KARELIA UNIVERSITY OF APPLIED SCIENCES
Degree Programme In Business Information Technology

Teemu Kokkonen

MICROTRANSACTIONS IN AN ANDROID GAME

Thesis
June 2014

Karelia UNIVERSITY OF APPLIED SCIENCES	**THESIS** **June 2014** **Degree Programme In Business Information Technology** Karjalankatu 3 80220 JOENSUU FINLAND 013 260 600

Author(s)
Teemu Kokkonen

Title
Microtransactions in an Android Game

Commissioned by
-

Abstract

The objective of this thesis is to explore the products sold within mobile applications and games, called in-app purchases or microtransactions. The thesis studies the history and nature of these microtransactions and examines their positive and negative effects on game design, as well as analyzes their usage in modern mobile games.

To reinforce the research, a mobile game codenamed TownBuilder was developed alongside the thesis. The game paid attention to the designs explored in the earlier chapters and incorporated design choices that attempted to optimize the effectiveness of the microtransactions. In addition to the design, a portable code package was set to be developed to simplify the usage of the in-app purchases in conjunction with Google Play services.

The game that was developed during the thesis reached an early alpha stage, with the microtransaction functionality demonstrably in place and some gameplay to go with it. The design of the game was inspired by several successful examples on the field, and even though it is not finished, the design was set a solid foundation for including microtransactions in a player-friendly way.

With all its perks and disadvantages, it remains still yet to be seen if microtransactions are the way of the future for the games industry. Despite the success stories, there are a lot of alternatives, some of which yet undiscovered.

Language English	Pages 48 Appendices 7 Pages of Appendices 9

Keywords

microtransactions, in-app purchases, monetizing, free-to-play, Android, game development, Google Play

	OPINNÄYTETYÖ **Kesäkuu 2014** **Degree Programme In Business Information Technology** Karjalankatu 3 80220 JOENSUU FINLAND 013 260 600
Tekijä(t) Teemu Kokkonen	
Nimeke Microtransactions in an Android Game / Mikromaksut Android-pelissä Toimeksiantaja -	

Tiivistelmä

Opinnäytetyön tarkoituksena on tutkia mobiilipeleissä ja -sovelluksissa myytäviä tuotteita, eli sovelluksensisäisiä ostoksia, mikromaksuja. Opinnäytetyössä käydään läpi mikromaksujen historiaa ja luonnetta ja tutkitaan niiden hyviä ja huonoja vaikutuksia pelisuunnitteluun. Opinnäytetyössä analysoidaan myös mikromaksujen käyttöä moderneissa mobiilipeleissä, käyttäen esimerkkeinä Supercellin Clash of Clansia sekä EA:n Dungeon Keeperiä.

Työssä kehitettiin myös mobiilipeli, joka kulkee koodinimellä TownBuilder. Pelissä pyrittiin toteuttamaan mikromaksut mahdollisimman tehokkaalla menetelmällä. Peliin toteutettiin siirrettävä koodikirjasto, jonka tarkoituksena oli yksinkertaistaa mikromaksujen käyttöä Google Play -kaupan yhteydessä.

Opinnäytetyön osana kehitetty peli toteutettiin alfa-vaiheeseen asti. Peliin sisältyvät mikromaksuominaisuudet esiteltävässä muodossa sekä jonkin verran pelattavuutta. Pelisuunnittelussa otettiin inspiraatiota alan onnistumistarinoista, ja keskeneräisyydestä huolimatta onnistuttiin jättämään hyvä pohja tehokkaiden mikromaksujen sisällytykselle pelaajaystävällisessä muodossa.

Kieli englanti	Sivuja 48 Liitteet 7 Liitesivumäärä 9

Asiasanat
mikromaksut, rahoitus, free-to-play, Android, pelikehitys, Google Play

Contents

1 Introduction

This thesis explores the world of microtransactions, what they are and how to use them in a game made for the Android platform. The author is a gamer and a game programming student with heavy interest in game design. The inspiration for the technical execution for this thesis came from a course project game done in Android, which was done without external game engines, giving wide options for customization. In addition to just making an Android game, I decided to add a focus to a very recent topic, one which will be very useful on my career later on, which is why I decided to implement the microtransaction functionality to the game and research its origins. The thesis aims to answer the question ”how do I use microtransactions in an Android game?”, attempting to establish a knowledge base that allows for making informed designs monetizing a free-to-play mobile game, as well as to give practical pointers on how to implement microtransactions programmatically.

The thesis can hopefully provide something for readers from different milestones of their career. For the beginner the document might be a good introduction to free-to-play and general game funding, pointing out possible sources to follow to find out more as well as giving an overview of practices and opinions regarding the microtransactions. For the seasoned veteran the thesis may not directly bring any new information, but it should work as a window to a student-gamer point of view to game monetization. If nothing else, reading the thesis should strengthen the hope or fear towards the future of the industry.

In the second chapter I will go back in time and study the origins of what today are called microtransactions. The timeline begins from pre-online era of gaming, where conclusions between microtransactions and arcade gaming are made. Then the story moves to the world of today, where the different business models that are potentially competing with microtransactions are explained. Last a brief look will be taken at what the microtransactions today actually are, saving the deeper analysis for the next chapter.

After the brief history of the subject, I will delve deeper into the nature of microtransactions in the third chapter. Different usages of in-app purchases will be illustrated through examples, pointing out design choices for certain types of microtransaction items. After that the focus will shift to what characteristics emerge from the already established framework, this chapter will look at terms such as ”pay-to-win” and ”whales”.

As soon as the origin and being of microtransactions have been explored, the fourth chapter takes a look at two examples of games utilizing them. I will analyze Supercell’s Clash of Clans and EA’s Dungeon Keeper with the UK Office of Fair Trading guidelines, hoping to gain an objective view on how these two games market their in-app purchases.

In the fifth chapter an attempt will be made to utilize everything written so far in order to make my own free-to-play mobile game on Android that tries to optimize the profitability without upsetting players. The chapter also describes the portable Java MTX package, which is an attempt to increase the ease of use of microtransactions and to separate the vendor specific code from the actual game code.

2 The Timeline of Microtransactions

2.1 Pre-online era

By definition, microtransactions – or micropayments – mean the action of using a small amount of money to purchase something online (Cambridge University Press 2014). This thesis will specifically study the microtransactions used in video games, but the timeline of using small quantity of currency to purchase commodities in games starts from before the Internet was a common and popular thing like it is today. The history leads back to the 70’s and 80’s, to a time where the World Wide Web was still taking its baby steps (Cailliau 1995) and when video games were only found in halls and were played on wardrobe-sized machines that had buttons on them. These machines would require the player to insert coins in them before they could start playing. These coin-operated machines were called arcade video games.

While the arcade game business model might not be strictly defined as being offline microtransactions, it shares some features with the micropayment model. For example, in some arcade games such as Metal Slug or Double Dragon you could buy yourself extra lives in-game by inserting a coin. This could be comparable to a feature in some modern mobile and social games, where you have limited actions you can perform in the game, and if you want to refill your actions, you have to use or purchase an in-game item that will allow you to do so. The amount of money involved in the transaction is also similar in both cases: The arcade game machines often accepted the 25 cent coins, which - adjusted to today's currency - is around 70 to 80 cents, nearing the typical low-end purchase of a modern microtransaction-funded game.

These two implementations do also have a bunch of differences. When a customer walks in to an arcade, they know they are going to spend money on the games they are going to play, since arcade games explicitly inform them that they cost money to play. Modern microtransaction games are often free-to-play, meaning that getting to play them does not cost the player anything, but the money comes in after a while in the form of in-game items and services, which actually highlight another difference between the coin-operated and microtransaction-enabled games. While arcade games have to be designed so that the player gets hooked to the gameplay, potentially enjoying it, and wants to keep playing the game and trying again after failing or completing the game by inserting more coins into the machine. The design could be defined to wanting to make the player play longer. In several mobile games the aim of the design could be claimed to have purpose that is the opposite. For example in Clash of Clans, as described in later chapters, as well as in many other social and mobile games such as Sims Social many of the commodities available help the player avoid doing tedious tasks or waiting for a fabricated timer to finish. By logic, if the player is to encouraged to consume money on the microtransactions, the free option of playing the game has to be made tedious and undesirable enough for the paid option to be viable. The design could be hyperbolically summarized as making the player play the game as little as possible and favour the made-up benefits that save them time in real life.

2.2 Video game business models

Moving on to the present time and to its milieu of game development funding, the path to understanding microtransactions starts from understanding that video games often involve countless hours of design, programming and other productive work, and these tasks are being carried out by humans. Doing this amount of work without compensation in monetary or other form can be experienced as counterproductive for a person's survival and well-being in modern society. Long story short, video games cost money.

There are several ways of funding game development, and the options differ most distinctively in what state of the production the player is charged money, but also in the amount. Each of these methods have their own pros and cons. A recent trend has been to gather money by ”crowdfunding” a game. With this method the developers pitch the game idea not to potential publishers, but instead to the potential audience and possible players. Crowdfunded games are often either in very early development or still in design phase. The risk of the developer not breaking even is minimized in this funding method, as the production does not need to be started if the project does not meet the funding goal. Even though supporters are not charged before the project has met at least its initial goal, the risk involved in crowdfunding is shifted more towards the players. Even a fully backed project can fail to deliver the product that the backers have paid for, or the finished product may not be what they had expected.

A more traditional approach in project monetization is charging the customer once for the finished game, after which they get unlimited access to the game and can play it as they wish. This funding method may prove risky for the publisher, if no proper market research has been done prior to design and development, as having a game with unclear or too niche target audience can cause the publisher’s investments to not get the expected return and possibly even leave them with less funds that they started with. In deals where the publisher is separate from the developer, the designers and programmers are often financially safe even if the game fails to succeed, as their salaries are included in the budget that the investments have been spent on.

During the last few years, especially amongst independent (”indie”) developers, a business model has emerged where access to a game has been sold while the game is still in development. Players who purchase early access to a game can play and test the game, which is often only in alpha or beta phases of the development and is prone to containing bugs and errors. The final funding method is not dependent on early access to a game, and the game may be sold at full price after release or possibly made as free-to-play. Often developers offer a discount on the planned retail price for people who purchase the unfinished version of the game. Selling early access is a more reliable funding method to guarantee at least some income from the game for indie developers, as they can start gaining money with the game earlier and this way possibly get financial motivation for finishing the product. Offering a game for early access also allows the developer to map the interest for the game, make adjustments and even promote the product through word of mouth, unlike with selling a finished game, when the success and popularity may remain questionable until the release of the game.

Some games not only cost to develop, but also involve upkeep costs. These costs are most often related to multiplayer games that sustain a virtual world, and the biggest genre in this category are the massively multiplayer online role playing games, or MMORPGs. To compensate the cost of keeping up the online world by having several servers run constantly, these games often require the user to pay a monthly subscription fee to access the game. While the subscription model provides the developer with a steady cash flow, the recurring nature of the fee may repel players who do not have a stable income to spare.

2.3 Modern microtransactions

Originating mostly from the need to find a more affordable funding model for subscription-based games, developers began offering virtual items and services for microtransactions . Microtransactions (sometimes abbreviated mtx) are, as their name would suggest, small payments, and as a business model they often appear as purchases that the player can make from inside the game (in mobile games often referred to as "in-app purchases"). Unlike with the other business models, funding a game with microtransactions often happens after the game has been designed, developed, released and even played by the customer.

The financial risk in microtransaction-based games is greater than in other business models, as the budget has to be planned ahead far enough to cover running the game and possible multiplayer servers it needs long enough for the players to start generating income. A game that utilizes microtransactions is also often "f2p", free-to-play, in order to encourage players to get in the game and play it enough to consider spending money on the microtransactions and to lower the step to install and get in the game. The free-to-play model is used in conjunction with microtransactions so often that the terms are almost synonymous. The high risk involved in free-to-play games also affects the developers’ ability to secure investments for their project. The demand for high revenue potential is even more common despite the recent free-to-play successes such as Clash of Clans, as the CEO of german mobile developer Fishlabs Michael Schade mentions in his interview to PocketGamer.biz (PocketGamer.biz 2013). When he is asked about the difficulty of getting an investment. he mentions that ”If you don't have a game that has the potential to generate $1 million a day, it is a tough call”. This may be one of the reasons why some games have gone as far as to exploit human psychology and invoke gambling habits (Rose 2013). The link to gambling is discussed in detail later in chapter 3.

Careful attention must be paid while designing microtransaction-equipped games to what the game offers, as the items you can acquire with microtransactions have to be tempting enough and make a big enough impact to the gameplay experience for it to be worth the price, but at the same time has to be bound by the game balance in order for it not to trivialize the game content, causing the player to get bored of the game and stop playing and spending more money on the game.

3 The Nature of Microtransactions

3.1 Overview of commodities

As described in chapter 2.3, microtransactions are small payments, and in the specific context of this thesis they are microtransactions inside a smartphone application used to trade for virtual commodities. To simplify payments and guarantee larger amounts for single transactions, a middleman is added by making the virtual items available for purchase only with a virtual currency that can be gained more or less exclusively with real money. In many games the virtual currency is acquirable by in-game methods, but the available amount may be limited or the commodities offered vary by game and genre.

In social games, the common idea for virtual commodities appears to be in gaining an edge over others, as competitiveness is a key element in majority of them. For example, in King.com’s Candy Crush Saga, players are able to purchase items that grant them extra moves or extra time that they can use to keep on playing and attempt to salvage the situation if they find themselves on the verge of defeat in a puzzle. The game’s randomness guarantees that each attempt at a puzzle is different, and if the dice are in player’s favour they may feel like they do not want to waste the chance to show off their score to their friends by submitting to a defeat that they could have avoided with a booster. In addition to extra time or moves, the game offers boosters to further help the player gather points in the puzzles, potentially gaining an edge over other players that may not have the same boosters at hand. In 8 Pool by Miniclip players play pool against each other. Edge can be gained by purchasing boosts that increase the maximum power or maximum spin of the player’s shot, or one that grants an extended guideline that trivializes the estimation required for a shot. The boosts are not necessary, but can give the player a measurable advantage over their opponent. In addition to these boosts, the game offers a variety of cosmetic options for their cue and table available for in-game currencies. The most expensive cosmetic items allow the player to display a level of prestige and status, for example the most expensive cue in 8 Pool, the Diamond Cue, costs 2.5 million coins. They are obtainable through normal play, but if the player chooses to avoid playing the game for hundreds of hours, they can buy the required coins with real money, costing them about €3000. Cosmetic purchases are also available in Criminal Case by Pretty Simple in form of clothing, hairstyles and other accessories that the player can wear on their avatar. The biggest money sink in this crime solving game is the energy restoration items, which are fairly typical for Facebook games. The game has missions that the player has to complete in order to proceed. Attempting to complete a mission costs Energy, which is a resource that regenerates either slowly over time or consuming an Energy consumable. The same mechanic has been used in numerous social games, such as Mafia Wars, Sim City Social and the Sims Social.

The design of mobile games affects the nature of commodities that they offer. As social games tend to be designed around being connected and their mechanics sometimes revolve around utilizing or exploiting these connections, mobile games are more focused on offering short bursts of gameplay able to be played on the go. A common characteristic for mobile game microtransaction commodities is that they aim to save the player’s time, as is the case in for example Clash of Clans and Dungeon Keeper, described in depth in chapter 4. In these games the player can use Gems that they can purchase with real world money to instantly complete building, upgrading or training actions, saving them hours or even days. Some items may not directly skip a timer to complete an action, like in Hill Climb Racing where the player can buy coins with their money. Coins are the single currency in the game, and are acquired by playing the game normally. They are expended in unlocking levels and vehicles as well as vehicle upgrades, and the prices are high enough that making a coin purchase saves a lot of time gathering coins and allows much more customization in form of different vehicles for the player.

Games released and developed especially for PC are generally made to be played for longer sessions at a time, thus limiting the flow with long delays for crucial actions is not the best design for gameplay. PC games often offer greater complexity than their social and mobile counterparts, which is one of the reasons they keep players captivated longer. Having a longer relationship with the game increases the emotional attachment to the actual game as well as to the player's avatar. This bonding is essential for the profitability of cosmetic commodities offered directly or indirectly for microtransactions. Cosmetic commodities are most common in games where the developer has decided to have in-game items to purchase with real money in a way that does not affect the balance of the game. Notable examples are Path of Exile by Grinding Gear Games, where players can purchase new appearances for their equipment or skill effect, and Guild Wars 2, which offers also different appearances for the player's equipment as well as different animations that the player can display while they defeat other players in combat, in addition to other cosmetic upgrades available. Competitiveness and skill are key elements in many computer games, which is why offering boosts that enhance the player's performance may label a game of this nature as unfair, potentially repelling players who do not want to waste their time and effort to something that does not reward them for it without them having to spend more cash on the game. This is another reason to offer only cosmetic items as purchasable commodities, as has been done in for example Valve's DotA 2 and Counter Strike: Global Offensive, both of which feature a dedicated e-sports community.

The use of microtransactions is relatively uncommon in console gaming compared to other formats. The dominant business model used in these games is the retail, pay to play model where the player purchases the game in full price and receives the full game without requirements to pay more. One of the few free-to-play games on consoles is DUST 514, by the EVE Online developer CCP. The game has two currencies, one which can be only acquired with real money. Players can use the premium currency to purchase visual upgrades and item variants, instead of direct upgrades to their gear. Several games offer downloadable content (DLC) however, which may in some context count as microtransactions as the prices of DLC vary between single dollars to dozens. The scarcity of microtransactions and free-to-play games in the console market may be explained with the high risk of the business model against the high maintenance and development costs for the console (Peterson 2012). Another reason may be the limited nature of console controls compared to a mouse or a touch screen.

3.2 Emergent characteristics

Competition is a big part of the gameplay and the experience in some games, as briefly mentioned in previous chapters, and it is important to maintain the equilibrium between players mechanics-wise to give the game a chance to be fair and enjoyable for all players, especially if they are playing against each other. Games that fail to maintain that balance in order to favour purchased items and content tend to be shunned by a large portion of the gamer community, resulting the game to be labelled ”p2w”, or pay-to-win. An example of a pay-to-win game is APB: Reloaded, by Reloaded Productions, where players team up and fight against other players in an urban environment. One side works as law enforcement, and the other act as criminals. In the game, the player needs to first unlock and then purchase weapons with in-game currency. Instead of permanently gaining access to the weapon, they only get a 10-day lease on it. By spending real money, players are able to purchase permanent leases on the weapons, as well as different weapons that are not available for purchase with in-game currency. The ”premium” weapons often tend to be more powerful and effective in order to make them more desirable. The dilemma of premium weapons is that they need to be made powerful enough to be desirable and to give the player their money’s worth of content, but at the same time they need to be in balance with the rest of the game so that the premium items do not repel the players who are yet to make a purchase or only wish to play the game for free.

Analysis and discussion on the performance of in-app purchases has brought up a need for specific terminology. Because of the fact that unlike in the traditional retail business model, in free-to-play games the revenue per user is not constant, or even a certainty, the industry has adopted terms like Average Revenue Per User (or Unit, if the wishes to remove the last reference to humanity from their view of customers) – ARPU – and Average Revenue Per Paying User, similarly abbreviated ARPPU. The former, ARPU, represents the average amount of money the game generates per player if all players of the game paid the same amount. The measure may be attractive from a pure revenue point of view, but suffers from inaccuracy caused by the earlier mentioned uncertainty of payments per user. While no average or approximation display information about all the users in accurately from a personal point of view, Average Revenue per User tends to distort the reality up to the point where it is not the most optimal metric to designers who wish to improve the gained revenues. To this purpose an alternative iteration of user data will work better. Average Revenue per Paying User represents the average amount of money used by the players who purchase at least something something by microtransactions. This takes into account the uncertainty factor in the payments in free-to-play games and as generally the amount of players who do not pay is significant [src], this attribute gives a better view on how much the average amount of purchases in a game is. The amount of paying users is also a popular metric in the analysis of profitability of microtransaction-offering games, and it is communicated through ”conversion rate”. The scarcity of paying users in free-to-play games has been analyzed by SuperData, reporting the average conversion rate in the first quarter of 2014 at 5,0% in the US and 2.9% in China in their presentation ”US and China Mobile Games Markets Brief”.

As implied earlier in this chapter, there is a divide in principle amongst players of free-to-play or other games offering microtransactions. The chasm that sets the two major categories of players apart is the willingness to pay and make a purchase of a game's commodities. The gap can be illustrated by further analysing numbers given for the first quarter of 2014 provided by SuperData in their US and China Mobile Game Markets Brief (Superdata 2014a). The analysis shows that the average revenue per paying user in the US is $21,60 and $32,46 in China. Comparing to the previous numbers we can deduce that the average revenue per user for these markets revolve around the amount of one dollar, $1,08 and $0,94 respectively. Not only do the numbers elaborate on the gap between paying and non-paying users, it also implies cultural differences between the east and west. Not only does the categorisation happen between paying and non-paying users, it also happens within the paying userbase. Free-to-play games attract behaviour that is also known in the casino business, where a player is willing to invest substantially larger amount of funds into the game than what is the average. These people tend to be labelled in discussion as ”whales” - a nickname familiar with the high rollers of casinos - but what differentiates the people who spend a significant amount of money on video game from a gambling addict is that these game ”hobbyists” tend to purchase considering a longer term investment instead of just being impulse-driven (Yee 2014).

Gambling references do not only occur in labeling benefactory players, but also deeper in the actual design of some games. Because developing games costs money, and free-to-play games are - of course - free to start playing, developers and especially designers need to pay extra effort in creating design that guarantees the return of investment for the shareholders of the project. This objective is made easier to reach by falling back on old and tested methods, such as the ones casinos operate on. While not all games are not explicitly comparable to the one-armed bandits, some of the more sinister social and mobile games base their mechanics on utilizing the same psychological perks (Tseng 2011). This currently poses a dilemma between financing a free-to-play game and making a quality product that is enjoyable by players without them having to keep their wallet open for the whole play session. The problem can be traced to pitching, as quality and enjoyment are abstract concepts that cannot be put into numbers that convince people with more money and less knowledge on games.

3.3 Maximizing the profit

As already established in the previous chapters, games cost money and they need to be funded somehow. Often the funding comes from investors, who promise money in exchange for more money later on. This is probably one of the biggest reasons that has lead to free-to-play game design in some games becoming increasingly exploitative, borrowing techniques from the ill-famed casino world. Social media has enabled a design that focuses on reaching as much people as possible and attempt to addict them to a game, later directing their addiction into converting them into paying customers. Gabe Zichermann lists four key points of this design, called gamification, in his 2012 post on Mashable. The first of his points is that the player should feel like the game is free, and instead of prohibiting actions the game should make the player wait. This way the decision to use the game’s option to pay money to skip the wait will seem more valuable in terms of how much it saves them time. The second of these points is that the game should not limit how much someone can pay, a note which concurs with the earlier described ”whale” mentality apparent in free-to-play games. As third point, Zichermann brings up that the items available for sale should not just be put out there, but their placement needs to be determined by analyzing and studying the players’ behaviour. To offer players smaller and more variable tasks helps to keep them engaged in the game for longer. The last point is that the game always needs more players. This is a fact that has been acknowledged in especially social games, where the game will attempt to notify the player’s friends about what they have done as much as possible, in hopes of convincing some of them to join the game as well and share the experience. (Zichermann 2012.) The list of design perks utilized in free-to-play games does not end with Zichermann’s, and there are several available analyses on the internet, each more or less authentic or credible.

Keeping players playing the game is, an aspect that has been studied heavily in games involving microtransactions, and especially in social games. Many people have presented critique towards the design of these games, and it has even been compared to behavioural psychology experiments. This criticism stems from the characteristic features that are used to maximize the amount of money that can be harvested with microtransactions from the users.

Independent game designer Jonathan Blow underlines some of these practices in his speech at CreativeMornings Portland (2013), listing them in his slides: ”Ensnare the player for as long as possible (build an infinite treadmill)”, meaning that the game does not have an ending, and might even have procedurally generated content to keep the player playing. ”Interrupt the player's life as often as possible” is often the case with mobile games and their push notifications, begging the player to come back to the game or reminding them of pending actions. ”Train players to spend your fake currency” describes the state of many microtransaction-equipped games’ introductions and tutorials, where the players are given some of the premium currency and forced to spend it. ”Get people to bug their friends”, as many social games do, trading premium currency or other benefits inside the game for friend invites. ”Make game about waiting; let player pay not to wait” is a frequently used principle in several mobile and social games. ”Give a reward, threaten to take it away”, which Mr. Blow describes as exploiting the psychological effect called ”loss aversion”, meaning that people often find it a greater loss if they lose something they have already acquired, instead of never having possessed it in the first place. As a conclusion from these points Blow presents, the direction of free-to-play game design does not necessarily bode well for the customers. (Blow 2013.) While it cannot be denied that the players, or customers, can have fun in the microtransaction-based games, it may be worth questioning the morality and ethics of the methods the entertainment is being delivered and how it is being used to maximize investor profits.

Since many free-to-play games are designed to include strong exploits of the human psychological systems, special care has to be paid when targeting children who play the games. The issue has been identified at least in the United Kingdom, where in January 2013 the Office of Fair Trading (OFT) released principles addressing the issues emerging with aggressive marketing and children playing the games. On their web site, the Office of Fair Trading list the concerns addressed in the principles:

> • a lack of transparent, accurate and clear up-front information relating, for example, to costs, and other information material to a consumer’s decision about whether to play, download or sign up to a game
> • misleading commercial practices, including failing to differentiate clearly between commercial messages and gameplay
> • exploiting children’s inexperience, vulnerability and credulity, including by aggressive commercial practices
> • including direct exhortations to children to buy advertised products or persuade their parents or other adults to buy advertised products for them
> • payments taken from account holders without their knowledge, express authorisation or informed consent.

The issues listed are not exclusive to just children playing the games. The principles give guidelines targeting existing methods for marketing and selling virtual items or currency in games. The common idea behind most of the principles is that it is the developer’s responsibility to make sure the player, and their parents, are aware of what they are signing up for when they download the game and what their personal information are being used for. The developer has to make sure that they have the player’s full consent when taking payment from them, and the player has to be able to identify messages and actions that have a commercial intent. All the 8 principles can be read in full description from the document released by the OFT, found on their web site. (Office of Fair Trading 2014.)

3.4 Success in practice

Wargaming’s World of Tanks, which is one of the top grossing free-to-play games (SuperData 2014b). World of Tanks allows the player to unlock tanks with XP points, the XP cost to unlock a tank rises significantly after the early tiers while the XP income stays somewhat same. Players are able to use their premium currency to convert the tank-wise regular XP into ”Free XP” which can be used for all tanks. The XP conversion in World of Tanks is one of the most significant real money sinks in the game (Weidemann 2014). In principle the conversion mechanic allows the player to use the resources they have gathered with their effort spent on an action that is not directly related in overcoming the requirements to unlock a specific tank. The concept of this may be an integral part in the success of World of Tanks, as the premium currency does not directly give the players resources but rather amplifies their rewards, nor will it give them an explicit edge over others, the game avoids being labelled pay-to-win yet still retains the value of its microtransaction items. A deal that is good both for the developer and the players.

Amongst the success stories such as Angry Birds and Clash of Clans, one could draw a conclusion that free to play is the absolutely best business model for a mobile game. The data studied in this thesis may have its say against that conclusion, low average conversion rates and the fact that majority of the income comes from a very small group of players combined with recent data that 19% of the players never open the game again after launching it the first time and 34% of the players keep on playing after the first 24 hour period (Swrve 2014). These numbers tell that even though free-to-play games reach significantly more users than the pay-to-play games, they highly ineffective in offering the players content that they feel like playing. In his 2014 opinion post on Polygon, The Room 2 developer Barry Meade describes that the problem does not reside necessarily within the business model itself, but the mentality that it has made money and thus will make money again upon repeating the steps. Concerning the future of the games industry, Meade claims that ”A fertile ecosystem needs lots of green shoots as well as the old redwoods”, a metaphor regarding the endless clones of successful games, easily noticable upon browsing a mobile application marketplace, all in hopes of recreating the success that their idol had. (Meade 2014.) For the health of the industry, it is incredibly important to keep on innovating, of course while keeping in mind the ”old redwoods” that made the industry what it is today.

4 The Manifestation of Microtransactions

4.1 Case: Clash of Clans

This chapter will take a look Supercell’s Clash of Clans and examine what the game is about, what is the context of microtransactions in it and how they are placed. The game is also analyzed with UK Office of Fair Trading’s guidelines for in-app purchases in mobile games.

4.1.1 The idea of the game

Clash of Clans is a mobile game produced by the Finnish game studio Supercell released in 2012 for the Apple iOS and in 2013 for the Android platform. In Clash of Clans players are given control of a town that they use to produce an army of units that have different functions and abilities. Players start with being able to produce only a small number of one kind of units, but are able to grow their army in size and variety by building different kinds of buildings.

Buildings are built by consuming either Gold or Elixir and occupying one Builder while the building is in construction, and take a set amount of time to complete. Already built buildings can be upgraded to strengthen their effect. If all Builders are already in use, no buildings can be built or upgraded. Building time increases exponentially as buildings get more effective, for example building the first level Gold Mine takes one minute, whereas upgrading it to level 4 takes one hour.

The objective in Clash of Clans is to certain extent defined by the player. The game presents the player with a list of single player challenges that increase in difficulty. Tougher challenges force the player to improve their town and acquire more powerful units in order to overcome the predefined single-player challenges.

Another objective to aim for in the game is the multi-player Leagues, meaning that instead of trying to conquer set challenges made by the developers, the players acquire "Trophies" by challenging and defeating towns built by other players. The player will be assigned a "League" based on the number of their trophies. Whenever a multi-player battle is concluded, the victor is awarded Trophies and the loser will lose Trophies, the amount gained and lost is calculated based on the estimate of the difficulty of the battle, taking in account both players' Leagues.

4.1.2 Microtransactions

The purpose of Microtransactions in the game is mostly to skip lengthy waits, usually while building or upgrading a building or unit. Microtransactions are used to acquire a special currency, and utilizing the game’s time-saving features use up this currency. The cost to skip a wait goes higher as the wait time increases, but when the build times are in several days, the temptation to pay the price may overcome many impatient players.

When the player starts the game for the first time, they are introduced to the game’s features and mechanics by a tutorial sequence. The integration of microtransactions is the most visible during this introductory phase, when the game figuratively holds the player’s hand in teaching them to use the microtransaction currency. Players are not explicitly forced to purchase the currency, as they are handed a moderate amount of it upon installing and signing up for the game. Over half of this amount is however used during the tutorial phase in an attempt to teach and encourage the player to use the microtransaction currency, lowering the barrier to consume it also later in the game. To reinforce this, wait times and the cost to skip them are relatively low in the early game, so that it is easier for the player to get accustomed to spending the leftover currency from the tutorial, and potentially purchase more in order to keep the pace of the game constant. Progressing in the game causes the buildings and upgrades to take longer to complete, and eventually the player will run out of activities to perform on that play session, and will be forced to make the choice between spending more currency to keep playing or ending the session and suppressing the possible ”flow” state they may be in. The competitiveness of the game encourages the player to not wait too long, as waiting can cause their competitors get ahead of them in unit upgrades and subsequently in trophies and rankings.

In addition to gaining an edge over the competitors by skipping long wait times, microtransactions can help the player fill out missing resources. In principle this is also about eliminating wait times, as resources can be gained over time. Alternative to the instant pay-off of missing resources, players can also choose to activate a ”Booster” for their resource-generating buildings, which makes them more efficient for a short period. Boosting a resource building is less expensive currency-wise than filling missing resources, but offers the player a strategic choice of to save their currency and spend more of their time instead. For the developer’s business the player’s choice does not matter, either scenario is beneficial; instantly gaining resources consumes more of the player’s currency but the longer-lasting Boosters tie the player to the game for a longer time.

Another usage for the microtransactions is in building an army: the training queues of the unit producing buildings can be either sped up with a ”Booster” similar to the resource buildings, but also by instantly completing the queue. This usage feeds from the competitive nature of the game, as the player is able to challenge and conquer more human players when they gain their units faster, allowing them to gain Trophies and rising in ranks and leagues faster.

4.1.3 Acquisition

Microtransactions manifest themselves in Clash of Clans in the form of Gems, a currency acquirable by only a few ways:

- Players are given a moderate amount of Gems upon first installing and signing up for the game.
- There is a random chance to get a minimal amount of Gems from performing a short action on shrubbery, rocks and other obstacles that appear in the player’s town
- Completing achievements that require the player to perform a set amount of actions or gather a specific amount of resources award the player with a small amount of Gems.
- Exchanging real money for Gems is the fastest and most reliable way to get larger amounts of gems.

Compared to buying the Gems, the other methods are unable yield enough Gems for the player to use them regularly. The initial handout during the tutorial is 500 Gems. Clearing out obstacles are easily repeatable actions that give the player anything between 1 and 3 Gems, and completing achievements happens on a rarer occasion but award the player usually with 2 to 20 gems except for the late-game achievements, that may give a one-time award of up to 2000 Gems (gained by reaching the Champion league). League success plays even more important role in Gem acquisition, since the top 3 Clans will receive a substantial amount of Gems for the top 10 players in those Clans. Gems awarded for each Clan are divided equally between the highest performing players. The highest ranking Clan gains 20 000 Gems, the second is awarded 10 000 and the third gets 6 000 Gems. For each of the top 10 players, this means 2 000, 1 000 and 600 Gems respectively for each.

4.1.4 Placement

While microtransactions themselves are confined in one specific menu, abstracting the money into Gems allows buttons that consume them to be placed in easily accessible locations. Most often a Gem-consuming button, or a Gem-button, is placed in the context of its effect, for example Gold, Elixir and Unit training Boosters are available for purchase when you select their respective building. Task-finishing buttons are often placed next to the upgrade’s or building site’s Cancel-button, giving the player a chance to consider their options and decide whether they should waste their resources (cancelling an upgrade or a building site will only return half of the resources used to initiate the operation), or spend Gems to finish the progress.

The actual purchases for Gems are as many clicks away in the menus as buildings. Gems are offered in bundles in predefined amounts. As listed in Table 1, the gems gained per money spent does not remain constant. This pricing model is implemented in order to encourage players to purchase more Gems at a time, give better value for bigger investments and balance the increase in Gem requirements for actions later in the game.

Table 1 - Amounts and prices of the Gem bundles in Clash of Clans

Amount	Price (€)	Gems per €	Relative increase in value
500	4,49	111,36	-
1 200	8,99	133,48	19,87 %
2 500	17,99	138,97	4,11 %
6 500	44,99	144,48	
14 000	89,99	155,57	7,68 %

4.1.5 Morality

Objective perspective to the game’s morality and responsibility can be gained through evaluating the game’s features with the UK Office of Fair Trading principles for online and app-based games. There are 8 principles concerning the developer’s responsibility to make the user aware that they are being sold services.

The first and second principles concern the clarity of information regarding costs that are associated with the game, and the information regarding the game’s functions and content such as its short description, hardware and software limitations and its social functionality. Costs ought to be broken down to specify the initial purchase price of the game, the possible subscription costs or mandatory payments that are required to keep on playing the game and the possible microtransactions that the game offers. The first principle also dictates that the amounts, or at least the range of the amounts, has to be present with the aforementioned details. All this information needs to be presented before the player begins to download or sign up for the game, meaning that the player must have this information at hand while they are making the decision to download or purchase the game.

The Clash of Clans Google Play store page includes includes a basic description of the game and it’s functionality. It also describes clearly that the game is social and that the player is likely to encounter human interaction within the game. There is also a note, informing the potential customer that the game is free to play, and does not require any subscriptions, but contains items that can be purchased for real money. (Supercell 2014.) If players do not want to use those features, the developers encourage them to turn off the in-app purchase option from their device. However, the store page fails to mention the exact amounts of the purchases, even though they are pre-defined easily available to view from inside the game. Due to this, the game is less likely to comply with the first principle of the guidelines presented by the UK Office of Fair Trading.

The third principle dictates that the information to contact the seller or developer of the game has to be clearly presented before acquiring the game or creating an account for the game, and that the information must be based on a reliable means of communication. Clash of Clans includes two e-mail addresses on their Google Play store page, one for suggestions and feedback and another for other queries. The principle also expects that the seller is able to be contacted rapidly and directly. This depends on the development team’s customer service capabilities, and cannot at this time and in this case be confirmed. Further contact information is only listed on the developer’s site, but the addresses on the store page should suffice to comply with this principle.

The fourth and fifth principles discourage ambiguity of commercial intents. These principles describe that all features involving real money have to be easily distinguishable from the actual gameplay, and that the player has to be aware that the actions they are about to invoke may involve or require utilizing microtransactions. The player also must not be mislead in to believing that microtransactions are a required part of the game where they are actually not. Direct purchases are strongly discouraged, and they should instead always point to a dedicated shop window, that can be recognized as a separate part from the gameplay. Clash of Clans may not fully comply with these guidelines. The game does force the player to go to the store when they are short of gems for an action, the wait times for different actions are displayed clearly and all screens where the usage of gems is a possibility there are also buttons to exit the screen. However the concern and a target for closer inspection is the distinguishability of the microtransactions from the core gameplay, as the game specially teaches the player to use the currency during the tutorial, and the wait times get so long down the line that the only logical thing to do to keep on playing might seem like purchasing with the microtransactions. The game does not really tell the player to do so, but the player may feel obligated to do so.

The sixth and seventh principles regard marketing towards children, prompting the developers to not exploit children’s inexperience and credulity that their young age includes. The seventh principle especially discourages direct exhortations to children, as they may feel forced to make a purchase, or ask and persuade someone else to do it for them. To achieve this, the game has to present the free and the paid options equally, and not provide a straight link to activate the microtransactions but rather encourage the player to purchase the required items from the shop. The game should also avoid using imperative language, as children may feel that they are forced to make a purchase to keep on playing. Clash of Clans mostly complies with these two guidelines, as the language used in the game is not in demanding form, but rather suggesting or asking. For example if you do not have enough Gems to finish a construction, the game throws you a popup titled ”Not enough Gems”, informing the player of their lack of currency, and then says ”Do you want to get more?” and gives the player a button to enter the shop to purchase the Gems.

The eighth and final principle is defined to ensure that when a payment is taken from the player unless they have given their consent, being aware of the consequences of the purchase. The developers must ensure that the player knows when they are being billed for their actions, and that they explicitly acknowledge that they are going to pay the given amount. In the case of Clash of Clans, an easing fact is that all payments are done through the designated store window. On Android, selecting a Gem bundle from the store window opens the Google Play store purchase confirmation dialog box, that has a clearly labeled button that says ”Buy”, and represents the price clearly right above it. This complies with the given guidelines, as the buyer should be aware that they are being charged when they get to the screen to perform the purchase.

In conclusion, Clash of Clans seems to be somewhat responsibly designed and mostly in compliance of the UK Office of Fair Trading principles for online and app-based games. It is important that these kinds of guidelines are being drawn and that they are followed, as overly commercializing social and mobile games does not promise a bright future for the gamers, when they are more and more being used as figurative money cows that are being tapped into and milked with games.

4.2 Case: Dungeon Keeper

This chapter will examine the recently released Dungeon Keeper published by EA and study its game mechanics and how they work with the microtransactions that are included in the game. The game is also put under analysis and is compared to the guidelines released by the UK Office of Fair Trading.

4.2.1 The idea of the game

Dungeon Keeper is a game released in 2013 for the Android and iOS mobile platforms, developed by Mythic Entertainment and released by Electronic Arts (not to be confused with Dungeon Keeper, the PC strategy game released in 1997 developed by Bullfrog Productions and also released by Electronic Arts). The developers describe the game on the Google Play store page as ”It’s tower defense…without the tower…and a lot more offensive!”, possibly referring to the tower-defense-like nature of the earlier games in the Dungeon Keeper franchise (EA Mobile 2014). The game’s reception was mostly negative, as of April 2014 its rating on the rating aggregate site metacritic is 42 out of 100, describing responses as ”Generally unfavorable reviews” (metacritic 2014b). Most of the criticism is aimed towards the game’s heavy use of microtransactions, so much so that even the original franchise creator Peter Molyneux had shunned the game for them (Handrahan 2014), which is why it makes a valid case for analysis for this thesis.

The gameplay of Dungeon Keeper revolves around the player managing their own dungeon, much like the player manages their town in Clash of Clans. The dungeon consists of rooms connected with corridors and traps that can be laid down on the corridors. The player can summon units which they can use in raids against other players’ dungeons. The center of the dungeon is a room called ”Dungeon Heart”, which works as a place to recruit units and it determines the maximum level of other rooms in the dungeon.

Rooms and units can be purchased with Gold or Stone, the two main resources used in Dungeon Keeper. Stone is acquired by claiming Stone Quarries which slowly generate Stone, and they can be harvested by tapping them when they have gathered some Stone. Gold can be acquired by claiming Gold Mines, which work the same way as Stone Quarries. Both Gold Mines and Stone Quarries are pre-placed on the dungeon map, each containing total of 4 both resource buildings. In addition to generating resources, the Gold Mines and Stone Quarries also work as entry points for enemy units during raids on the player’s dungeon.

Units can be used to conduct raids on other players’ dungeons. Much like in Clash of Clans, whenever units are used in raids they are expended and can not be used again. In addition to units, player can use Spells to their advantage during raids. Spells have various effects, and they are generally more powerful than units and use Mana whenever they are cast. Mana can be acquired by building a Dark Library, which works like Stone Quarries or Gold Mines, generating Mana over time, and is gathered by tapping the Dark Library when it has generated some Mana. Raids reward the player with Gold and Stone as well as Trophies and Attack Points. The latter two are used to determine the player’s rank on the game’s global leaderboards.

4.2.2 Microtransactions

The real-money sink for Dungeon Keeper is its premium in-game currency, Gems. The Gems in this game work fairly similar to how they work in the previously analyzed Clash of Clans, at least as far as their usage goes. The main commodities that can be exchanged for Gems are time, in-game resources and different ”Boosts”.

The gameplay in Dungeon Keeper, especially in later stages, involves a lot of waiting in the form of building, training and mining times. Waiting times are troublesome for the player due to their tendency to break the play session flow, which can in some cases mean that the player becomes annoyed or frustrated because they have had good time playing the game for several minutes and now need to put the game down despite their desire to keep on playing due to the timer delaying their actions in-game. This, intended by design, promotes the option to use the game’s premium currency to skip these waiting times in order to keep on playing.

Resources to build and upgrade buildings can be gathered from resource buildings that generate them slowly over time. The buying power of the player depending solely on the income from the resource buildings diminishes quickly, as costs to upgrade buildings increase exponentially, whereas the extra income from resource building upgrades increases in linear fashion. To gather resources required to upgrade the central building, the Dungeon Heart which limits the progression of everything else, to maximum level will take the player around 103 continuous days having only the resource buildings as an income. To compensate for the radically increasing costs the players can purchase resource fills with Gems. Resources can also be gained by launching attacks with your own troops – raiding – other players' dungeons or completing raid "Events". The amount of resources that can be gained through raiding is limited by the level of the Dungeon Heart, which allows for some scaling towards the higher costs of late-game upgrades, but also prevents the Gem purchases from becoming redundant.

One of the more significant commodities the player can purchase with Gems are the ”Imps”. They are similar to Clash of Clan’s Workers, as they limit the player’s concurrent actions. The player can get around this limit by either waiting for their actions to finish, using Gems to finish an action in progress or purchasing another Imp, which provides another slot for an action. The amount of Imps is limited to six, and the cost of hiring them increases exponentially with each, the advantage being that once you purchase them, the Imps are permanently enabled on your account.

To temporarily boost the player’s defensive or offensive abilities or to increase their resource gain the game offers a variety of Boosts. These Boosts can be purchased with Gems, and they are in effect for a predefined duration that starts upon their activation. There are 5 different Boosts: ”Cave-in”, which prevents other players’ raids on your dungeon but is discarded if the player decides to raid someone else’s dungeon. ”First Strike” increases Minions’ health, damage and movement speed as well as Spell damage by 10% while attacking. ”Fortification” Boost increases Trap, Room and Minion health and damage while defending your dungeon. ”Scouting” is a Boost that reveals hidden Traps during raids. As the fifth Boost there is the DK Premium, which is comparatively longer Boost that increases resource production, Raid rewards and gained Gem rewards from Exclusive Raids.

When the player starts Dungeon Keeper for the first time, they are taken through an introductory tutorial for the game’s mechanics. In addition to teaching the basic gameplay, the tutorial seemingly attempts to teach the player to be liberal at consuming the premium currency they are given upon signing up for the game. There are several parts during the tutorial when the player is told to spend Gems, and in case of introducing the feature to skip a wait time during construction or unit training the prompt can be ignored by just waiting until the task is completed, but for example in telling the player about the Imps and how they are required for different tasks the player is forced to use 100 Gems to perform the purchase action in order to proceed in the tutorial. The amount of Gems required for the second Imp and skipping waits are significantly lower during the tutorial than what they are in the actual game, which may lead the player to a sense of false security about the amount of Gems they possess and the amount of buying power they actually provide. If the player gets used to skipping all waits during the tutorial, nearly all of them being under one minute, for 1 Gem and they continue the behaviour after the tutorial is done, they may quickly burn through the initial Gem reserve and be faced with a decision to buy more Gems to keep up the freshly learned playstyle.

4.2.3 Acquisition

Gems in Dungeon Keeper can be acquired in several different ways. The easiest and most time-efficient way of getting Gems are the Gem bundles, which are available for real world money, but the game also rewards the player with Gems for completing achievements or doing certain tasks. Achievements usually reward the player with 5 gems per completed achievement, the longer term achievements give out 10–50 gems and the multiplayer league related achievements return 10–2250 gems. The league achievements are awarded for people ranking first in their current leagues, which may make using Gems to climb the ladder tempting, seeing that beating other players gives back a substantial amount of Gems.

Another cost-free way of gaining gems are the in-game tasks. The player’s dungeon consists of rooms and corridors carved in rock. There are three different types of rocks, Soft Rock, Gem Vein and Hard Gem Vein. The Soft Rock is the fastest rock type to mine, but they only have a small chance of containing a small number of gems. The Gem Vein and Hard Gem Vein take longer time to mine, 4 hours and 24 hours respectively without a building time boost, but they also yield the most gems when mined. Gem Vein contains 2–3 Gems and Hard Gem Vein 9–10 Gems, significantly more than the Soft Rock on average. To get any profit from the Gem Veins, the player has to wait for the mining time to finish as skipping the Gem Vein’s 4 hour build time would cost 49 Gems, resulting in 46–47 Gem loss for the player. In addition to mining the rocks in the dungeon, the player can take part in ”Exclusive Raids”, which are like regular Raids against a computer opponent, except that the player is defending in their own dungeon and the opponent has 10 waves of monsters to attack with. If the player can defend for all of the 10 waves without their Dungeon Heart destroying, they are awarded 46 Gems. Retrying the Raid will cost the player 30 Gems, but they have another shot at the 46 Gem reward.

The Gems are sold in bundles, priced between 4,49 € for 500 gems and 89,99 € for 14000 gems, as described in detail in table 2. The price points are identical to what Clash of Clans has, as shown in table 1, with the exception of the second cheapest bundle. Despite it possibly seeming like a minor difference, it may have an impact on the player's decision to spend just a couple of dollars more than they originally intended by having a better increase in value for lower amount of money. The increase in value compared to the one step cheaper item can be seen visualized in figure 1. Once the player's threshold to purchase bigger amounts of gems has been lowered in this manner, there may be a greater chance that they make similar decisions in the future.

Table 2 - The sizes and prices of the Gem bundles in Dungeon Keeper

Amount	Price (€)	Gems per €	Relative increase in value
500	4,49	111,36	0
900	6,99	128,76	15,62 %
1200	8,99	133,48	3,67 %
2500	17,99	138,97	4,11 %
6500	44,99	144,48	3,97 %
14000	89,99	155,57	7,68 %

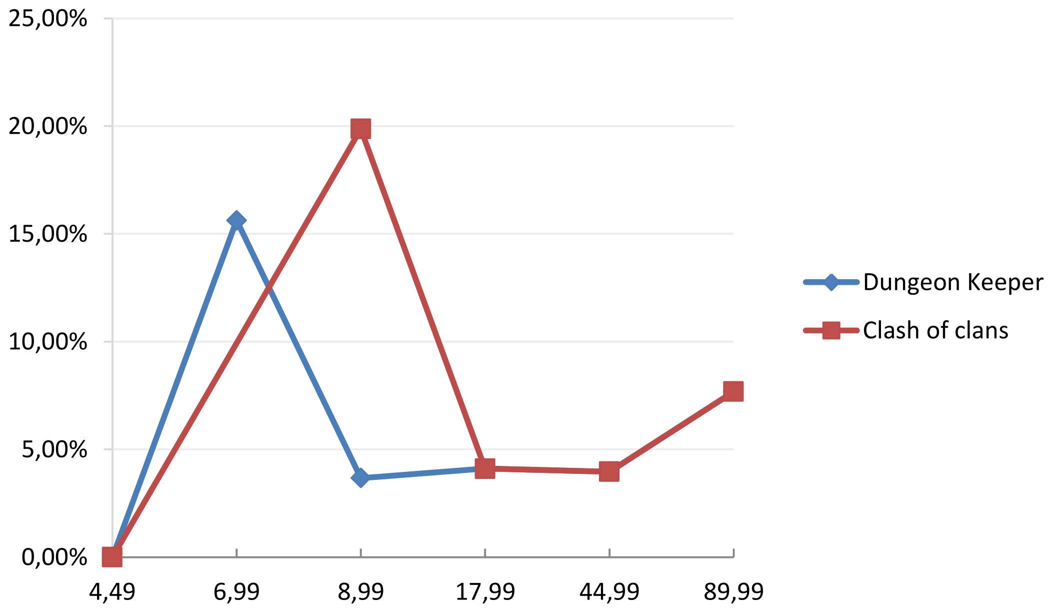


Figure 1 – The increase in value of a gem bundle compared to the previous price point

4.2.4 Placement

The placement of microtransactions in Dungeon Keeper is very similar to what it is in Clash of Clans: the actual real money purchases are available from a single menu screen and the buttons that consume gems are located in the specific context where their actions are needed. Boosters are an exception, as they can be bought from a separate screen to be used later as well as before starting a raid.

One of the differences between the placement and advertising between Dungeon Keeper and Clash of Clans is that Dungeon Keeper tends to advertise and pressure the player more to use the gems. As mentioned earlier, the buttons to expend gems are strongly highlighted during the tutorial, and the player may feel that they can not progress without consuming some of their starting gems.

4.2.5 Morality

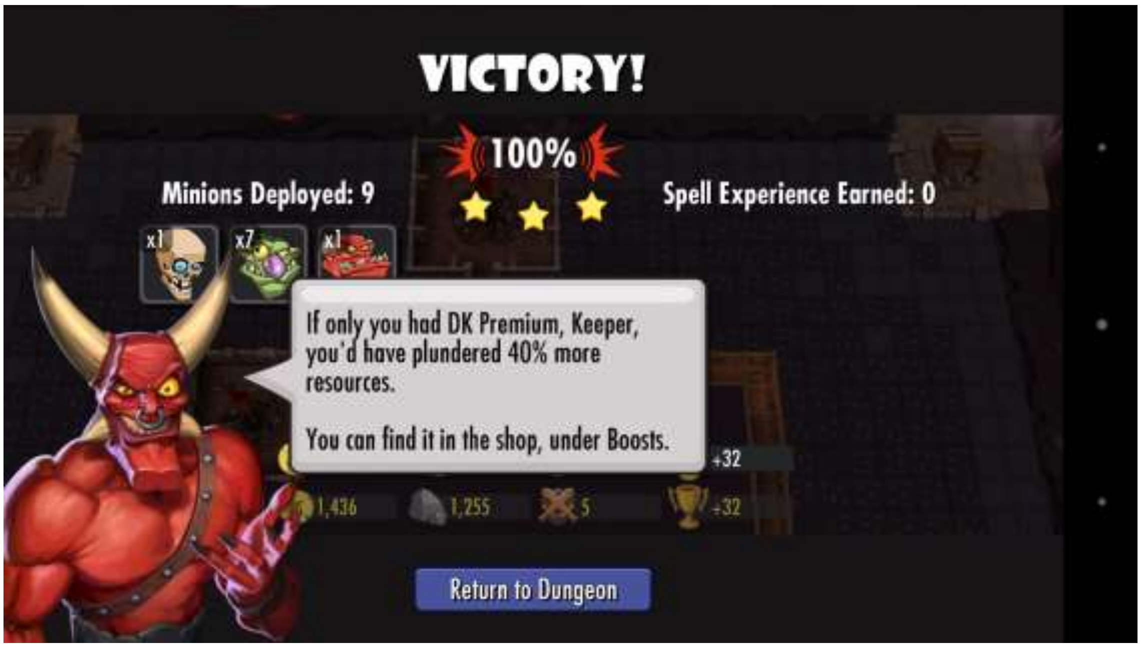
Like in the previous case with Clash of Clans, Dungeon Keeper’s morality can be judged by comparing it to the guidelines issued by United Kingdom Office of Fair Trading. The guidelines include 8 principles regarding the presentation and advertising of in-app purchases.

The Dungeon Keeper Google Play store page provides information on the cost for signing up for the game, which is none, as well as a mention on the availability of microtransactions, as suggested by the first principle of the guidelines. The page fails to break down the optional costs of purchasing gems from inside the game however, as also mentioned in the principle. The advantage of having the information on gem bundles available, without the need to install the game, is that the guardians of younger players, the people who pay the bills, can plan and advise the player on the use of those in-app purchases. While the breakdown of in-app purchases is missing, these details are likely to comply with the guidelines.

The store page for Dungeon Keeper describes in sufficient detail the game’s functionality. It also mentions the ability to group with other people and the possibility of the player’s dungeon being invaded by other players. The second principle expects the developer to disclose information about the game’s characteristics so that the customer is able to make a decision of buying or signing up for the game with enough information about it. In addition to information about the game, the game’s description addresses the use of the player’s personal data that is being gathered and the inclusion of advertising for the developer’s or its partners products. This amount of information should be sufficient to comply with the given guidelines.

The third principle outlines that the customer should be informed of ways to contact the developer or the publisher before they purchase or start to play the game. The Google Play store page for Dungeon Keeper includes an e-mail address to direct queries about the game. The page also lists several social media sites related to the game, but a lack of mention may mean that they are not used by the support personnel and can not be used for support requests or complaints. The functionality of the support e-mail address can not be verified within the timeframe of this analysis. Despite the scarcity of contact information, the game is likely to comply with this principle as it meets the minimum requirements.

As described in the fourth principle, commercial intent of any advertisement must be easily told apart from the actual gameplay. The fifth and sixth principles prohibit misleading the consumer into thinking payments are required or integral part of the game, and exploiting young players’ naivety in order to pressure them to make a purchase, respectively. To the same category belongs the principle seven, which advises a game to not include imperative wording that a child may consider as an order to purchase something. While Dungeon Keeper does not advertise products or services unrelated to the game, some of the game related advertising may be confusing for younger players. As an example, some time after installing the player is greeted with a popup upon launching the game, promoting a Dungeon Keeper related ringtone. The wording of the popup is in an imperative and almost taunting tone, reading ”Enjoying Dungeon Keeper? Install the exclusive Dungeon Keeper Ringtone Rap onto your device now. Go on, it’s FREE!”. Out of the four buttons below the text, only one makes the popup go permanently away. While it does not promote paid content, the wording gives an example of the relative aggressiveness of advertising in the game, potentially veiled under the personality of the support character that is displayed saying them. Another example of this is when the player completes one of their first raids, and they are congratulated with a message saying ”If only you had DK Premium, Keeper, you’d have plundered 40% more resources. You can find it in the shop, under Boosts" (figure 2). While the message does not strictly order the player to purchase the Premium Boost, it does strongly suggest so by again nearly taunting the player. While the language and messages may not be strictly in violation of the principle due to the theme of the game being "evil" and because the payments are hidden beyond the Gem currency, careful consideration should be done while deciding on the compatibility with the guidelines.



If only you had DK Premium, Keeper, you'd have plundered 40% more resources.

You can find it in the shop, under Boosts.

Figure 2 – Dungeon Keeper's congratulatory message received upon first victory – Screen capture from the game Dungeon Keeper by EA

The final principle implies that a payment should not be made without the account holder complying to it. Dungeon Keeper on Android does not directly have control over this, but the Google Play Store offers a setting that requires password for purchases made from the Store. The setting can be set to require authentication for all or no purchases, or to ask the password every 30 minutes. This offers a possibility for the account holder to be aware of all purchases unless their password has been compromised, complying with the guidelines.

4.3 Case-to-case observations

Prior to writing this thesis, the great likelihood between Clash of Clans and Dungeon Keeper was not completely obvious to the writer. During research it did become clear that these games are both made with the same formula, but what most differentiates them is their success and reception. The gap of these two games can be seen in their metacritic ratings, where Clash of Clans has average critic score of 74 and user score of 7,0 (metacritic 2014a) while Dungeon Keeper has generally negative reviews with critic score of 42 and user score of 0,4 (metacritic 2014b).

The reception of Clash of Clans was relatively neutral and quiet compared to that of Dungeon Keeper. According to Google Trends , as seen in figure 3, Clash of Clans gained its significant search popularity boosts long after the App Store launch. Dungeon Keeper on the other hand bases itself on a well known franchise named the same, and is thus already placed under heavy scrutiny even before its release. Despite its reference to a well known and established series, the game did not gain significant search popularity boost after its announcement, as seen in figure 4 where marker D signifies the approximate time of announcement and marker C the launch for limited audiences. The game was properly released in December 2013. The series has had a long pause in proper installation releases, last one dating back to June 1999, which creates even more anticipation amongst fans of the Dungeon Keeper franchise. The high fan anticipation and the fact that Dungeon Keeper does not seem to be aimed towards those loyal fans but rather towards the mobile free-to-play audience are part of a recipe that probably caused the outstandingly negative reception for the game (Levy 2014). Because of their similarity in the microtransactions and baseline gameplay, one can only draw conclusion that the failure of Dungeon Keeper is mostly because of the fans not getting what they wanted, and the developer taking a well known title and transforming it into a completely different experience.

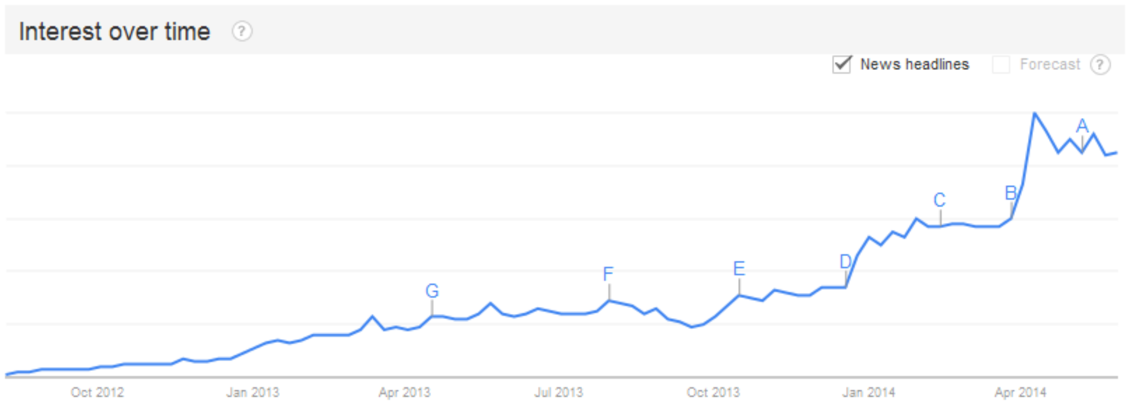


Figure 3 – Clash of Clans search popularity over time. Screen capture from www.google.com, retrieved 1.6.2014.

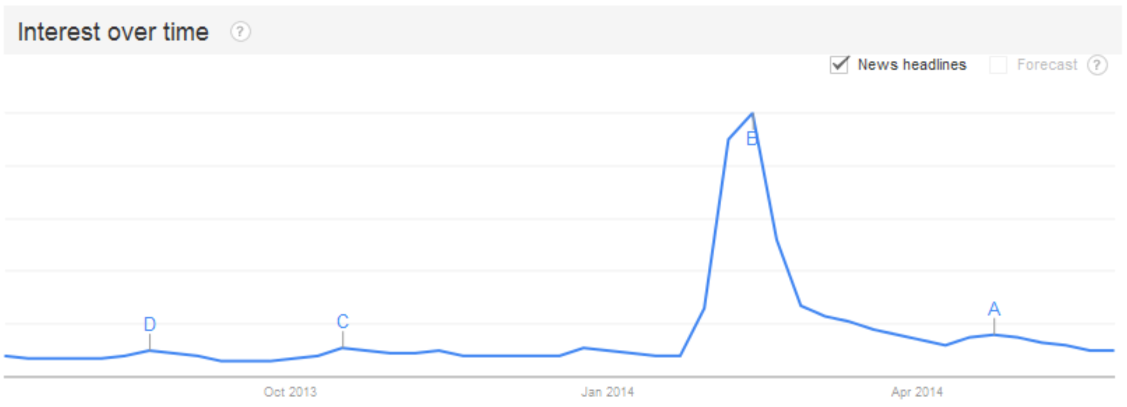


Figure 4 – Dungeon Keeper search popularity, starting from its announcement. Screen capture from www.google.com, retrieved 4.6.2014.

While studying the success of each of the inspected games, a strong difference of post-release performance can be seen. While Supercell, the developers of Clash of Clans, enjoyed success in the form of acquisition by SoftBank and GungHo (Softbank 2013), the developers of Dungeon Keeper were not as fortunate. Late in May 2014 news that EA, the owners of Mythic Entertainment, are closing the studio emerged (Schreier, 2014). Drawing conclusions from big companies' actions like EA's may not lead to a realistic estimate of the undergoing events, it could be speculated that the financial performance of Dungeon Keeper was not on par with what the stakeholders had expected, especially with the game's release after the success of Clash of Clans, and they had to let the team go.

5 The Usage of Microtransactions

5.1 Introducing Project TownBuilder

The idea for an Android game for this project originates from a course project that was made for the platform. The Android platform offered much more freedom than game engines did, and it seemed like a worthy system to study and learn. The game has been built from ground up on the Android platform libraries, and it utilizes its user interface tools as well as the Canvas element for drawing the game.

The design for a peaceful co-op town building game stems from personal experiences and preferences. The genre may be somewhat niche, but it is only an advantage to this specific project as it allows it to stand out from the crowd a little easier. The genre’s compatibility with microtransactions seems also fairly undocumented, so even though it is out of the scope of this thesis, the project will serve as an experiment on the viability of the format.

The whole game will not be released as open source, but the modules that are directly related to this thesis, the MTX modules, will be available to download for public and they will also be appended to the thesis as they currently stand. The current implementation may be inefficient and hastily made due to the timeframe available for the thesis.

5.2 The Design

5.2.1 Gameplay design

Project TownBuilder is a mobile game where the player is put in charge of running a post-apocalyptic town. The player has to set up their headquarters and start gathering resources from their surroundings in order to improve their central buildings and grow their town larger. The objective of the game is to gather as big score as possible by converting their own resources or using them to trade for resources that other players have in order to gain more points for themselves. The planned multiplayer features concentrate heavily on cooperation instead of opposition.

At the beginning, the players start with nothing else but their headquarter building and some resources to be able to build some of the starting buildings. To progress in the game, they have to identify the resource nodes and build resource gathering buildings on them in order to start gaining more resources to spend on other buildings. Placing the available buildings unlocks new ones according to a technology tree, opening up an element of planning for the player. As they progress in the game, the player can eventually unlock and build buildings that allow them to expand their territory, uncovering new resource nodes and other valuables, as well as buildings that enable them to communicate and reach further players with their trades.

When the player signs up for the game, they are created a profile where their progress is being saved, so that they do not have to start their town from zero every time they start playing. This way the game provides a sensation of progression, and allows the player to progress by only popping in the game for a short time to build a building or upgrade their headquarters. Unlike in most social and mobile games such as Sim City Social, gathering resources does not require player action so that the income stays steady. Technically this is done by evaluating the progress that has happened while the player has not played the game when they next time log in the game.

5.2.2 Microtransaction-induced design

This chapter will list the design choices that are induced by the intended microtransactions. The design of the microtransactions are set to follow three principles:

- The game has to be enjoyable and playable without spending any money
- People who wish to support the developer by making a one-time purchase must get their money’s worth in the game without giving them unfair advantage over others.
- People who are willing and able to make several, even regular, purchases must also get their money’s worth in the game without giving them unfair advantage or eliminating difficulty, causing the game to become less enjoyable.

The principles are not based on professional experience or research, but rather on personal experience in gaming and the ideas and desires projected from that. While the principles may appear utopistic at first, they are worth aiming for as the field of the subject is still relatively fresh and its true limits undiscovered.

The game will incorporate a ”premium currency” system, which can be seen in majority of games that allow microtransaction-paid content. The wide adaption of the system, in numerous titles such as Settlers Online or Hill Climb Racing to name a few, can be interpreted as its proof of efficiency over alternatives such as directly purchasing in-game items or bundles. A premium currency grants several perks: it simplifies the purchase process, eliminating the need to process orders for dozens of items and narrowing the purchases to bundles of premium currency, this can be confirmed by comparing the market mechanics for example in Team Fortress 2, which has real money price for every item and requires a money transaction for every purchase, and APB:

Reloaded, which relies on a premium currency (G1C), which allows several item purchases per real money transaction. The premium currency in TownBuilder does not act exactly as it traditionally does, like in for example the earlier analyzed Clash of Clans or Dungeon Keeper. Players can purchase the premium currency with real money, or they can find or produce it themselves in the game in smaller amounts. In addition to being a commodity to trade for in-game items, the currency itself works as a booster of kind, improving the efficiency of conversion rates for regular resources when consumed in the conversion process. The idea is somewhat inspired by the XP conversion mechanic in World of Tanks, described more in detail in the chapter 2.3. The premium currency itself does not add you resources, but allows the player to get better worth for their already spent effort. This attempts to invoke loss aversion by having the player already done their work and letting them decide themselves whether they want to get paid more for them. The premium currency is designed to allow the gaming hobbyists' large scale consumption, but in addition to buying it with real money it will be also available through regular gameplay, so that it can be included in the game's core mechanics without leaving out the players who do not wish to pay for the game. Catering to the non-paying players is important, as they make up the majority of players in a free-to-play games as discussed in chapter 3.2 on the topic of conversion rates.

Purchasable items can be divided into two categories: one-time-purchase features and consumable items. The pricing of features is determined by setting the game a price point, which will be equal to the sum of all features available for purchase. In a sense, the game is sold in separate parts. The features that will be sold are designed as being non-mandatory for effective gameplay, but still offering not necessarily an advantage, but rather ease on restrictions or alternative gameplay methods for those who possess them. An example of a purchasable feature is the ”Exploration Pack”, which is designed to allow the player leave their town and discover other players' town in order to trade and otherwise communicate with them in order to boost their own production. The purpose of the purchasable features is to allow people to contribute to the game by making a single purchase, but still gain something that will affect the rest of their time with the game.

Consumable items are designed to give a permanent or a temporary boost or cosmetic change to the player's units or resources. Due to the project not containing a graphic designer, cosmetic purchases may need to be closed out as consumable option in the early phases of development. The premium currency bundles also count as consumable items, although the player is in no control of when to consume the bundles, as they are consumed upon purchase in order to give the player the amount of premium currency they have just purchased.

The principles listed in the beginning of the chapter should be fulfilled with these baseline design instructions. The non-payer player's enjoyment must be guaranteed with the actual game design, which is already important for the success of the whole project. For players who wish to make a contribution or a single purchase, the one-time-items should satisfy their needs. The premium currency has no purchase limit, which makes it optimal for people with regular disposable income they wish to spend on the game. The attractiveness of the premium currency can be adjusted by the items that the player can purchase with them, as well as by balancing its availability within the game.

Due to the unfinished and largely undesigned nature of the game's user interface, the game cannot be put under analysis for the UK OFT guidelines used in the case studies, but as they are acknowledged during the design, they will be taken in account while putting together the user interface for the game. Some concrete examples are that the in-app purchases will be confined to a shop menu behind a visible, but non-intrusive button. If design-wise possible, the game will try to avoid any kind of direct purchase impulse buttons. The game will be designed for young adults and older, so the children are not in the main focus of the microtransaction marketing. Regardless of the main demographic, the game will not be restricted from children and will attempt to comply with the previously mentioned guidelines, as they also imply respect for the human value of the player, which will be of priority in the design of this game.

5.3 The Implementation

5.3.1 Used tools and frameworks

The tools and frameworks used in the project have been chosen to cater for Android development using Google's and Android's own libraries and technologies. The programming, app layout design and app setup is done in the Android Developer Tools

(ADT), which is a modified version of the open sourced programming IDE (Integrated Development Environment) Eclipse. The reason why ADT was chosen was mainly because it was one of the only Android development environments during the beginning of the project, and was the offered tool from the Android team.

It is worth noting that the start of the TownBuilder project was nearly one year before the work on the actual thesis started, and during that time some alternatives to the Eclipse-based ADT have arisen. One potential alternative is the Android Studio, which is currently under development and available from the Android developer resources. Instead of Eclipse, the Android Studio is based on IntelliJ IDEA. It is still not finished, but an early access preview is available for those who do not mind facing bugs or incomplete features. One of the main reasons the TownBuilder project was not migrated to the Android Studio was that the relatively short schedule that had been planned for the project would be compromised by the need to learn the new environment and becoming as fluent with it as with the ADT.

Another alternative to the ADT was Unity, and while it is a highly efficient tool to produce content on, it did not meet the personal requirements set for this project. While selecting the tools for TownBuilder, Unity did not have its 2D toolkit implemented yet, and while Unity had 2D frameworks available, the toolset itself was still optimized for 3D production (Saarelainen & Pakarinen 2013, 10–12). Another criteria for dismissing Unity was the desire for advancing personal skills on Android development and the motivation to learn something new. Using ADT and a custom game engine permits to apply personal touch to the game without being constricted by the limits of another game engine, such as Unity. Soon after starting the development, Unity released its 2D toolkit, which could have made the production of TownBuilder faster, but the change of tools was never made due to the custom game engine being in a good shape, and because one of the core ideas for the thesis was to make something that is not made with Unity. In hindsight the Unity environment would have been too heavy for this purpose, as the game is only in 2D and is not action packed enough to require heavy calculation and sprite optimization.

5.3.2 The Microtransaction-modules

For the purpose of implementing microtransactions into TownBuilder, which is an Android game, the ”MTX”-package was set to be developed. This will not be the first of its kind, as somewhat similar solutions already exist, for example prime[31] offers an In-App Billing plugin for Unity that allows Google Play microtransaction integration. What separates this MTX module from the Unity plugin is that the MTX module offers greater customization, as the Unity plugin is merely for communicating with Google Play. The purpose of this package is to simplify the usage flow of using microtransactions in a Java application. The structure and logic of the package is described in figure 5 as it is in use in the project TownBuilder. The design for the modules in this package was inspired by the idea that the developer could simply invoke a method call from their main code and receive a result to a method that handles the data according to their own programming. The advantage to this is to separate the different independent actors in the purchase to their own modules is the versatility in case of porting to a different platform or a different vendor, when the functionality can be programmed to their own modules without the need to change game code.

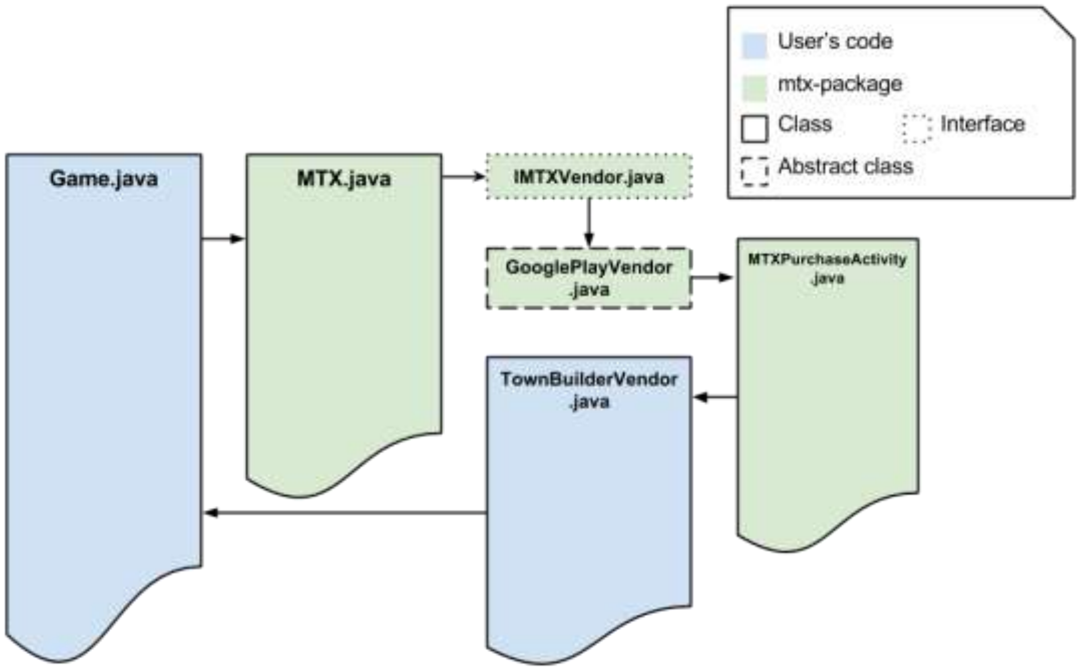


Figure 5. The structure and data flow of a purchase activity in MTX module

The Game.java code file in picture 1 represents the actual game code. The function calls to the MTX class are not made strictly in one code file, but rather they have been placed in more context-related locations such as buttons' on-click functions. Overall control of the modules in this package is in the MTX.java, which relays the commands to the appropriate vendor objects and handles received events. The MTX class is used by creating an instance of it in the game code, passing the game-specific vendor implementation which in this case is TownBuilderVendor.java. The vendor class that the MTX module uses bases itself to the IMTXVendor interface, which defines methods for purchasing items and retrieving the product list from the vendor. In addition to the commands, the vendor must also implement methods to handle the events of receiving the product list and receiving a successful purchase. These methods are to be defined in the app-specific implementation, as they are expected to display information on the screen, and the control of the app's UI is left to the implementing class. The next iteration of the vendor module is the abstract class defined in GooglePlayVendor.java, which connects to the Google Play service upon creation. This class implements the purchasing and product list fetching functionality through the connection that is stored in it, but also imposes new abstract methods that need to be implemented in the app-specific class. These methods are related to the Android platform and used for retrieving the current Context object or an id pointing to a store layout. The Google Play Vendor class also requires the MTXPurchaseActivity class, which inherits the properties of an Android Activity, which are basically used to display different screens in an app. The Activity is required to successfully display the purchase confirmation dialog. Further iteration of the module could allow the currently displayed Activity to be used for the purchase dialog, but such feature is not present in the current version of the package. Finally in the vendor class hierarchy is the TownBuilderVendor, which handles the incoming product lists by displaying them in a store dialog, and purchases by checking the product id of the received purchase and performing the predefined actions set to them. The TownBuilderVendor class also defines the product ids that are available for purchase, and that are used to query the serverside product list.

A purchase with the module is done by first instantiating an instance of the MTX class in the game's code and then calling the purchase method through the object. The command is relayed to the vendor class defined in the MTX object's constructor, which uses its connection to the vendor service to retrieve the data and request the purchase from the user. The actions are done in an asynchronous task, so that the app will stay responsive and does not freeze completely. After the task has finished, the vendor class calls back to the MTX class to report that the action is completed, and that information is again relayed to the app-specific implementation of the vendor class. This class then processes the purchase, and adds the player some of the premium currency for example, had the player purchased one of the bundles.

The MTX modules were written with portability in mind. The different platform requirements are confined within single modules, which aims to make the code package usable in virtually any Java environment. The support for non-Android projects can be achieved by writing a new platform-specific class that implements the IMTXVendor interface, although as of now the only option is the Android and Google Play implementation. In order for a developer to implement these modules to their project, they need to import the packages and write their own implementation of the GooglePlayVendor class that implements the responses to receiving a product list or a purchase confirmation. After this the developer only needs to create an instance of the MTX class with their vendor implementation and call the appropriate methods when they need to make a purchase or retrieve the product list from the vendor.

5.3.3 Integration with Google Play

In order to sell in-app products with Google Play the developer will have to gain access to a Developer Console, which requires a Google Play publisher account, and a Google Wallet merchant account. Registering a publisher account requires a payment of $25 to Google, and the Google Wallet account can be created through the Developer Console. Registering for these accounts requires the developer to provide personal contact information and allows for providing company and tax information as well for those who it applies to. However it is not required to have a company in order to register for a publisher or Google Wallet account.

The actual communication with Google Play from the app is done in TownBuilder by the GooglePlayVendor abstract class, which utilizes the Google's com.android.vending.billing library. Upon being created, the class instantiates a ServiceConnection object to the InAppBillingService, which is bound to the application's Context, meaning that it will exist as long as the app exists. The connection is used when the buy or product list retrieval methods are called, although in both cases the user is not the class itself. In purchase action the connection is retrieved by the initialized Activity that helps display the purchase confirmation dialog. In retrieving the product list, GooglePlayVendor retrieves the application’s product strings defined inside the program and then instantiates an asynchronous task to wait for the response from the server. The product list fetch contacts the Google Play servers and retrieves information of in-app-purchasable items that match the strings that are sent to it. To ensure that the service is established, the user can pass a parameter that makes the program wait for the service for a set amount of time. This is most useful when the service needs to be used quickly after its initialization, and should not be required otherwise.

6 The Conclusion on Microtransactions

The setup for this thesis can be described as a question, asking ”how do I use microtransactions in an Android game?”. The research question is intended to be a double entendre to cover the two aspects of the perceived results: the design and the practical code implementation. The basis for the resulting design of the project TownBuilder was laid in chapter 3 when the nature and characteristics of microtransactions were studied. Although these practices were taken in account while designing the in-app purchases for TownBuilder, it is impossible to achieve perfection, and the resulting items, pricing and gameplay integration may need to be iterated several times in order to achieve the best efficiency. The objective was not to design a final product, but to create a foundation for steady design that includes in-app microtransactions. The most important goal was to avoid the ”black hat design” in maximizing the profitability of the microtransactions. This was achieved by making the premium currency available inside the game by gameplay and with the limitation that the premium currency does not necessarily create resources for the player out of nothing and instead they only reward the player better when they actually play the game. Optimizing the profitability was done by offering both one-time buyers and the ”hobbyists” an option for a purchase. Premium currency is for the heavy spenders, and the purchasable features work as one-time purchases that give advantage for a long time without unbalancing the game. After having written the report and seen some seminars from industry veterans such as Teut Weidemann, it appears the design principles set for

TownBuilder are not as utopistic as thought, and similar designs actually in use in successful free-to-play games already.

The backing for the code implementation results in the report comes from the actual TownBuilder project. With the code files made and tested in practice, it gives a proper source to base the findings in the implementation process. The MTX package is appended to the report, so that its portability can be tested in practice. The game itself is in an early alpha stage at the time of finishing the thesis, but it will be made available on Google Play as soon as it is in a finishable state, so that the actual design can be experienced in practice as well. No significant technical problems emerged during the project, and those few problems that did were related mostly to Android platform’s Activity management.

The tools, frameworks and methods that were chosen for this thesis were sufficient and fit for their purpose. Android Developer Tools proved to be a well supported up-to-date platform to develop an Android game on, with plenty of support in problematic situations from the developer as well as the community. The research methods for the history and nature of microtransactions were mainly studying the articles and documents related to the topic, and was mainly done on sources available on the internet. Due to the relative freshness of the subject, very little up-to-date literature was to be found. The two case studies in this thesis gave several viewpoints to the state of Android mobile games, as it was found after the evaluation of both games that they were essentially made with the same formula. Analyzing Clash of Clans and Dungeon Keeper with the UK Office of Fair Trading guidelines shed some objective light on the in-app purchases offered by both games. While studying mobile games, objectivity is important as in some cases the majority of a gaming community and the developers may not be in perfect harmony, which causes a strong bias of facts and opinions on both sides: the players wish to have enjoyable gameplay and do not expect to pay too much but on the other hand the developers and especially the stakeholders wish to maximize their investments, sometimes at the cost of the players’ wishes.

As mentioned in the chapter defining project TownBuilder’s design principles, the writer of this thesis is merely a game programming student with no real accountable professional experience. Despite the intention to stay objective while studying microtransactions, all objectiveness originates mostly from a gamer and customer perspective. To guarantee credibility, a variety sources have been used ranging from research companies to releases from appreciated industry newsletters.

The personal learning process during this thesis has been significant. Less in general information about microtransactions, but even more so in Java and Android programming, project and time management as well as writing capabilities. While starting the TownBuilder project, my Android knowledge was still limited and I was not completely clear on Activity lifecycles and general inner workings of Android apps. Now after several iterations on the game project, I have better knowledge on good practices and optimizations in Android projects. The writing of the thesis report was fairly difficult at first, but as the report grew larger and larger, the confidence in my own writing also strengthened, and I could see that it is possible to create large amounts of content in a short time if the premises are set correctly. I became aware that studying the subject yourself helps significantly in the writing of new content.

As mentioned before, perfection is not within reach, at least for this thesis. The development on TownBuilder will continue with this research backing its design, and the current plans for the project is to finish it as a playable game and release it properly on Google Play. The MTX modules will be released as open source, and updated to improve performance and ease of use. Further research ideas in the area of microtransactions could be, for example, examining the revenue numbers and their correlation to the prices of in-app purchases of a game or the advantages offered with the purchasable items. Hopefully the thesis itself will find some use in helping developers’ decisionmaking and making customers understand the reasoning between some of the microtransactions that are used in the games they play, all this in hopes of making all kinds of gaming better and more profitable, without harming the customers, so that the future of the industry can be guaranteed.

REFERENCES

Blow, J. 2013. Game design: the medium is the message. CreativeMornings. http://creativemornings.com/talks/jonathan-blow. 2.2.2014.

Cailliau, R. 1995. A Little History of the World Wide Web. http://www.w3.org/History.html. 2.6.2014.

Cambridge University Press. 2014. micropayment noun - definition in the Business English Dictionary - Cambridge Dictionaries Online. http://dictionary.cambridge.org/dictionary/business-english/micropayment. 2.6.2014.

EA Mobile. 2014. Dungeon Keeper - Android Apps on Google Play. https://play.google.com/store/apps/details?id=com.ea.game.dungeonkeeper_row. 2.6.2014.

Handrahan, M. 2014. Molyneux: EA's Dungeon Keeper "crucifies my patience" | GamesIndustry International. http://www.gamesindustry.biz/articles/2014-02-11-molyneux-eas-dungeon-keeper-crucifies-my-patience. 2.6.2014.

Levy, E. 2014. Gamasutra: Ethan Levy's Blog - The demon driving Dungeon Keeper backlash. http://www.gamasutra.com/blogs/EthanLevy/20140210/210391/The_demon_driving_Dungeon_Keeper_backlash.php. 2.6.2014.

Meade, B. 2014. Mobile is burning, and free-to-play binds the hands of devs who want to help | Polygon. http://www.polygon.com/2014/5/9/5699058/free-to-play-mobile-candy-crush-the-room. 2.6.2014.

Metacritic. 2014a. Clash of Clans for iPhone/iPad Reviews – Metacritic. http://www.metacritic.com/game/ios/clash-of-clans. 2.6.2014.

Metacritic. 2014b. Dungeon Keeper Critic Reviews for iPhone/iPad – Metacritic. http://www.metacritic.com/game/ios/dungeon-keeper/critic-reviews. 2.6.2014.

Office of Fair Trading. 2013. The OFT's Principles for online and app-based games. http://www.oft.gov.uk/shared_oft/consumer-enforcement/oft1519.pdf . 15.3.2014.

Peterson, S. 2012. Next-gen consoles mean increased development costs. http://www.gamesindustry.biz/articles/2012-04-03-next-gen-consoles-mean-increased-development-costs. 2.6.2014.

PocketGamer.biz. 2013. Investors aren't interested unless you generate $1M a day, says Fishlabs. http://www.gamasutra.com/view/news/203337/To_get_investors_interested_you_need_to_generate_1M_a_day_says_Fishlabs.php. 2.6.2014.

Rose, M. 2013. Chasing the Whale: Examining the ethics of free-to-play games. http://www.gamasutra.com/view/feature/195806/chasing_the_whale_examining_the_.php. 2.6.2014.

Saarelainen, T. & Pakarinen, M. 2013. 2D Game Development With Unity 3D. http://theseus.fi/bitstream/handle/10024/68508/Saarelainen_Taavi_Pakarinen_Miika.pdf?sequence=1. 2.6.2014.

Schreier, J. 2014. EA Shuts Down Longtime Game Studio Mythic Entertainment. http://kotaku.com/ea-shuts-down-mythic-the-studio-behind-warhammer-onlin-1583376655. 2.6.2014.

Softbank. 2013. SoftBank and GungHo Announce Strategic Investment of USD 1.5 Billion in Supercell.

http://www.softbank.jp/en/corp/set/data/news/press/sb/2013/20131015_02/pdf/20131015_02.pdf. 2.6.2014.

Supercell. 2014. Clash of Clans - Android Apps on Google Play. https://play.google.com/store/apps/details?id=com.supercell.clashofclans. 2.6.2014.

SuperData. 2014a. US and China Mobile Games Markets Brief. http://www.slideshare.net/superdata/us-and-china-mobile-games-markets-brief. 3.6.2014.

SuperData. 2014b. Comparing MMO ARPU for major free-to-play titles. http://www.superdataresearch.com/blog/mmo-arpu/. 2.6.2014.

Swrve. 2014. Swrve Finds Over 35% Of All In-Game Revenues Are Delivered In The... -- SAN FRANCISCO, April 9, 2014 /PRNewswire/ --. http://www.prnewswire.com/news-releases/swrve-finds-over-35-of-all-in-game-revenues-are-delivered-in-the-first-three-days-of-player-life-254526381.html. 2.6.2014.

Tseng, J. 2011. Zynga "Appeals to the Same Psychology as Gambling," Says Analytics Expert Jeff Tseng. http://www.forbes.com/sites/benzingainsights/2011/08/12/zynga-appeals-to-the-same-psychology-as-gambling-says-analytics-expert-jeff-tseng/. 2.6.2014.

Weidemann, T. 2014. Dissecting World of Tanks Monetization | Teut WEIDEMANN – YouTube. https://www.youtube.com/watch?v=c7H40Qd_lis. 2.6.2014.

Yee, N. 2014. Free-to-play whales more rational than assumed. http://www.gamesindustry.biz/articles/2014-04-01-free-to-play-whales-more-rational-than-assumed. 2.6.2014.

Zichermann, G. 2012. Zynganomics: 4 Secrets of the Social Gaming Business Model. http://mashable.com/2012/03/23/zynga-economics/. 2.6.2014.

```
// For latest version, please see https://github.com/i-h/mtx-java

package org.rautasydan.mtx.common;


import java.lang.reflect.Constructor;
import java.util.ArrayList;

public class MTX {
          static IMTXVendor vendor;
          private static ArrayList<String> skuList;
          public MTXItemTable products;

          public MTX(Class<? extends IMTXVendor> vendorType) {
            IMTXVendor newVendor;

            try {
                      Constructor<? extends IMTXVendor> c =
          vendorType.getConstructor();
                      newVendor = vendorType.cast(c.newInstance());
            } catch (Exception e) {
                      e.printStackTrace();
                      newVendor = null;
            }
            vendor = newVendor;
            skuList = new ArrayList<String>();

            for(String s : vendor.getSkuList()){
                      skuList.add(s);
            }
          }
          public static void setVendor(IMTXVendor v){
            vendor = v;
          }
          public MTXItem purchase(MTXItem item){
            if(!vendor.buyItem(item)){
                      item.setQuantity(0);
            }
            return item;
          }
          public static void receivePurchase(MTXItem item){
            vendor.receivePurchase(item);
          }
          public void destroy() {
            vendor.destroy();
          }
          public void getProductList(){
            vendor.getProductList();
          }
          public void setSkuList(ArrayList<String> itemSkuList){
            skuList = itemSkuList;
          }
          public static ArrayList<String> getSkuList(){
            if(skuList == null){
                      skuList = new ArrayList<String>();
            }
            return skuList;
          }
          public static void receiveProductList(MTXItemTable items){
            vendor.receiveProductList(items);
          }
}
```

```
package org.rautasydan.mtx.common;

public class MTXItem {
         String id;
         String priceString;
         int quantity;
         float valuePerItem;

         public MTXItem(String itemID, String price){
          id = itemID;
          priceString = price;
         }
         public MTXItem(String itemID, float itemValue) {
          id = itemID;
          quantity = 1;
          valuePerItem = itemValue;
         }
         public MTXItem(String itemID, int itemQuantity, float
         itemValue){
          id = itemID;
          quantity = itemQuantity;
          valuePerItem = itemValue;
         }

         public String getId() {
          return id;
         }

         public int getQuantity() {
          return quantity;
         }

         public float getValuePerItem() {
          return valuePerItem;
         }

         public float getTotalValue() {
          return valuePerItem * quantity;
         }
         public String getPriceString(){
          if(priceString == null || priceString.equals("")){
                    return valuePerItem + "";
          } else {
                    return priceString;
          }
         }

         public void setId(String id) {
          this.id = id;
         }

         public void setQuantity(int quantity) {
          this.quantity = quantity;
         }

         public void setValuePerItem(float valuePerItem) {
          this.valuePerItem = valuePerItem;
         }


}
```

```
package org.rautasydan.mtx.common;

import java.util.ArrayList;

public class MTXItemTable extends ArrayList<MTXItem> {
         private static final long serialVersionUID =
         7936494652191185166L;
}
```

```
package org.rautasydan.mtx.common;


public interface IMTXVendor {
          public boolean buyItem(MTXItem itm);
          public void getProductList();
          public void receiveProductList(MTXItemTable items);
          public void receivePurchase(MTXItem item);
          public void destroy();
          public String[] getSkuList();

}
```

```
package org.rautasydan.mtx.android;

import org.rautasydan.mtx.common.MTX;
import org.rautasydan.mtx.common.MTXItem;
import org.rautasydan.mtx.common.IMTXVendor;

import com.android.vending.billing.IInAppBillingService;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.util.Log;

public abstract class GooglePlayVendor implements IMTXVendor {
        final protected Class<? extends Activity> storeActivity;
        final String tag = getClass().getSimpleName();
        final int IAPversion = 3;
        static IInAppBillingService iabService;
        ServiceConnection iabServConn = new ServiceConnection() {
         @Override
         public void onServiceDisconnected(ComponentName name) {
                   Log.i(tag, "iabService disconnected!");
                   iabService = null;
         }

         @Override
         public void onServiceConnected(ComponentName name, IBinder
        service) {
                   Log.i(tag, "iabService connected!");
                   iabService =
        IInAppBillingService.Stub.asInterface(service);
         }
        };

        public GooglePlayVendor(Class<? extends Activity>
        storeActivityClass) {
         storeActivity = storeActivityClass;
         // Bind to IInAppBillingService
         Intent billingIntent = new Intent(

         "com.android.vending.billing.InAppBillingService.BIND");
         getContext().bindService(billingIntent, iabServConn,
                         Context.BIND_AUTO_CREATE);
        }
        @Override
        public boolean buyItem(MTXItem itm) {
         try {
                   Bundle buyIntentBundle =
        iabService.getBuyIntent(IAPversion,

         getContext().getPackageName(), itm.getId(), "inapp", "");
                   Intent i = new Intent(getContext(),
        MTXPurchaseActivity.class);
                   i.putExtra("buy", buyIntentBundle);
                   getContext().startActivity(i);

         } catch (RemoteException e) {
                   e.printStackTrace();
```

```
          }
          return true;
        }

        @Override
        public void getProductList() {
          Bundle querySkus = new Bundle();
          querySkus.putStringArrayList("ITEM_ID_LIST",
        MTX.getSkuList());
          MTXTask task = new MTXTask(getContext(), this);
          task.execute(querySkus);
        }

        @Override
        public void destroy() {
          if (iabService != null) {
                    getContext().unbindService(iabServConn);
          }
        }

        public void displayStore() {
          Intent i = new Intent(getContext(), this.getClass());
          getContext().startActivity(i);
        }

        public void waitUntilConnected(float timeout) {
          if (iabService != null) {
                    return;
          }
          float wait = 100.0f;
          float elapsed = 0;
          float maxWait;
          if (timeout <= 0) {
                    maxWait = 1000;
          } else {
                    maxWait = timeout;
          }
          Log.w(tag, "Starting wait for iabService...");
          while (iabService == null || elapsed < maxWait) {
                    Log.i(tag, "Waiting..." + elapsed + "/" +
        maxWait);
                    elapsed += wait;
                    try {
                              Thread.sleep((long) wait);
                    } catch (InterruptedException e) {
                              Log.e(tag, "Wait interrupted: " +
        e.toString());
                    }
          }
          Log.w(tag, "Wait for iabService over.");
        }

        public IInAppBillingService getIabService(boolean wait) {
          if (wait) {
                    waitUntilConnected(-1);
          }
          Log.i(tag, "iabService = " + iabService);
          return iabService;
        }
        protected abstract Context getContext();
        protected abstract int getStoreLayoutID();
}
```

```
package org.rautasydan.mtx.android;

import java.util.ArrayList;

import org.json.JSONException;
import org.json.JSONObject;
import org.rautasydan.mtx.common.MTX;
import org.rautasydan.mtx.common.MTXItem;
import org.rautasydan.mtx.common.MTXItemTable;

import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.RemoteException;
import android.util.Log;

public class MTXTask extends AsyncTask<Bundle, Integer, Bundle> {
         GooglePlayVendor vendor;
         Context context;
         String tag = "MTXTask";
         int timeout = 20;

         public MTXTask(Context ctx, GooglePlayVendor service) {
          super();
          vendor = service;
          context = ctx;
         }

         @Override
         protected Bundle doInBackground(Bundle... querySkus) {
          Bundle skuDetails= new Bundle();
          skuDetails.putInt("RESPONSE_CODE", 1);
          try {
                     skuDetails =
         vendor.getIabService(true).getSkuDetails(3,
         context.getPackageName(),
                                        "inapp", querySkus[0]);
                     Log.i(tag, "Got skuDetails: " +
         skuDetails.toString());
          } catch (RemoteException e) {
                     Log.e(tag, "Error retrieving sku details for
         product list" + e.toString());
          } catch (NullPointerException e){
                     Log.e(tag, "iabService was not ready yet!" +
         e.toString());
          }
          return skuDetails;
         }

         @Override
         protected void onPostExecute(Bundle skuDetails) {
          super.onPostExecute(skuDetails);
          MTXItemTable items = new MTXItemTable();
          int response = skuDetails.getInt("RESPONSE_CODE");
          if(response == 0){
                     ArrayList<String> responseList =
         skuDetails.getStringArrayList("DETAILS_LIST");
                     for(String s : responseList){
                               try {
                                        JSONObject obj = new
         JSONObject(s);
                                        String sku =
         obj.getString("productId");
```

```
                                        String price =
obj.getString("price");
                                        items.add(new MTXItem(sku,
price));
                                        Log.i(tag, "Item received:
" + sku + ": " + price);
                        } catch (JSONException e) {
                                        Log.e(tag, "Couldn't parse
sku detail response: " + e.toString());
                        }
            }
 } else {
            Log.e(tag, "Response not ok, was " + response);
 }
 MTX.receiveProductList(items);
}

}
```

```
package org.rautasydan.mtx.android;


import org.json.JSONException;
import org.json.JSONObject;
import org.rautasydan.mtx.common.MTX;
import org.rautasydan.mtx.common.MTXItem;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.IntentSender.SendIntentException;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

public class MTXPurchaseActivity extends Activity {
         String tag = "PurchaseActivity";
         public static boolean isActive = false;

         @Override
         protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          if (getIntent().hasExtra("buy")) {
                     Bundle buyIntentBundle =
         getIntent().getBundleExtra("buy");
                     PendingIntent pendingIntent = buyIntentBundle

          .getParcelable("BUY_INTENT");
                     try {

          startIntentSenderForResult(pendingIntent.getIntentSender(),
                                                    1001, new
         Intent(), Integer.valueOf(0),

          Integer.valueOf(0), Integer.valueOf(0));
                     } catch (SendIntentException e) {
                               e.printStackTrace();
                     }
          } else {
                     Log.e(tag, "No buyIntentBundle passed, not buying
         anything >:(");
          }

         }

         @Override
         protected void onStart() {
          super.onStart();
          isActive = true;
         }

         @Override
         protected void onStop() {
          super.onStop();
          isActive = false;
         }

         @Override
         protected void onActivityResult(int requestCode, int
         resultCode, Intent data) {
          super.onActivityResult(requestCode, resultCode, data);
          if (requestCode == 1001) {
            String purchaseData =
data.getStringExtra("INAPP_PURCHASE_DATA");

            if (resultCode == RESULT_OK) {
                    try {
                            JSONObject jo = new
JSONObject(purchaseData);
                            String sku =
jo.getString("productId");
                            int amount = 1;
                            float price = 0.00f;

 Toast.makeText(getApplicationContext(),
                                                   "You
have bought the " + sku + ".",

 Toast.LENGTH_LONG).show();
                            MTXItem boughtItem = new
MTXItem(sku, amount, price);

 MTX.receivePurchase(boughtItem);
                    } catch (JSONException e) {

 Toast.makeText(getApplicationContext(),

 "Failed to parse purchase data.", Toast.LENGTH_LONG)

 .show();
                            e.printStackTrace();
                    }
            }

 }
 finish();
}

}
```