn and WhatsApp (Figure 22).

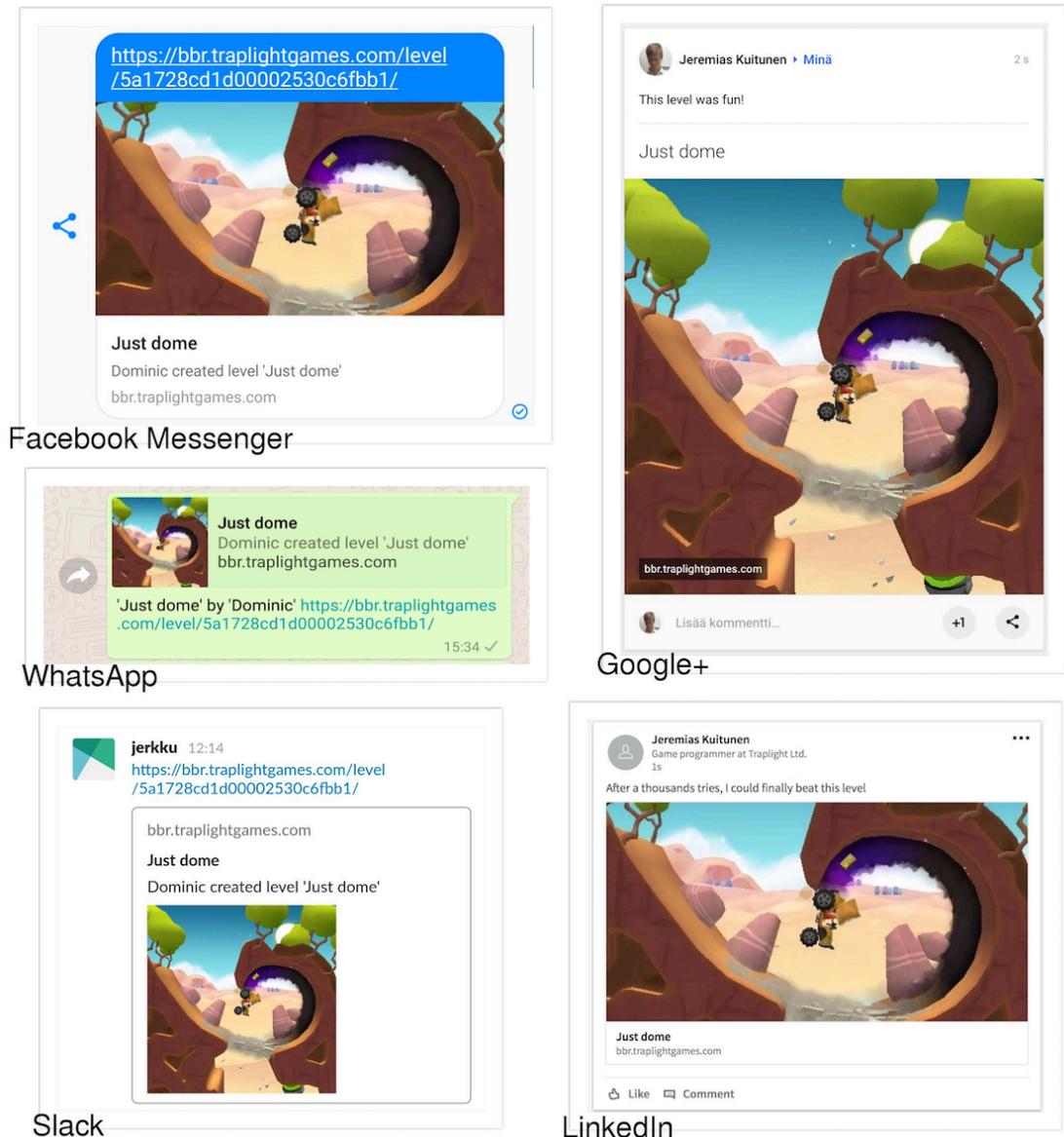


Figure 22. A post containing a deep link on different social media platforms

4.3 Facebook's App Links for Android

Facebook has developed its own App Links to tackle the issues with deep linking. However, by doing so, Facebook does not allow deep linked applications to be launched from their Android application with the usual set-up. Instead, Facebook requires Facebook's App Links to be implemented with the application.

A custom URI scheme needs to be added to the main activity of the application in `AndroidManifest.xml` file.

```

<activity android:name=".DeepLinkActivity"
android:label="@string/app_name" >
  <!-- Facebook App Links -->
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="bigbangracing" />
  </intent-filter>
  <!-- other activity related definitions -->
</activity>

```

To make Facebook launch the application with that URI scheme, the URI needs to be defined in the head element of the deep link HTML as a meta tag. Also, the application name and package name need to be defined.

```

<meta property="al:android:url"
content="bigbangracing://playlevel/@game.id" />
<meta property="al:android:package"
content="com.traplight.bigbangracing" />
<meta property="al:android:app_name" content="Big Bang Racing" />

```

4.4 GIF sharing with deep linking

Dying in Big Bang Racing can be considered as a funny moment. The vehicle explodes leaving a crater in the ground while the ragdoll character spins uncontrollably in the air. When the player dies in Big Bang Racing, it is recorded and the recording is given for the user to preview (Figure 23). By pressing the preview, the user can share the recording as a GIF.

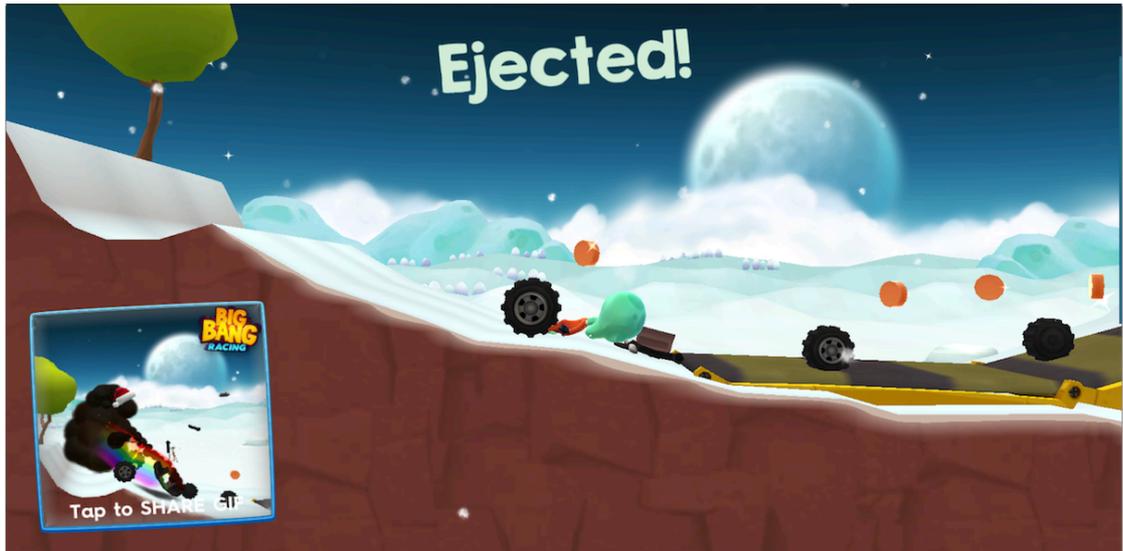


Figure 23. Preview of GIF

With deep linking, a shared GIF can be made interactive by allowing users to enter the same level where the GIF was recorded from. Basically, a shared GIF is a deep link to a level, however, with an entertaining recording.

The game constantly captures frames to a designated buffer. When player dies, and decides to share the recording by tapping the preview, the frames in the buffer are encoded to an animated GIF. The GIF is then sent to the server to be stored and a deep link URL to the GIF is provided for the client in the HTTP request's response.

The URL is a regular deep link URL with the GIF's identifier:

```
https://bbr.traplightgames.com/level/57a338a03e00004100f81179/56be24e23300008300df657f1511781792929
```

To have a landing page support for GIFs, an additional route in the routes file is needed.

```
GET /level/:id/gif      controllers.DeepLink.levelLandingPageGif(id:
String, gif: String)
```

The requests translate into `DeepLink.LevelLandingPageGif`, which takes the level's id and the GIF's id as parameters. If the identifiers are not valid, an error page will be presented to the user. If both ids are valid, a deep link landing page containing the animated GIF will be returned to the client as a result:

```
Ok(html.deeplink.main(game, html.deeplink.metagif(game, gif), gif))
```

The deeplink.main template's first line is modified so it can take a GIF identifier as an optional parameter:

```
@(game: model.MinigameMetaModel, meta: Html, gif: String = null)
```

Also, the template is modified so that the picture for the deep link landing page is the GIF if it has been provided, as seen in the example below.

```
@defining(if (gif == null)
  routes.Minigame.findScreenshot(game.id)
else
  "http://bbr.gifs.traplightgames.com.s3.amazonaws.com/"+game.id.reverse+gif+".gif"
)
{imagesource =>
  
}
```

To make the GIF visible when GIF deep link is posted to Facebook, proper metadata needs to be provided using the Open Graph Protocol (Figure 24). The most important difference compared to a normal deep link is that the canonical URL og:url needs to be set to the actual path of a GIF file instead of the landing page's path. This way Facebook is able to display GIF on its feed.

```
<meta property="og:url"
content="http://bbr.gifs.traplightgames.com.s3.amazonaws.com/@{game.id.reverse + gifId}.gif"/>
```

The image property is set to be the same path. Also, the size of the image is defined to be more suitable for the GIF.

```
<meta property="og:image"
content="http://bbr.gifs.traplightgames.com.s3.amazonaws.com/@{game.id.reverse + gifId}.gif"/>
<meta property="og:image:width" content="200">
<meta property="og:image:height" content="200">
```

The title and the description are set to fit the context of the GIF:

```
<meta property="og:title" content="@game.name killed me"/>
<meta property="og:description" content="Level '@game.name' by '@game.creatorName' was way too dangerous for me!">
```

If for any reason Facebook is not able to open display the GIF, a fallback image is set to be the normal level picture:

```
<meta property="og:image"
content="https://bbr.traplightgames.com@routes.MiniGame.findScreenshot(game.id)"/>
<meta property="og:image:secure_url"
content="https://bbr.traplightgames.com@routes.MiniGame.findScreenshot(game.id)"/>
```

Clicking the posted GIF will take user to the landing page of the level, where the level image is replaced with the GIF.



Figure 24. GIF deep link posted to Facebook

Twitter unfortunately does not support the GIF format in their Player card, and therefore a tweeted GIF deep link will have the picture of the normal deep link.

5 Results and discussion

5.1 Deep linking in Big Bang Racing

Big Bang Racing tracks metrics of how the game performs. Different metrics can be analyzed to understand the usage of the deep linking features. The values in the examples represent the metrics that have been tracked between 14.4.2017 - 29.10.2017.

One fourth of Big Bang Racing's players publish levels (Figure 25). Of all players, 10.7% have shared at least one level. Of all users, 7.5% share GIFs. The time that it takes to encode a GIF and send it to server could be one reason why GIF sharing is less popular than level sharing. The sharing metric tracking is triggered when a sharing dialog is presented to a user, which does not guarantee that the user actually shares the deep link URL to others, as it is possible to cancel the sharing.

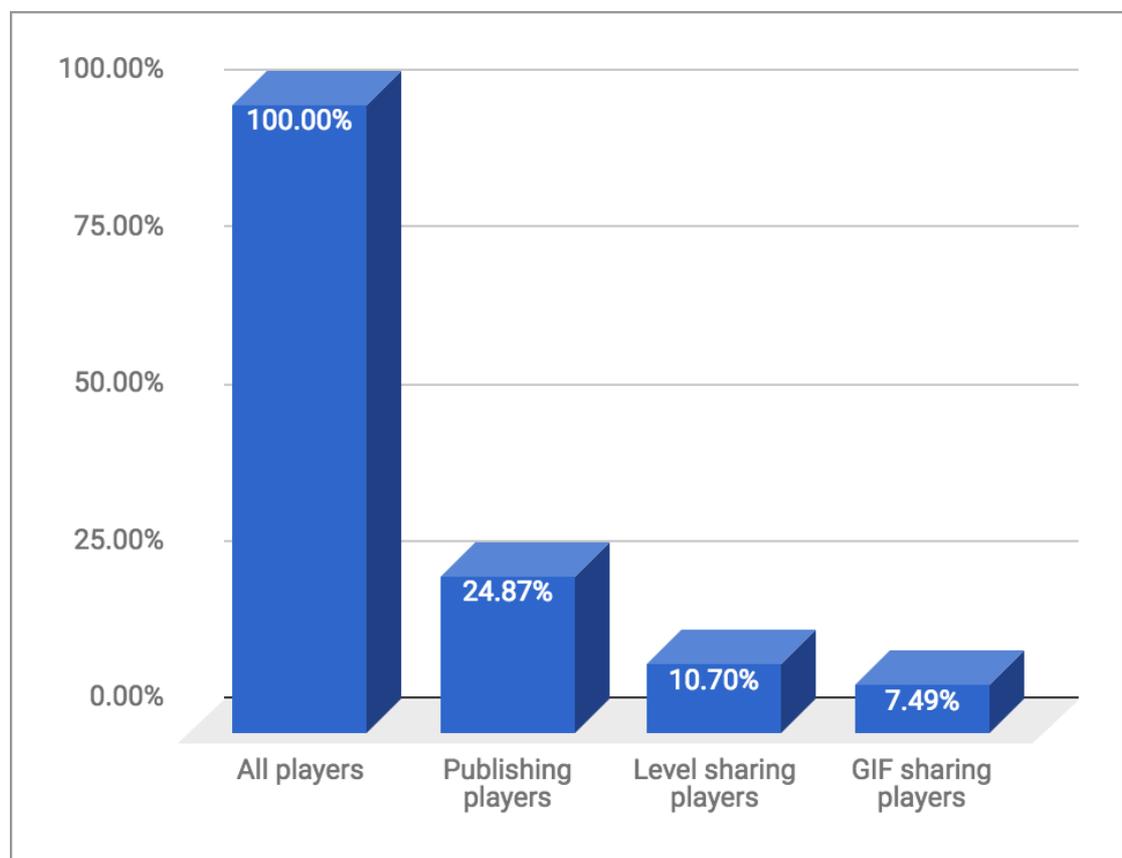


Figure 25. Chart of deep link features used by users

Of all level shares, 58.6% come from players who also publish levels, which can be explained by the fact that players mostly share their own levels (Figure 26).

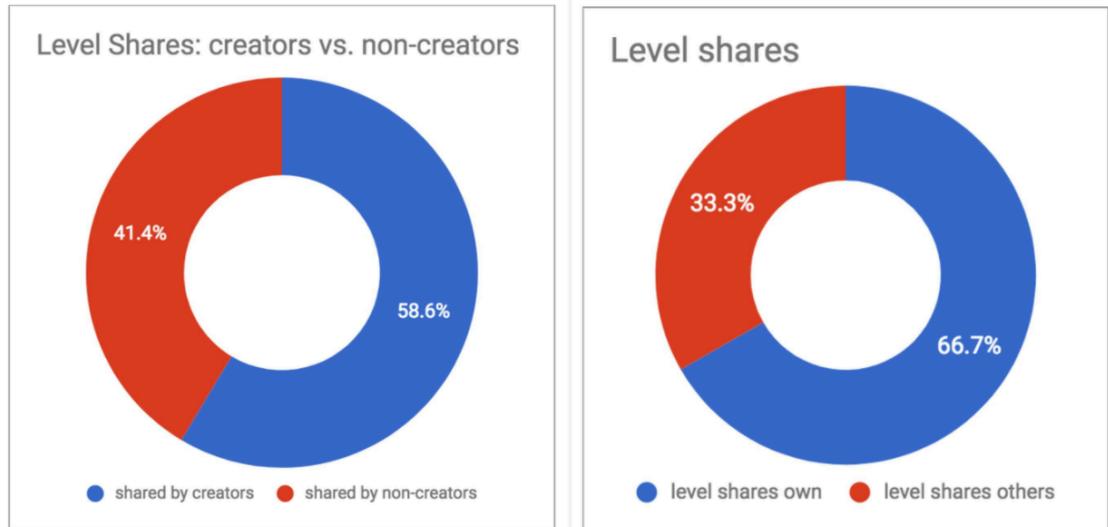


Figure 26. Sharing of levels by player type and users sharing own levels

According to the metrics, deep links have been opened only 2.66% times compared to total amount of deep link share events (Figure 27).

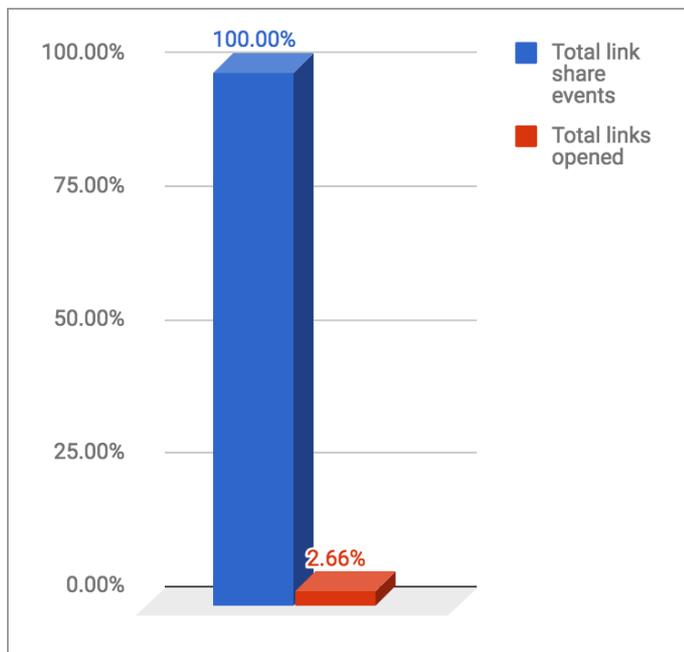


Figure 27. Comparison of deep links opening the application versus total share events (GIFs and levels)

There are multiple factors that can partly explain the significantly lesser amount of links being opened than shared. It should be noted that not all sharing events translate to actual sharing of the link to other people, which makes the opening of a link percentage seem smaller. Also, if a user without the application clicks a deep link, (s)he is redirected to the App Store page of the game. If the user installs the game and enters it, no metric event will be triggered as the web URL did not launch the application. A further inspection revealed an issue regarding players who have not played the first tutorial level. If a player has not finished the first tutorial level when opening a deep link, the user will not be taken to the shared content, as the first level is required to be played first by design. However, the metric event is being called only when a player has been taken to the shared content. This makes the opening of deep link metric lesser than it actually is and therefore more unreliable.

To have more reliable data regarding the deep linking functionality, the preceding issue should be fixed by tracking all application launches by web URLs. Currently, all of the game's metrics are tracked from the client. The landing page events, such as pressing the play button and redirects to app stores, should also be tracked for more meaningful data.

The tracked metrics do not offer much more to be analyzed in their current form. However, the data should be considered as a baseline for future deep link related metrics. If a Big Bang Racing update affecting deep linking was to be published, the new metrics should be compared to the existing data to see if the changes were for better or worse. Big Bang Racing's metrics should also be kept in mind when examining deep link usage in a new product.

To improve the sharing of animated GIFs, the format should be changed to another format, such as WebM. WebM is more suitable for sharing videos online, as the file size is smaller which decreases the required bandwidth usage and storage capacity. Also, WebM is more compatible with social media channels, such as Twitter. A WebM video can be played in a Twitter's Player card, which is not possible using an animated GIF.

To make better use of deep linking in Big Bang Racing, it should be made possible to share teams and player profiles. Also, it would be interesting to see how deep links are used with a game feature that was designed specifically for deep linking, e.g. a game mode, where users challenge others to race only through deep links.

5.2 Further improvements for DeepLink solution

The developed deep linked solution works as intended; however, the development could be taken further to make it more versatile. Deferred deep linking allows the same deep linking functionality even though the application is not preinstalled on the platform. In other words, when the user opens an application installed through deep link app store redirect, the deep linked content would be presented to the user (Figure 28). The challenging part is that neither iOS nor Android supports passing data through app store to a newly installed application. One way to have this functionality is to use the backend of a game to store identifying information of the user that was redirected to an app store. This identifying information would be matched with connections coming from new users in the game. If a match was found, the server would provide the information regarding the clicked deep link and the client would present the deep linked content to the user. The identifying information could include clients IP address, information about the operating system, screen size, device model with other information. However, such matching is somewhat prone to errors, especially when several new users with the same IP address install the game (school campuses, public networks, players using VPN).

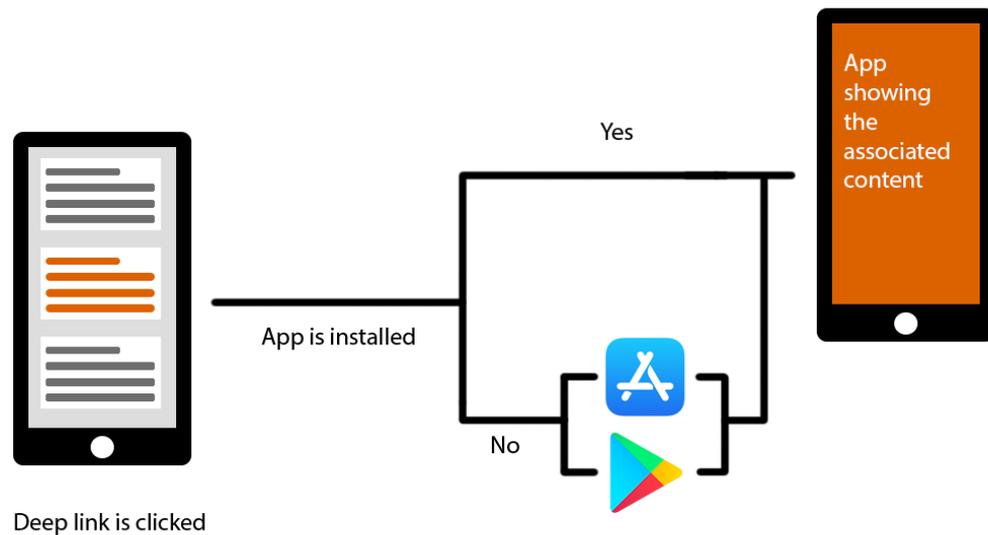


Figure 28. Deferred deep linking

Android App Links are Google’s equivalent to iOS’s Universal Links. Both require a configuration JSON file to be uploaded to the domain that can launch the application. With the current deep link solution, when a user opens a web URL that can launch an application, a system dialog will appear. The dialog asks whether to open the URL in the deep linked application or in a browser. Since both applications can handle the URL, it is up to the user to choose the right application. With Android App links, the experience is made more seamless by removing the dialog, thus automatically opening the deep linked application. Android App Links is supported from Android 6.0 onwards.

A level deep link landing page serves its purpose by making deep links work with applications using webviews. However, from a user’s point of view, it is an extra step to get to the shared content. A landing page requires a user to press the play button, which will fundamentally decrease the success rate of the deep link. On iOS, the user triggerable play button is necessary. However, it is worth researching if an Android user could be taken to the associated application with a server redirect or with client JavaScript redirect from the landing page. The redirect should point to the web URL registered to open the application. Naturally, if it were to work, the redirect should

only be made to affect Android platform, which can be done by identifying the platform from an HTTP request's user-agent. Based on initial testing, the Android redirect looks promising, as it worked flawlessly with WhatsApp on Android. GIF landing pages, however, serve a purpose of presenting the animated GIF to the viewer and should not be automatically redirected.

For existing users, the deep link landing page is informative and has the familiar design from Big Bang Racing. However, for new users, the landing page can be intimidating as there is no pre-existing reference to the game's visual appearance it was made to resemble. The landing page should be made look cleaner, more informative about the game, and therefore more user-friendly for new players.

Even though players of Big Bang Racing share user-generated content, the actual shared deep links are slightly generic and lack personal touch. When a level is shared, the metadata description for web crawlers about the deep link is always "creator_name created a level level_name". Also, the description for a GIF recording is always "Level level_name was way too dangerous for me!". To have more versatile representations of deep links, distinctive deep link sharing should be considered. This would allow having metadata that is more personal (Figure 29).

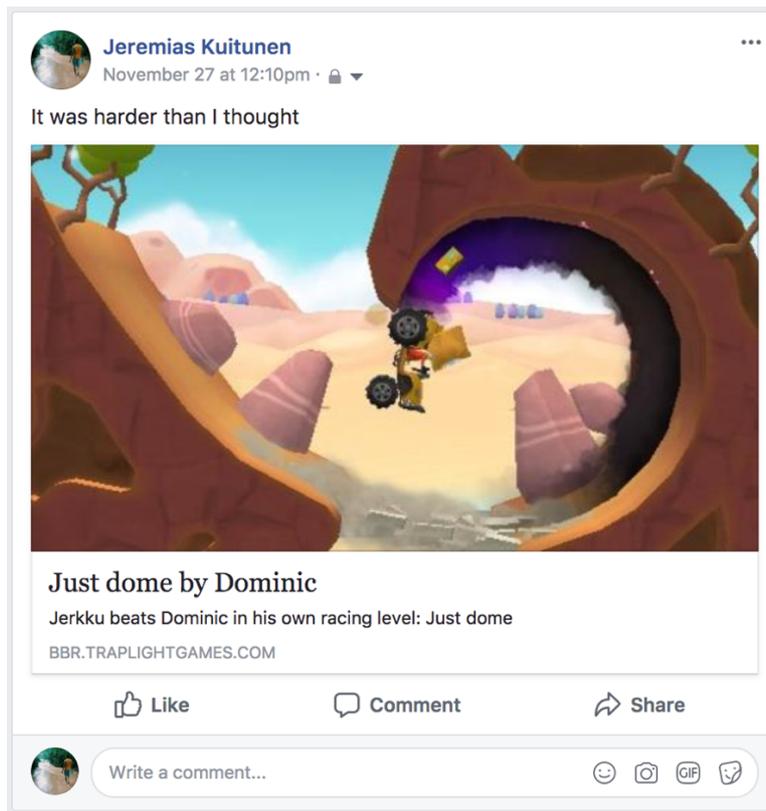


Figure 29. More personalized deep link sharing

In theory, more personal deep links should be implemented by creating a data object of every distinctive deep link share. Basically, if a user presses a sharing button, a server request would be made for the sharing instance. The server creates the contextual data object and returns a web URL with a reference to that object for the client to share. The data object would consist of information about the share:

```
{
  "id": "unique_id",
  "sharedContent": {
    "type": "gif",
    "level": "level_id",
    "gif": "gif_id",
    "deathReason": "electrocution"
  },
  "sharer": "player_id",
  "timestamp": "time_of_sharing"
}
```

This would allow having personalized metadata for the deep links. Also, the web URLs would be cleaner as there was no need to form the URL from many ids (compare to GIF URLs). When the application was launched with this kind of URL, a server request would have to be made in order to present the deep linked content to the user. With better analytics tracking, this would allow better understanding of what kind of deep links are more likely to get users attention and help to adjust accordingly.

5.3 Development of Unity plugins

The developed way of communicating between the platform's native and Unity code came in handy, when Big Bang Racing's push messaging plugin needed to be updated to support data payloads. The data payload included in a push message could be provided to Unity's side by saving the payload to PlayerPrefs from the native code using pre-existing method `SaveStringToPlayerPrefs`. The payload can be used to take the user to the correct location within the game when the user clicks a push notification.

At the time of developing the plugin, there was no official Unity documentation of where the PlayerPrefs were stored. Also, after Unity 5.5 update, Unity changed the filename of where PlayerPrefs were stored on Android without any update notes mentioning it. The new filename added "v2" in the end. According to Unity 2017.2 documentation, the filename has been changed back to what it was prior Unity 5.5. Updates such as these make fully functional plugins incompatible with new Unity versions. Without any official documentation or update notes, these kinds of changes can make updating Unity plugins time consuming.

5.4 Other deep linking solutions for Unity in 2017

Branch offers a deep linking solution called Branch Metrics Unity SDK. Branch is specialized in deep linking and business analytics. The SDK supports deferred deep linking, referral systems and analytics. The SDK supports both Android and iOS.

Google has a free deep linking solution called Dynamic Links. Dynamic Links is a part of Google's Firebase mobile and web development platform. Firebase Dynamic Links

offer a wide range of features, including deferred deep links with user referral rewards, analytics and social metadata. Dynamic Links support both iOS and Android platforms as do other Firebase Unity SDKs. Other SDKs Firebase offer to Unity include:

- Firebase Analytics
- Cloud Storage
- Firebase Cloud Messaging

The go to push message solution for Android is Firebase Cloud Messaging (FCM), since it is the Google's official supported push message system for Android. If a new Unity Android game has push messaging capability, it probably is using FCM. A single Firebase project (created from the Firebase console) can be used with all the Firebase SDKs for Unity, which makes setting up multiple Firebase SDKs for Unity relatively easy.

Currently, the easiest way to achieve deep linking features in a mobile game is by deploying a third party deep linking solution. However, it is worth considering the drawbacks of using a third-party solution. The drawbacks include the reduced control over the feature and that the SDK works under the terms of the SDK's developer. In other words, if a user of a SDK is not satisfied with how it works, there is not much else to do but adapt to it. The advantages include the variety of features, ongoing support and the level of know-how that the specialized SDK developers have. For example, if an operating system update makes current deep link solutions incompatible, a quick respond from the SDK's developer to the issue can be expected. Integrated analytics make both Branch and Firebase Dynamic Links SDKs attractive choices for Unity developers. Firebase Dynamic Links is worth considering especially if other Firebase ecosystem's SDKs, such as Analytics or Firebase Cloud Messaging, are already implemented to the Unity project.

5.5 Current state of mobile deep linking

Developing a mobile deep linking solution revealed some issues with the current way mobile deep links work. Even though an application is set up to launch from a web URL from the operating systems perspective, the application from which the link was

opened can prevent it from working. For some applications, the answer is to have a deep link landing page, which has the URL that can launch the application as a link. However, for example a certain version of Slack for Android did not work even with a landing page.

At the moment, feature rich, fully compatible deep linking solution can feel unnecessarily complicated to keep up to date and working. Facebook pushing App Links only compatible with their own applications makes deep linking more fragmented. Standardized deep linking conventions would help supporting every possible application easier, without having to worry about supporting applications individually. Also, from the application developer's perspective, the mobile operating systems should have the support for deferred deep linking. Currently, supporting deferred deep linking requires gimmicky procedures that are more or less likely to break with the major mobile operating system updates.

References

- AndroidJavaClass. N.d. Unity documentation about AndroidJavaClass. Accessed on 22 November 2017. Retrieved from <https://docs.unity3d.com/ScriptReference/AndroidJavaClass.html>
- AndroidJavaObject. N.d. Unity documentation about AndroidJavaObject. Accessed on 22 November 2017. Retrieved from <https://docs.unity3d.com/ScriptReference/AndroidJavaObject.html>
- iOS 9.0. N.d. Apple's article of new features in iOS 9.0. Accessed on 9 December 2017. Retrieved from <https://developer.apple.com/library/content/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html>
- iOS Support Matrix. N.d. Supported Apple devices by different iOS versions. Accessed on 9 December 2017. Retrieved from <http://iossupportmatrix.com/>
- Kemp, Simon. 2017. Digital in 2017: Global overview. Accessed on 11 October 2017. Retrieved from <https://wearesocial.com/special-reports/digital-in-2017-global-overview>
- Khira, Firoz. 2015. General Sharing in Android & iOS in Unity. Accessed on 22 November 2017. Retrieved from <http://www.theappguruz.com/blog/general-sharing-in-android-ios-in-unity>
- Maddern, Chris. 2015. A Brief History Of Deep Linking. Accessed on 13 of October 2017. Retrieved from <https://techcrunch.com/2015/06/12/a-brief-history-of-deep-linking/>
- Mobile Gaming: Social Motivations. 2014. Study commissioned by Everyplay. Accessed on 11 October 2017. Retrieved from http://files.unity3d.com/everyplay/Mobile_Gaming_Social_Motivations.zip
- Murphy, Kevin. N.d. Unity Game Engine Review. Accessed on 9 December 2017. Retrieved from <https://www.gamesparks.com/blog/unity-game-engine-review/>
- Smartphone apps crushing mobile web time. 20.6.2016. Accessed on 11 October 2017. Retrieved from <https://www.emarketer.com/Article/Smartphone-Apps-Crushing-Mobile-Web-Time/1014498>
- Support Universal Links. N.d. Apple's documentation about Universal Links. Accessed on 22 November 2017. Retrieved from <https://developer.apple.com/library/content/documentation/General/Conceptual/AppSearch/UniversalLinks.html>
- The 2017 Facebook User-Generated Content Benchmark Report. 2017. Accessed on 12 October 2017. Retrieved from <http://info.mavrck.co/facebook-user-generated-content-benchmark-report-q1-2017>

The New York Times Customer Insight Group. 2011. The Psychology of Sharing: Why do people share online content? Accessed on 11 October 2017. Retrieved from <http://www.iab.net/media/file/POSWhitePaper.pdf>

Traplight. N.d. Traplight website's page about the company. Accessed on 9 December 2017. Retrieved from <http://www.traplightgames.com/traplight/>