LAPIN AMK
Lapland University of Applied Sciences

INDEPENDENT GAME DEVELOPMENT

Juha-Matti Mäkinen and Juho-Petteri Oinas

Bachelor’s Thesis
Lapland University of Applied Sciences
Business Information Technologies

2015

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Lapin Ammattikorkeakoulu
Business Information Technology

Tekijä	Juha-Matti Mäkinen, Juho Petteri Oinas	Vuosi	2015
Ohjaaja	Etunimi Sukunimi		
Toimeksiantaja	Toimeksiantajan nimi		
Työn nimi	Independent Game Development		
Sivu- ja liitemäärä	42 + 7		

Tämä opinnäytetyö on tarkoitettu niin aloitteleville kuin jo edistyneemmillekin pelinkehittäjille. Opinnäytetyö käsittelee pelien teoriaa ja pelien kehittämistä. Opinnäytetyössä tutkitaan kaupallisten ja sitoutumattomasti kehitettyjen pelien eroavaisuuksia sekä työkaluja, joita käytetään pelinkehityksessä. Opinnäytetyössä tutkitaan tarvetta pelinkehityksen käsikirjalle ja sitä, miten pelejä kehitetään sitoutumattomassa ympäristössä.

Opinnäytetyön tarkoitus on tutkia pelinkehitystä, jota tarkastellaan taustatiedon avulla. Tavoitteena oli luoda käsikirja olemassa olevien töiden, kirjallisuuden ja käytännöllisen pelinkehityksen avulla. Käsikirjaa voidaan käyttää pelinkehityksessä. Tavoitteena oli myös tehdä pelin prototyyppi ja kuvailla sen tekeminen osana opinnäytetyötä.

Opinnäytetyö sisältää pelinkehityksen, testauksen, ideoinnin ja kehitysprojektin. Opinnäytetyö ei käsittele rahoitusta, markkinointia, julkaisua, tai sitoutumattoman ohjelmiston ylläpitoa. Opinnäytetyössä oletetaan, että sitoutumattomat pelit eivät ole hyvin tunnettuja lukijalle ennestään, siksi joitakin käsitteitä on selitetty perusteellisemmin.

Opinnäytetyö pohjautuu kirjallisuuteen, videoihin ja aiheeseen liittyviin tutkimuksiin. Rakentavaa ja tutkivaa tutkimustapaa on käytetty opinnäytetyössä syvemmän ymmärryksen saavuttamiseksi siitä, mitä pelinkehitys on ja mikä on pelinkehitysympäristö kokonaisuudessaan. Opinnäytetyön lopputuloksena on käsikirjan avulla kehitetty prototyyppi ja käsikirja alkeelliselle pelinkehitykselle, jota voidaan käyttää sitoutumattomassa pelinkehityksessä oppaana.

Avainsanat pelit, pelin kehitys, sitoutumattomat pelit

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Lapland University of Applied Sciences
Business Information Technology

Author	Juha-Matti Mäkinen, Juho Petteri Oinas	Year	2015
Supervisor	Sari Mattinen		
Commissioned by	Name of Commissioner		
Title of Thesis	Independent Game Development		
No. of pages + app.	42 + 7		

This thesis was meant for the budding developers and advanced developers. The thesis was about the theory of games and game development. In this thesis the difference between independently and commercially developed games, as well as tools, was investigated. The thesis investigated the need for a handbook with guidelines for game development and how games were developed in an independent environment.

First objective of the thesis was to study game development by understanding the theoretical background. Second objective was to create a handbook for game development based on existing literature and practical game development. Third objective was to create a game prototype and describe its creation as a part of the thesis.

The scope of the thesis included game development, testing, conceptualizing and a development project. The scope of the thesis did not include monetizing, marketing, publishing or updating independent software. The thesis assumed that the reader was not familiar with the concept of independent games, therefore some of the concepts were explained more in-depth.

The result of the thesis was a prototype developed with the help of the handbook and a handbook for basic game development, which can be used as a guide for independent game development.

Key words Games, Development, Independent games

CONTENTS

1 INTRODUCTION

The first objective of the thesis is to study the independent game development, as well as game development. The observations made focus on analyzing the history of both development fields and the studied fields of development history will be used to achieve a deeper understanding of the topic. The second objective of the thesis is to produce development methods and tools to guide and be used in development of games. Producing the guide for developers will be done by investigations of existing handbooks, literature on software development and observing the practical development of games. Within the handbook, the goal is to collect the information to produce a guide that is up-to-date. The third objective is to design and implement the handbook in the form of a prototype of an independently developed game's mechanics.

In game development there are two different fields. These are Independent and corporate fields. These two fields bring forth different types of games to the gamers of the world. Their differences in development methods, are fairly minimal, however their budgets and collaborators can differ astronomically. The styles of development section speaks of independent development and triple-A title development, summarizing their differences in the comparison section.

The game development and management section goes through genres, development roles, design and finally production. The genres are divided into categories that are popularly used in media. Games are usually categorized so players can find what they are looking for. The development roles explains the different roles that are present in game development. These roles can sometimes intertwine, when the cast is small, as it usually is in independent productions.

After game development, comes design. In the midst of development comes fault testing, bug searching and defining the 'game world'. Graphics, user interface, story, narrative and artistic choices are all parts of the design process.

The first focus in this thesis is to find the major differences between the two fields of game development. These are the risk management and the scale of the project. We will also take a brief look at historical materials, to give insight on the differences of genres. As a development focused topic, this thesis will eschew the marketing, monetization and post-release maintenance related partitions.

Before starting the thesis proper, it is best to define what the word ‘game’ actually means. While some sources, such as “the Extra Credits” a Youtube-based series focused on game development and analysis, dismiss the notion of defining games at all as arbitrarily limiting them as an art medium. Instead Extra Credits prefers the term ‘interactive experiences’ for the entire medium. However, the term used by Extra Credits is too vague for the purposes of this thesis, due to it referring to experiences as a game, when a rollercoaster ride and watching a DVD are not ‘games’, but entertainment mediums that offer little interactivity beyond starting and stopping. As such, a more concrete definition of games is required. (Extra Credits - ‘What is a Game? – How This Question Limits Our Medium’.)

Juul (2005, 35-44.) mentions that all games include six definitions, that make games what they are. Of course games try to be different from each other, altering this list, but basically all the games include:

> “*A game is a rule-based system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort in order to influence the outcome, the player feels emotionally attached to the outcome and the consequences of the activity are negotiable.*” Juul (2005, 36.)

The model, described by Jesper Juul, is the basis upon which games are built. Having these six features is enough to make a game. However it is not necessary to have all six features in a specific combination. There are games that run linearly, excluding for example player choices, but offer different sets of endings, depending on how well the player performs in game. Combining these six features, mixing and matching portions, gives more variation and creativity in new games these days. (Juul 2005, 35-44.)

For the purposes of this thesis, a distilled version of Juul's definition is shortened into two mechanical requirements for a game:

1. The players have the ability to act and the actions affect the game world.
2. The game has at least one victory or failure state caused by the player actions. After the actions, the player is either rewarded or penalized, or the game ends.

The first requirement allows for the player to interact and affect the environment within the game with their choice of action. The second requirement allows for the game to penalize the player for errors and encourage actions that advance the story.

1.1 Structure of Thesis

Chapter 1 introduces the thesis to the reader. It goes through motivations, objectives, scope, topic and limitations. Chapter 2 details the research methods and research questions. Chapter 3 introduces concepts independent and triple-A, drawing a comparison between them. Chapter 4 is about genres in general, going through what they are and brief history. Chapter 5 describes the different roles present in the development of games. Chapter 6 goes through the phases of design. It talks about analysis, design, implementation and testing of a game. Chapter 7 goes through different stages of production. The chapter tells about pre-production, prototyping and game builds.

Chapter 8 has been made to demonstrate what is discussed in the Handbook found in Appendix 1. The chapter features the practical side of the thesis, where a prototype is built. Chapter 9 answers the research questions and presents the conclusions of the thesis.

2 RESEARCH METHODS AND RESEARCH QUESTIONS

The development of the handbook led to the choice of constructive research method. The constructive research method builds constructs in order to construct a solution for issues that are present in real world scenarios. The aim of constructive methodology is to produce results that will detail how the issue can be solved. Constructive methodology combines the practical partitions with theoretical partitions to create a construct compiling their answers into a solution. (Lukka 2000.) Constructive research is used to present a scientifically relevant problem, as well as innovatively constructing a solution to the problem. Constructive research also tests the solution to the problem to prove validity of compiled solution. (Rohweder & Virtanen 2008. 11-14.)

The exploratory research methodology is used to acquire information in this thesis on areas that would otherwise be left vague. Exploratory research gives a deeper understanding of the topic. Exploratory research methodology concentrates on developing, as well as comprehending, instead of acquiring reproducible data. (Wiid & Diggines, 2009. 55.)

Both, exploratory and constructive research were used to create the texts present in the thesis. Exploratory research method was used to gather the data from internet articles, thesis works, books, videos, and tutorials. Constructive research methods were used to create the handbook based on the materials and results of explorative research.

The following section talks about the three research questions of the thesis in more detail. The research questions are presented in a single sentence above the details. The answers will then be presented in research findings.

How does Independent Game Development differ from the triple-A standard Game Development?

The games that are independently developed are usually done by small teams. Independent games have little to no funding to start with. Independent games are developed in a very different manner from triple-A games. Independently developed games do not have a recipe to repeat, all the work is made towards the game being fun and not so much to follow a certain standard.

How would a handbook on game development tools and methods benefit the developers of an independent game?

The question has been answered in the manner of a handbook. The handbook is of general aid to the budding developer of independent games. The handbook is also of use to the professionally trained people. It covers a few reminders on what should be done and what to avoid.

How are games developed in the Independent Game Development environment?

Games within independent development area are developed differently than those done within well-funded triple-A environments. The key differences were not all negative in our findings as there was a lot of creativity brought out within the independent developer communities. Independent development provides the industry with some very different styled games.

3 INDEPENDENT AND TRIPLE-A

This chapter focuses on game developers. There are standards for game development, categorized to independent and to triple-A. These are the standards that divide the enterprise from the hobbyist. The path to follow is similar, yet has different challenges.

3.1 Independent

Independent development processes are free of the burdens of producing the next big thing. The developer is concerned of making the story entertaining and concentrates on polishing game mechanics and playability, than polishing graphics to the maximum. Concentrating on the contents of the game, is where independent development usually goes. Delivering sound emotions through story, not through state of the art graphics.

One of the independent game defining features is the lack of outside help. The lack of funding means that the game’s development has to be very sustainable. The developers of independent games favour the simple approach. The simple approach is three stages; the choice of genre, planning the design and production.

In the field of independent development are also a few perks. Time is not an issue for most of the independent developers. They make their own schedule, developing the game when they have time for it. They still have a work schedule and deadlines. There are also no penalties to dropping a project the developer does not like. There are few strings attached to the game. (Saarikoski 2015, 17-18.)

3.2 Triple-A

A triple-A title is a large, corporation fueled, project to produce the next big selling game. It is an industry led standard, where the game follows a pattern from previous successes. Therefore innovation is somewhat limited in triple-A development.

Triple-A titles are polished graphically. Many artists use time and effort to make everything in the game world look as good as possible. The mechanics are written in huge teams, usually leaving pieces of code and even hilarious bugs in the game on release. The triple-A development is funded by different companies, or one company that has ordered a game to be developed. There is nothing the triple-A development lacks.

The methodology of game development in triple-A environment is strict. Developers have all the tools and assets they need available for their work. They have little creative freedom of their own and must follow strict schedules, as well as deadlines in order to complete their work in time. (Saarikoski 2015, 17-18.)

3.3 Comparison

Indie games are more experimental, than their triple-A counterparts. Indie games experiment more freely with the many effects and flaws.

The development team of an Indie game is often small teams making their first title. The project is amateur work, therefore there are personal touches in the game. Games developed in independent sector, reflect more from the makers, than triple-A titles, which are more impersonal.

The development process becomes more intimate to a smaller group of people, than it does to a larger team. The aforementioned differences are also related to the Indie genre in the manner of them trying to break out of the mainstream triple-A games and use it as a selling point.

An example of a successful independently developed game is Minecraft, which at the year 2000 by its surprising rate of sales. Fifteen million copies sold, has brought the developer profit, worth over fifty million dollars. With such a breakthrough, other well doing independent games became more prominent in their own sales.

Many digital distribution companies, such as Microsoft, have therefore opened more business avenues for independent game developers of all sizes, recognizing the possibility of profits, paving the way for solo entrepreneurs and teams alike. Tripple-A title productions use similar gateways, working closely with the different platform providers (Sony, Microsoft, Nvidia and AMD). (Saarikoski 2015, 16-19)

4 GENRES

This chapter offers information on different genres of games, that are present in both independent and triple-A sectors. Their histories will be briefly mentioned for illustrative purposes and give a little insight to their origins. This chapter also discusses the differences in features within genres. (Egenfeldt-Nielsen, Smith & Tosca 2013, 231-252.)

4.1 Action Games

Action game genre is based on fast paced gameplay mechanics. The player is often given only a second, or less, to react appropriately. Racing, fighting and shooting games all fall under this genre. As such, the action game genre has become more of a super-genre, due to the games gaining more identity in their respective sub-genres. First person shooters that saturate the games market, or hybrids that blur the lines of genres, such as action-adventure games.

For much of the early action game history, the arcade space games were the defining feature, as Defender sold around sixty thousand copies of the game. However, in the year 1980 the fascination of space games would start deteriorating, as new games with different features started pushing out. One of the most notable action games was Pac-Man, in which the player collected little dots and power-ups, while fleeing from the ghosts. Occasionally trying to eat the ghosts, when a power-up was received.

For a more modern example, the ARMA game series started with only vector graphics. However,they were for their time, realistic enough to warrant a special order of the original game by the USA army. The ARMA series of war simulators was created for their special use in training. (Egenfeldt-Nielsen *et al.* 2013, 231-252.)

4.2 Adventure Games

Adventure games, are all about the exploration and wonder. Finding out more about the game world, which depending on the developer can be vast and rich of story. Allowing the players to proceed through the game by their own pace. Not forcing the player to react to events, which dictate the route and speed. In addition, many adventure games feature different methods of problem solving, which appear as environmental based puzzles.

Role-playing games, AKA RPG’s, are one of the most notable adventure games, where the players explore the game world, solving puzzles, interact with creatures, non-player characters and advance a plot. The player’s decisions can affect the game world slightly. These games are heavily reliant on storytelling. Dungeons and Dragons-series games are a good example of such games.

The adventure games has a close connection to process oriented games, with early adventure games having similar control mechanics schemes. These are typically seen in process oriented games. The first game to use advanced commands was Adventure, which was followed by Zork, with a bit more advanced features. Zork also was the product that first became monetized. The two games had one similarity with each other in inspiration and influence, which was derived from Tolkien’s works, as well as the quickly growing popularity of Dungeons and Dragons. Later, Mystery House was the game, which brought this genre some simple graphics and introduced player immersion within the game world. (Egenfeldt-Nielsen *et al.* 2013. 231-252.)

4.3 Strategy Games

Strategy games feature two different sub-genres. Real-time strategy and turn-based strategy games. The main differentiating factor being the time to react to different events, which happen within the game. For example, a turn-based strategy game can take minutes or hours to decide the next move. It therefore resembles classical board games, like chess in nature. Real-time strategy games are fast paced, giving seconds or less to react.

The history of strategy games began from Hammurabi; snippets of code that could be expanded on. Hammurabi concentrated on managing a country's resources. However there was a game called Empire, made in 1978, which became more popular. In Empire, the player use military units, trying to conquer the world. There have been more complex strategy games since, however these two were the roots of strategy games.

Some of the more modern strategy games, like Starcraft, has become iconic games of the genre. Some of these have become games, with competitive tournaments and cash prizes. (Egenfeldt-Nielsen *et al.* 2013, 231-252.)

4.4 Process-Oriented Games

Process-oriented games revolve around the user giving commands to the game and the game reacts to those commands. Part of the game sometimes is about figuring out the right syntax in a right situation, to progress further in the game. An example would be a Multi-User Dungeon, where many players come together to conquer this command based environment.

MUD, a game created at the Essex University, which was called Multi-User Dungeon. This was one of the very first types of games where people interact with each other as well as non-player characters. Form parties where they could go fight monsters and other players together. MUDs however, were restricted to those who had a network access.

MUDs still became a place for hobbyists and academics alike. In later years of the procedural game genre, it has become available to everyone who has internet access. People can access and play these games at leisure, which is visible as an increase of player base in different MUDs. (Egenfeldt-Nielsen et al. 2013, 231-252.)

4.5 Games in Education

The players of educational games must feel in control. The game should also have a fantasy element to keep the game interesting. The players want to be challenged to learn more. These games also make the players curious about the game worlds.

The games are domains of learning through interactivity, learning history or science as they play. That said, there can be more to a game, that makes the player want to learn and experience, than from a textbook on mathematics for example.

Video games aim to give the player a feeling, that they are part of the game’s universe. Being a part of the education process, giving them the motivation to learn, while making learning more enjoyable.

Players can also learn different ways to learn through games. This has opened a multitude of new ways to learn and probing for the most natural way to learn has become a part of game design. Not all people are the same and they learn in different ways.

A textbook gives facts and is usually considered a valid source of information, so too can a game involve concepts, which can be as educational as books. This is just another platform of learning. Games in use by studies should be taken objectively. If a game truly is educational and entertaining, it teaches the player in game. Here in lies a problem, for if the player begins to play solely for entertainment.

There is a rift between learning from the game and playing the game for amusement. The rift between learning and playing causes some players to disbelieve the game and trust their pre-learned material instead. The other players blindly believe in what the game says, instead of critically studying what is said in the game.

Unlike it might seem, teachers can affect the student’s learning in many and wonderful ways through interacting with them within the game universe, giving important information about the tasks, or just the briefing, however when necessary the teacher can also correct possible misperceptions the student might otherwise get from the games.

The teacher is also the one who chooses his or her tools to teach the students with. So, in this sense, the teacher’s part is highly reflective to the use of educational games. A teacher with prior experience might use proven educational material. (Egenfeldt-Nielsen et al. 2013, 231-252.)

5 DEVELOPMENT ROLES

Most beginners will start the project alone, however one might find a group to work with. The group will often have different visions from one’s own, it is important to balance and search for compromise when necessary. There are roles that one must fill, and they are described in this chapter.

The designer is the one who generates ideas and uses their inspiration to make new directions, new concepts and new ideas for mechanics and other development related ideas, however the designer doesn’t always do the designing process as alone as one might think, there might be many designers in a design team working on the same design that they want to present to the others. They are the ones who have to communicate the core idea of the game to the other workers as well as be aware of what is the so called line of life of the game that they have designed to keep the vision’s boundaries clear. The designer also identifies requirements, functionality as well as the contents along with the outlook.

The programmer is the one who creates the code for the game. The most graphics oriented games need code in order to function. However, as active, problem solving and progressive as the programmer’s role is within the team, they too can get stuck on the code. Refining their code over and over again however, is what makes the game in the end. What is visible to the gamer is not the game, but the graphics. It is enough, that everything is functional on the visible level. If it works, that is.

The artist is the person behind the looks of the game. The aesthetic feel of the game brings the sounds and visual assets to life. The artist can also be the coder, the writer or anyone with artistic skills. The most visible work the artist does is the production of visuals for the game. Artist brings the game to life in the audiovisual level.

The manager is the person who will see the project through. The manager will take care of the team, find funding and manage the affairs of the team. Manager handles recruiting, scheduling and following the advancement of the project. It is also the manager’s role to report the progress to the stakeholders of the project.

The team is the work of the manager, and the team’s success is the success of the manager him or herself. The manager has to make the team get along, it can be hard to get a team to work together towards a unified goal. The manager oversees the work process.

Novice developers with no team must rely on his/her own skills to carry the project from start to finish. Working alone, the developer does not have the team to help, nor be a drawback to their work. The developer needs to consider how many roles the developer can fill alone. The role of designer, artist, and coder are hard tasks to do alone. However to make up for it, the commercial and some of the non-commercial game engines have tools to make the process easier. There are ready scripts and artworks, which the developer can use to achieve their goal. There are also varieties of literature that will help a beginning game developer get a deeper understanding of games and their mechanics. (Blackman 2013, 11-12.)

6 PHASES OF DESIGN

The design process is not fully about inspirations, it is a process that iterates over four different phases. The phases are analysis, design, implementation and testing. The following explains what the developer needs to know about the phases. (Manninen 2007, 28-32.)

6.1 Analysis

Analysis is a phase that traverses the whole project. In this phase, the focus should be on understanding where the developer is currently within the work. What is the goal the developer is working towards? The important point is to understand what the developer is trying to solve, or what the developer should try to exploit. Through analysis, the developer can investigate new sources, if they are valid to the current project. Analysis can also help with timing a project. Time critical projects need close attention to deadlines and those deadlines should be upheld as often as possible. Delay is not unheard of in projects, but at times, it is not crucial.

Developers need to know to who the game is built for. A game made for children should not have explicitly mature content. There are rating authorities, who make the decision, whether a game is suitable for children or not. Another important part is the platform the developer is making the game for. A real-time strategy game would work poorly on a mobile phone compared to a personal computer or a console. While analyzing, it is important to the game that it is being analyzed objectively, keeping in mind it is not made for the developer.

During the analysis the developers should also take a look at what the developer can do on their own. The developers could also look for someone who might be able to help them to accomplish their goals. (Gibson 2014, 727-789.)

6.2 Design

It is good to use feedback from players or the genre, to avoid mistakes in game development. Developers will otherwise go for their own ideas of what is best, ignoring the necessary advice. It is easy to make a good looking game, where content and controls are very different from what players are used to. Doing this right is important for success.

Ignoring players this early in development is not a good idea. Players have a lot of experience in gaming and have good points. Players play games they enjoy, not games the developers enjoy. The audience pays for games they like to play, so demoing and taking feedback is a very good idea. (Gibson 2014, 727-789.)

6.2.1 Basic System Mechanics

The basic mechanics in games depend on the game’s genre. The expectations depend on the game play that the genre imposes. There are universal standards that are shared between games within a genre. Sometimes they are in a genre that would not normally include them. Health and mana points commonly found in role playing games can be in first-person shooters and strategy games. Taking damage in first-person shooters might damage the user interface, rather than be shown in numbers.

It is good at times to try new approaches, such as removing the user interface for better immersion to the game. In some games this can become a drawback, because it lacks the information the user interface provides. It can become an inconvenience for the player, so this should always be carefully considered and tested. (Skaff et al. 2012, 167-238.)

A common game mechanic that is used in most game systems is the use of numerical scoring and progression systems. As an example, role-playing game systems utilize the user interface that tracks different values, such as health, currency, mana, experience and other meters. Spendable points within games are used in massively multiplayer online games. There are less point based systems in role-playing games, because they are made to be more immersive.

Another example of a system in games includes a competitive element for their scoring systems used in competitive activity. Systems that are based on competence inspire players to compete against each other for higher scores and ranks. The scores compare individual skills in the game. (Skaff et al. 2012, 71-96.)

6.2.2 Win and Loss conditions

Many games that feature multiple players have a singular winner. However, there are also other kinds of games, where everyone either wins or loses. In games with only a single player, the condition for player victory acts as a goal.

Sometimes the game itself lacks the conditions of victory or loss completely. The Failure and success happens, regardless of player actions. That does not mean that these games are not games at all. Sandbox games often lack failure states, but allow the players to experience, explore and create in game. Games where failure is the only ending, challenge the player to survive as long as possible and to attain as high a score as possible.

Older games tied achieving the victory conditions to the end of the gameplay itself. Modern games that feature an open world, allow the player to continue the gameplay even after the victory condition has been reached. An example of a game without ending condition is Skyrim. Skyrim transitions from a goal oriented quest system to a sandbox type game, where the winning is not the end condition of the game. (Skaff et al. 2012, 71-96.)

6.2.3 Events

There are many different events in games. These events can be categorized into two sets of events, either random or player events. Player events are pushed into activity by the player. Random events happen as they should, randomly. A whole game uses these events to push the player or react to the player.

The usual two types of the same mechanic are that the player is driving the event, while the event is the one driving the game. The other is, the player is driven by the events, the player cannot go back, and have to progress from event to event. Two types of reactions exist, object reactions, either passive, or active event, the other actively reacts, the other waits for a trigger.

Messages between objects are sent after events, modifying the way objects act. Messages inform the main object that an action has been taken, event triggered and the player is ready for the next stage. Messages might also carry functions such as advancing game time, or adjusting the settings otherwise.

The events can be made random and really malevolent to the player at unpredictable times by using the random generation of events. For example, a player is flying through the space the usual route, there is a block that makes them go another route because of a random event occurring. The event driven objects are hard to define, but it can be said that a player object is event driven. The events come from either player controls, or underlying game mechanic. (Thompson 2002, 91-101.)

6.2.4 Basic Narrative

Common structure of a game narrative usually goes along the lines of a teaser, in which the player is introduced briefly to the story, as well as lore. From there the elaboration of the plot continues to the next step, explaining the basics of the narrative, and the lore, until the point of no return.

The point of no return being where the player can no longer go back in the game or their choices. Instead the player has to act upon choices up until the point of escalation. In the escalation part the choices of the player usually come to life and escalate the existing conflict. For example, if the player before has helped the bad wizard, the good wizard will lose the wizard duel, or begins losing it in the escalation.

The duel ending is held in the climax moments of the narrative, where the player is led through the very heart of the story. The player choices are presented and the consequences are revealed. In the resolution part of the narrative everything usually becomes clear to the player. How their actions have changed the world around them and how the game has turned out, how the story ends.

A sequel’s narrative will follow the same path, with different variations to the narration. Eventually everything is explored and the franchise becomes obsolete, if the repetition of the narrative becomes too obvious. (Egenfeldt-Nielsen et al. 2013, 205-213.)

6.2.5 Complexity

Rules can be hindrances to the potential amusement provided by the game, with too many basic rules is hard on the new players to begin with. The game might also have too few advanced rules can frustrate experienced players of the game. The barrier of rules has to be equaled out with the rest of the rules, a good set of rules should be no longer than a page to read through. It is often the case that beginner level players of the game will not be able to progress, if the direction is too vague. Likewise it will make experienced players irritated, if there is a lack of clarity in the advanced rulesets.

Computer based games have advantage over pen and paper ones in the sense of the rules. They rarely require the player to read a set of complex rules before getting into the game itself. In computer based games the computer is the only rule keeper necessary. (Skaff et al. 2012, 167-238.)

6.2.6 Player Motivation

The incentives of playing games vary between playing the game and fun, but are not limited to it. The games bring either mental or physical, aesthetic enjoyment of the game’s well-crafted story or scenery. Getting adrenaline rushes from horror games, or competition in first person shooters. Fleeing from reality and enjoying the social sides of the game itself. Sometimes the developers have a message they want to deliver through their game. A morale or some other such personal motivation, which can add to the rewarding feeling of playing the game.

Other than emotional or information based rewards, there are sometimes also real-world rewards for playing a game. Most common rewards are monetary for competitive and gambling based games. Other rewards can also be gained from playing in a competitive game and winning, such as cards in a card game for example. (Skaff et al. 2012, 71-96.)

6.2.7 Efficiency and Pacing

It is important for developers to keep in mind not to make the players feel like they are wasting their time in playing of the game. To allow for opportunities for the player to choose their own gameplay pace, or even store their progress made during gameplay. This becomes more apparent in in longer games where the game lasts for hours. Options for saving and loading states so that the game can be continued at a later date or even incorporated to the gameplay itself by creating downtime, a lull in action, in sections that allow the player to catch their breath and restock on resources.

Downtime in games is often considered a bad design choice to make by the developers. Excessive gameplay restrictions are often employed to artificially prolong the gameplay, or to promote the use of micro-transactions to skip the downtime. However there are some cases where downtime is a good feature to allow the player to focus on the other aspects of the game. In multiplayer based games downtime helps with socializing, giving time to form alliances with other players. Proper pacing of gameplay segments with proper amount of downtime allows the player to explore. Another, meta-game example with downtime is, that the player can spend their time developing their gameplay strategy. (Skaff et al. 2012, 71-96.)

6.3 Implementation

Implementation phase takes care that the developer implements all the necessary parts. The implementation phase is about getting from the idea to the prototype and finally to game play. At the end of this phase there should be at least one prototype ready for testing.

During the implementation phase it is important to keep an eye on the plans. However, it is necessary that the developer checks if the ideas from design phase work. The mechanics itself might be good does it fit in with the game is a whole another question. It is recommended to store the unused ideas for future use in some other project, if they do not fit in the current project. (Gibson 2014, 727-789.)

6.4 Testing

The testing phase is where outsiders are often involved in the development of the game. The test can be as simple as putting someone outside of the project to sit before the product to play it. This phase is important step to repeat many times to check for errors. Feedback from the testers will help to solve the problems that were found when testing the game.

During this phase the developers should listen to their team, as well as their audience. The development team might have some insights and disagreements to the developer’s plans. Making compromises can in the end lead to a better end product. The audience will report bugs and glitches in the game, if you let them test it. Letting the audience play the game is a good way to get important feedback for fixing the bugs before its final release. (Gibson 2014, 727-789.)

7 PRODUCTION

7.1 Pre-Production

The starting point varies from developer to developer. The first things in the list should be figuring out the basics. It is important for developer to think what they are doing, keep the right people informed and have the right workers in meetings. Keeping everyone involved in the process will help in the brainstorming session.

During the brainstorming session, ideas should not be discussed to an extent, nor should any idea be criticized. When ideas stop coming, the developer needs to be prepared to push out even more ideas themselves to keep the brainstorming session going.

Developers, solo or team, need to often think of the game mechanics. For example, will the game have multiplayer or does it even have the potential to be made into a multiplayer game. If it is a single player specific game, it should not have multiplayer to muddle it up. Learning curve is also one of the cornerstones of the game. Should the game have a steep or easy curve? How high is the entry level to the game’s world?

Next, concept art, meaning the game’s artistic styling, character appearances, enemy weapons, ships, spaceships, and the worlds. In a big team, artists draw up several choices for a brainstorming session, for it is easier to draw the vision and show it to the team, than it is to explain it in words.

Audio elements are the voices of the world. Should every last non-playable character in the game be voice acted or only the main characters? Do the players have their own voice played while they play the game or do the players select a set of voices to use? What ambient background noises should be played in which part of the game and what style of music does the game have? Music often creates the atmosphere of the game, as much as the graphics. (Chandler 2014, 129-153.)

There is the possibility of disagreements. Any team is not harmonious from the start. Developers work with the team, sharing their vision, which might be altered during the brainstorming. To avoid any large rifts between team members, it can be a good idea to define some ground rules and have clear goals for the team. If the vision is not shared by all team members, it might turn from one project, into two.

As problems occur, particularly in a solo production, there are tutorials, demonstrations, tools and helpful people over the internet, who can help overcome the problem. If nothing seems to work, the developer can always change the concept, so that the task becomes unnecessary. (Vuorela 2007, 36-38.)

7.2 Prototyping

Prototyping is an important tool for the developers. Prototyping allows the developers to make design adjustments on the fly. There are different ways of prototyping, one example being paper prototyping. Letting testers work with the prototype can be done just by giving them a slip of paper. Give the testers menus and they will decide where to go from there, as well as how to interact with the options presented. The gained data will then be implemented, resulting in better quality of the product. (Gibson 2014, 727-789.)

Prototyping is a technique used by many developers outside actual game development. It is considered a cheap and easy way to test different mechanics. A prototype is a small test to see if a portion of the game works, before including a possibly unstable piece of code to the first draft. It can be done at any point to determine if the product looks right. It can determine if a mechanic works with the game or not and if any changes should be made to the plans before continuing forwards.

A prototype can be hand written, a computer mode or a test code. A playable prototype of how the mechanics work before implementing it into the game. Prototypes of the graphical user interface, which will help design the menus, health and mana bars or other visual appearances of the game.

Prototypes are quick, otherwise there would be no point to using prototypes. A prototype tests a piece of the game, and it needs to be limited to only testing that, otherwise it risks taking too long to be effective. (Manninen 2007, 174-175.)

Quick prototypes made out of paper and dice can test a logic or random effect and visualize the wanted result clearly. The main advantage here is simplicity, for changing a paper prototype is quick, making it ideal for small tests of logic.

The other thing about paper, can show sequences before computer models are built. Quick drawings placed on the wall, can be quickly rearranged in order to keep the big picture in sight. (Gibson 2014, 125-140.)

7.3 Game Builds

After pre-production, the game begins to develop in builds. It is the first real step of the game development, where the actual coding, graphics, locations, maps, props, rules and game mechanics are built in small prototypes and combined together to form a pre-alpha stage. The game slowly begins to form.

Alpha phase of the game is where all the functionality should already be in the game, locked into place and ready for testing. The only changes after this stage should be level design changes, character model fixes, bug fixes and user interface optimization. After an alpha release, all the bugs and issues are collected, to be fixed in beta.

Beta stage is where one only fixes the bugs and tries not to affect the game anymore, the only fixes that one should have to do in this phase are corrections minor misspellings or game breaking bugs. (Gibson 2014, 89-104.)

8 PRACTICAL PART

In this chapter the instructions to create a working prototype are found. Instructions are presented to guide the creation, including all necessary code. Following these instructions a prototype can be created.

Unity Engine will be a necessary download, which will be available at: https://unity3d.com/get-unity/download. Start by installing Unity Engine to the computer, following the instructions given once the file is downloaded from Unity's homepage and opened.

Open Unity, if there are no existing projects, Unity will prompt to make one. Next click the button where it reads 'new'. Name this project 'prototype', and choose the project saving location. After the new project has opened up, save the scene that Unity has created.

Create the Plane from GameObject menu, it is in the list of 3D Objects with the name Plane. Use a name related to the project when naming the object, often it is called floor, or ground. Reset the position from the gear menu of the Plane object.

Create the PlayerObject from GameObject menu, it is in the list of 3D Objects with the name Sphere. The PlayerObject is presented as 3D model of a Sphere, reset the transform as well. Move the object up from the transform position, set the value at 10 will place it above the plane.

Add a rigid body to the PlayerObject through the menu, after clicking on the PlayerObject to select it. Go to the menu, select Physics and pick the RigidBody component, RigidBody automatically adds the gravity to the object. A working collision prototype is ready, hit play and the sphere should not pass through the plane.

Go to the PlayerObject and add a component from the add component button in the object properties, from the list choose new script. Once clicked it will prompt to name the script, as well as choose the language. Use gear menu and edit script option, add the following script.

```
public float speed;
private Rigidbody bc;
void Start (){bc=GetComponent<Rigidbody>();}
void FixedUpdate()
{float moveHorizontal=Input.GetAxis ("Horizontal");
float moveVertical=Input.GetAxis ("Vertical");
Vector3 movement=new Vector3 (moveHorizontal, 0.0f, moveVertical);
bc.AddForce(movement*speed);}}
```

The script allows to do simple keyboard movement with the arrow keys, which is enough for the prototype. Place value five in the PlayerObject’s speed value, by clicking on the object, the value will be there now. Press play to test, if this works, and how it works.

Implement two cube objects and two duplicates, following the same procedure as with the Sphere, instead choose cube. Make one cube, reset position, place value ten in the scale’s X value so it extends over the playfield. Drag the cube to the edge of the playfield, right-click and select duplicate. Drag the duplicate to the other side of the playfield. Repeat, instead input scale Z as 10, drag in position, duplicate and take it to the other side.

Create a new 3D object, this time select sphere, reset position. Set the scale to 0.5, 0.5 and 0.5, for X, Z, Y values respectively in its inspector window. Duplicate it twice and place them in a position where the player can easily get them.

Go to the inspector of the sphere and modify the tag by clicking the tag list and selecting new. Add in the necessary tag, for this example it will be: bog. The tag will then need to be applied to the two objects made from the tag menu where it exists. Next select the sphere that was left floating high in the air, identifiable from its script PC. Open PC script for editing and add in:

```
void OnTriggerEnter(Collider other)
```

```
   {
               if (other.gameObject.CompareTag ("bog"))
                              {other.gameObject.SetActive (false);}}
```

Which will make the code as a whole appear like:

```
using UnityEngine;
using System.Collections;


public class PC : MonoBehaviour {
               public float speed;
               private Rigidbody bc;
               void Start (){bc=GetComponent<Rigidbody>();}
               void FixedUpdate()
               {float moveHorizontal=Input.GetAxis ("Horizontal");
               float moveVertical=Input.GetAxis ("Vertical");
               Vector3 movement=new Vector3 (moveHorizontal, 0.0f,
moveVertical);
               bc.AddForce(movement*speed);}


               void OnTriggerEnter(Collider other)
   {
               if (other.gameObject.CompareTag ("bog"))
                              {other.gameObject.SetActive (false);}}}
```

Close the editor and select Sphere(1) expand sphere collider, as well as toggle its trigger, after which Sphere(2) is toggled the same way. Run the game and collect the objects and test obstacles.

Winning condition will be text saying "You WIN!", after the PlayerObject has collected the 2 spheres, for this add: using UnityEngine.UI; to the top of our code after: using UnityEngine; and in public class of PC, add: public Text victoryText;

Add to void Start victoryText.text=""; to set the initial value. Return to PC script and add private int count; to introduce the counter.

Go to the void OnTriggerEnter(Collider other) and add into the IF-statement: count=count+1; which will then do the counting. Add into the same IF-statement another IF statement right after, if (count>=2){winText.text="You WIN";}

Right-click the hierarchy menu, select UI and choose text, which will create the text object. Rename the text as victoryText, set the position as 2 in the pos Y after resetting the position from the gear menu. Select the player sphere and drag the victoryText into Victory Text slot. Click play and test the game prototype, it is fully playable and has mechanics to test.

Click the File tab and choose Build Settings, choose the platform, use standalone player for this example. Press add current scene and press build, choose the location of the build, name it, and click save. Open the location and double-click the executable to test if the prototype works. (Unity 2015.)

If the prototype works, then the test is a success, if the prototype does not work, then it is a failure. In this prototype the controls, collectibles, collision, and the winning condition are created and tested. The tests are done within the limitations of a mechanical prototype.

9 DISCUSSION

This thesis was written in part to study the start, middle and end of a game development process and the difference in independent development and triple-A development. Another goal was to make a handbook that describes the process of creating a prototype of a simple game.

Independent development gets the focus of this thesis and concentrates more on ID for practical reasons. A hobbyist can be an independent developer, but describing the coding, graphics and expectations of game mechanics becomes complex.

The research made for independent production and triple-A production doesn't differ on methods, but in the tools used by the developers. It is our opinion that great games can come from both sides.

9.1 Research Findings

This section talks about the findings of the thesis in more detail. The research questions are answered in a summarized form. The conclusion to each research question is given.

It was clear that while a large team of triple-A developers could work out a game in months within their studios, these independent developers would do the same in years. They also usually have very poor tools to work with in comparison to the big industry standard, who seem to have the best tools and sophisticated methods available. In conclusion there are major differences between the game development tools and methods used within the two different fields of the same development.

The need for a handbook became clear by looking for a quick to use handbook that did not require several hours to get started. The developers needed a handbook that was quicker, as well as more efficient in delivering the information.

For the aforementioned reason, the handbook is very short and to the point, instead of a hundred pages. The handbook available for reference in Appendix 1, it is seven pages long, which is enough to cover the basics.

Experienced developers would benefit from a quick read of the general guidelines and practices. The budding developers could construct their own prototypes successfully to see what they really want to develop. In that same manner, it is of use to any developer, if only to refresh their memory.

In independent game development, there are no courses that can be taken, no certain linear path to greatness and glory, but a rocky road of learning. There are schools for developing games. Many independent developers start off by following the word of mouth, or the forum texts of a more experienced developer.

In conclusion, the development in independent environment is different from normal development standards. It is possible for an independent developer to start working in a funded environment as well. We feel that the thesis in overall was a success.

9.2 Conclusions

We learned from creating the prototype how independent environment affects the development of software and what roles are present in the gaming industry. From the Unity game engine we learned that it is easy for the beginning developers to pick up, as we started the work with only a few sources to assist us.

From the literature research and writing we learned that there is many ways to code a game. The easiest of the methods being Unity with so much already available. From Juul’s book we gathered what a game requires. The basics in independent development and the logic behind the mechanics.

The development of games can be fun, because of the creative freedom of what can be created. However we also learned that it is hard work, and requires dedication to get results.

The process of creating the prototype was a great experience for us. It taught us how game prototypes should be created to keep the workflow fluent. Of course some problems with the prototype at the last moments gave us a headache, the prototype data having vanished, which taught us to always make the backups available as soon as possible, and as often as possible.

The handbook was a really fun challenge to make, including the searching for all the necessary data across all the different books, sites, articles, youtube, videos and news coverage on new games development. We had to remove some of the sources to keep the text relevant and accurate.

The comparison between independent development and the triple-A development was tricky, because it is hard to visualize working in a large company, building a game with only a few friends to talk about this with. Many large companies have announced new games for many new platforms, these all have great triple-A titles in development and it feels different to know more of the process.

We believe the thesis is feasible for the beginning game developers as well as advanced developers alike, for some time to come. It will become obsolete as technology advances with time.

As a conclusion, there is new developments in virtual reality games that will come with headsets and new ways of control. This unfortunately is still out of the scope of the thesis, for no independent has yet started, as far as we know on such games.

REFERENCES

Blackman, Sue 2013. Beginning 3D Game Development with Unity 4 All in one, multi-platform game development.

Chandler, Heather Maxwell 2014. The Game Production Handbook 3rd edition.

Chatfield, Tom 2011. Miten pelaaminen mullisti viihdeteollisuuden: HUPI OY.

Egenfeldt-Nielsen, Simon & Smith, Jonas Heide & Tosca, Susana Pajares 2013. Understanding Video Games: The Essential Introduction, second edition.

Extra Credits – Game Design and Analysis. Downloaded November 28, 2015. <https://www.youtube.com/playlist?list=PLB9B0CA00461BB187>

Gibson, Jeremy 2014. Introduction to game Design, Prototyping and Development.

Harviainen, J. Tuomas & Meriläinen, Mikko & Tossavainen, Tommi 2013. Pelikasvattajan Käsikirja: Temmerprint OY.

Juul, Jesper 2005. Half-real: Video Games Between Real Rules and Fictional Worlds.

King, Lucien 2002. Game On, the History and Culture of Videogames: Laurence King Publishing ltd.

Lecky-Thompson, Guy W. 2002. Infinite game universe level design, terrain, and sound.

Lukka, Kari 2000. Konstruktiivinen tutkimusote. Downloaded November 30.11.2015. <http://www.metodix.com/fi/sisallys/01_menetelmat/02_metodiartikkelit/lukka_const_research_app/kooste#9.>

Manninen, Tony, 2007. Pelisuunnittelijan Käsikirja, Ideasta eteenpäin.

Mäyrä, Frans 2008. An Introduction to Game Studies, Games in Culture.

Rohweder, Liisa & Virtanen, Anne 2008. Kohti kestävää kehitystä Pedagoginen lähestymistapa. Opetusministeriön julkaisuja 2008:3. <http://www.minedu.fi/export/sites/default/OPM/Julkaisut/2008/liitteet/opm03.pdf>

Saarikoski, Samuli 2015. Indie-pelien näkyvyys digitaalisissa jakelupalveluissa Tarkastelussa Steam-jakelualusta. Metropolia Ammattikorkeakoulu, Helsinki. Thesis work. Downloaded 28.November.2015. <http://www.theseus.fi/bitstream/handle/10024/90832/opinnaytetyo_SamuliSaarikoski:pdf?sequence=1>

Skaff Elias, George & Garfield, Richard & Gutschera, K.Robert 2012. Characteristics of Games.

Unity – ROLL-A-BALL TUTORIAL. Downloaded November 30, 2015. <http://unity3d.com/learn/tutorials/projects/roll-a-ball/introduction?playlist=17141>

Vuorela, Ville 2007. Pelin Tekijän Käsikirja: BTJ Kustannus.

Wiid, Jan & Diggins, Colin 2009. Marketing Research.

APPENDICES

The Handbook

To aid you in getting started on the project we've provided a set of questions one should ask themselves in order to properly gauge one's limitations and goals for the project. Once the answers to the questions have been made clear to oneself, one may begin the development proper.

- "What is your vision of the project?"
- "What do you want to accomplish with the project?"
- "What obstacles do you expect to encounter during the development?"
- "How ambitious is the project compared to one's skills and vision?

Tool Selection

We have decided to use Unity after reviewing the development progress of simple game mechanics for the game development tool. Unity has no hidden fees as everything is accomplishable with only it's base assets, the 3D models and other such artistic appearance of course needs to be brought from one's own resources, however even in this Unity Asset Store is a viable solution. In Unity Asset Store, one can find material for the use of game development, either for free, royalty or one-time payment later, when using someone else's assets however one must always make sure to read the license agreement of the asset before utilizing it. Due this flexibility of Unity, and the ease of its use, we have decided to use Unity as the example engine of prototype development.

An alternative choice would be to use GameMaker, a proprietary development tool with a free of charge demo version, however the demo places limitations such as limited amounts of screens and objects one could use, and as such we had to put it aside. One major drawback was that the engine is made for 2D game development only.

Other engines we tested were various however Unity was clearly the easiest engine to work with, using our beginner level knowledge of Unity to build a working game mechanic prototype.

Preparation

In this part we will instruct you on how to prepare in order to make a working prototype.

Firstly one must have a rough idea on what kind of a game one wishes to develop and what mechanics are required to be used in order to create and expand the gameplay. In order to find the required elements one should examine other games of similar genre and browse the Unity Store for pre-existing components to use as an example.

For the purposes of this handbook we've chosen to create a simple 3D platformer with object collection elements and as such require the following elements: player actor, horizontal movement, unit collision, game area, collectibles and time limit.

As a start of the development proper, one should make certain that one has installed the development tool of one's choice and made any assets and handbooks that are planned to be used readily available and easy to access. Reusing assets and parts of code from earlier projects can be beneficial to save time and effort, even if only to use as placeholders for the purposes of the prototype.

It is important to note that one should avoid using copyrighted material such as sounds or artwork for one's project as not only the make the creator liable to lawsuits, but are in most cases detrimental to the player's enjoyment. As such, it is advised to find material that fall under public domain, fair use or are royalty free for commercial use and to credit the actual creator for these materials.

Gameplay Mechanics

After the planning stage and asset gathering one should have a grasp on the mechanics that enable the gameplay and what the game itself should look like. One should prepare to make their plans flexible in case of errors, change in the vision of design or adopting more pragmatic development solutions.

Starting off the new project one should first create the ‘stage’ or ‘plane’ on which the gameplay happens, applying constraints such as floors and borders in order to prevent the actors from leaving the gameplay area. Unit collision is a decent way to restrict the actor’s movement, especially in a three dimensional game. It is recommended that one uses easily distinguishable placeholders instead of committing to textures and models for now as they can be added in a later stage devoted to them.

After creating the world itself, it’s time to fill it with actors and create the required mechanics for them to function within the intended area. Having placed the actors onto the game area, one should start with creating the mechanics of movement and assign their player keybinds to commonly used movement-sets such as WASD or the arrow keys. While not necessary, one can assign movement sets for non-player actors such as collectibles or adversaries in order to enable AI-controlled behavior sets.

With movement and collision detection added to the example game prototype it enables the player controlled actor to collide with the actors assigned as collectibles, upon which a mechanic is executed that removes the collectible from the game area and increases the score counter on the user interface accordingly.

For the prototype to count as a rudimentary game it should include player interaction, which was established in the previous step, but also a condition for the gameplay to end such as victory or failure conditions. It should be noted that a victory condition is not a mandatory feature as there are games where there are none and the purpose is to either amass as many collectables or survive before the failure condition is met. However, a failure state should be considered as a mandatory feature as a game without one would be impossible to lose and as such diminishing the gameplay value and repeatability of the game itself.

In the example prototype two end states are used: a victory condition of having collected all the collectibles and a failure state of the timer reaching zero. Additional failure state inducing objects such as hazards and adversaries can be added to spice the gameplay at the developer's discretion.

User Interface and Camera

With gameplay enabling mechanics in place one's attention should be directed at the user interface in order to make the gameplay appealing to the player. One should decide early on when designing an UI for the game on how much information the game should offer to the player and what information is relevant to the gameplay itself, such as hitpoints or ammunition count in games of shooter genre. In more complex games UI clutter should be minimized as much as possible with methods such as tooltips and toggle able UI layers in form of additional menus and ledgers.

The choice of player camera used in the game is one of the more important choices a developer must make as it directly effects on how the game is perceived by the player. In the choosing of appropriate camera mode for the game, the game's genre should be kept in mind as some genres such as shooters can function with multiple camera angles with first person shooters or FPS being the most popular, top down, third person and isometric are also viable, if less used.

Textures, Models and Soundset

The choice of artistic style is one of the features the creator should give a lot of thought as it is a decisive factor on how the game itself will look. Not all game creators have an artistic flair to create gorgeous pieces of artwork, it is still possible for them to decide on an art style that gives the game consistency, character and recognizability by choosing a style that fits the game's theme. While finely crafted artwork of the surroundings and actor models give an impression of a more living game environment and can by itself attract players to try the game, choosing a minimalist or even something as simplistic as stick figures can give the game as much personality.

The use of 3d-models and high resolution textures can cause strain on the game itself as every asset has to be loaded for the game to function, as such it should be kept in mind that the game's performance should be affected as little as possible by graphical quality by keeping the size of the models and files as small as possible. In the case of 2½d games that employ 3d-models there is a trick to reducing the effect of the models on the game that involves not modeling or texturing the side of the model that is not shown to the player at all.

The creation of 2d- and 3d-models can be achieved by using programs such as Blender, K-3D, 3ds Max and Maya. Please consult the guides of the respective programs for more information on their use.

The choosing of sounds and music present in the game follows largely the same rules as artwork, but usually take far less effort as not all sounds have to be custom made for each game. Stock sound libraries offer a large variety of prerecorded sounds for creative use. For creators who wish to create their own sound assets, a simple recording device and common desk utilities such as sheets of paper and pens can be used to create a wide variety of sounds when a little creativity is applied.

Narrative

A compelling story is another feature that is important for most games, even if it is used for an excuse for the player to engage in the gameplay. Even a simple idea such as ‘You are a pirate.’ is sufficient to fill the role of a narrative, due to the player’s expectations on the word ‘pirate’ itself. For heavily narrative focused games such as roleplaying games the story can be even more important than the gameplay itself.

Testing & Release

Once the developer has decided that game is near completion, one should take some time to polish the fluidity of the gameplay by reviewing the game’s code for excessive and/or redundant lines, and spend some time to make certain that every part of the game functions as intended. As a guideline, when testing a game one should play the game with the intention of attempt to break it. While most developers would spend as little time as possible on testing, one should try to avoid falling into laziness of only checking the core features and thoroughly seek game breaking errors and bugs. Only when the game has been completed multiple times without running into any such errors should the game’s release be considered.

Online game sites and mobile entertainment application providers such as Kongregate, Google Store and Armor Games can be excellent locations for one to seek to publish their prospective game on.

Before making the decision whether to publish the game or not, one should ask a few questions to him-/herself:

- “Is the game fun?”

- “Is the game what I (the developer) wanted it to be?”
- “Is this game worthy for the world to see and experience?”
- “Do I (the developer) want the game to be published?”

Even an unpublished game can be used as proof of your own skill and progress as a game developer, even if it was created on a whim or merely to practice. More importantly, it is a stepping stone for the next project that might be the one to get released to the world.

Retrospect

No matter if the game is the first or the hundredth that one develops, after each project one should ask the two questions:

- *“What did I learn from the project?*
- *“What did I not learn from the project?”*

These questions help the developer to assess their experience gained and to move on from the project to the next and start again with a new idea and more experience.