Try and Fail Game Development

Wrestle with your first Game

A.D.Mynx

Forword

I wrote this, for indie-developers. People who want build a game on their own and not buy everything. This is not a recommendation list if you want pay a team and develop them for you. If you want to do pay a lot of money and don't know the basics of game development it really will costs you a lot and the result maybe is absolute not what you expected.

Forget for a moment pre-built engines like Unity, Unreal, Quake SDK,…and so on. The all have really nice tools, really fast progress with very less knowledge, but very soon you will come to a point where you want make some difference in the game. You want make it your game and this is the point where very much game-developers seams to fail.

The SDKs provide a lot of knowledge pre-packed, but no guide how to modify and understand them. Some of the communities really helps you, others seams to only want make money. You start to search on the web for scripts that you can put together, you maybe have already learnt how to script.

You maybe have already made a mod for a game or something like that, but now you have the feeling you know nothing? This is the point where we all started and where I think maybe this guide will help you to build up some knowledge, some basics and some understanding what you need to get to your game.

This book will not teach you to script, to make models, animate or develop them. All of this I will let you open. I maybe suggest a program which I use but this guide should only explain the basic for the understanding what is needed and how you can plan for yourself, by learning from things that I already learnt by mistakes.

For this book I think you already know some about scripting & modeling- & animation-techniques. No worries, I don’t expect much, but everything here in the book you can read more on web-sites and other books.

Chapter 1 - How to Start

Where to start?

The key for you first game is to keep it simple and follow some simple rules, so you not got frustrated scripting and building before you finished.

Try to make simple things, make little steps and always keep on a plan which you can handle. No try to reach the stars or you will burn like Ikarus. Game-development is a really hard business, with very little tolerance for failures. But it's possible to do, nowadays to make good game alone or as a small team.

RULE 1: No hope without scripting knowledge.

Very clearly announcement, what often kills Indie-development teams or ideas. When you have to find a programmer, which "helps" you building your game, you can expect that you have to pay them. Nearly absolute never, you will get a programmer which works for you free and searching for them always is painful. Expect costs 15-50$ per hour for scripting on the engine and 100-150$ per hour for server scripts or services processing. Keep in mind that it can take several days to make solutions for complex features. If you don't make an AAA game with putting much money on it, you have to learn scripting.

Nearly everything can work as game-engine, to know what's best and possible. You need to know what a language can do before choosing one. This seams easy but very often is impossible for a beginner. I recommend that you take a look on their background, for what the origin build was develop and then you have a nice idea what this thing do without putting much time on reading. There are a lot of different possibilities and nearly everything works.

Simple rule (for absolute beginners): JAVA - simple games, VB&C-based languages - big games, SQL - database

I recommend that you use a “standard” programming language. Nearly everything works; it’s you that have the power to do so. I suggest using Microsoft VisualStudio with absolute popular languages like C++, VB.net, C#. Some of the SDKs and pre-build engines use own scripting languages and I absolute not recommend to use this, because you never can switch to another SDK without double your time-input on learning another language again. Some of them tricked you to think you use i.e. C# and you find a code on the forum of a C# developer and it not works on the compiler of your SDK. A programming beginner nearly never can solve this kind of issues, because it’s beyond your skill level. On a standardized programming language, you have a really good base of guides for beginners and nearly always a solution on which you can learn. Most recommend C# because nearly every IT-school learns to basic on this. The most game-engines which you can get are made in C++, but C++ is nearly dying. It’s a very great language, but very complex and a very hard to master. I prefer VB.net, it's on you what you likes most.

RULE 2: Story is not the Core on your first try

Would I say story is not important? Absolute not! I really love a good story, but we are not writing a book, we are developing a game. You really can think about story if you built an engine, most upcoming studios run behind a story with a solution code that they can’t handle, or running wild on a search for a “free” SDK which can make what they want.

Even if your game will be the best game ever, it wouldn’t make any sense if you wrote a AAA-game when nobody buys your idea. If you want sell it , maybe you absolute really got lucky and a studio will buy your idea. But this is very unusual. But your game will definitely not look what you hope for it.

If you need go straight hardcore to story, with less and simple scripting, you should maybe think on a point’n’click-adventure. If it’s 2D, they have static pictures, easy to handle graphic overlays and you can handle this kind of game real easy.

RULE 3: Keep always your resources in mind

Plan always forward and stick to this plan. If you not sure again, what a function should do or what a feature was thought it should be look like, than take a look on this plan, step back and make it again from this point. Make a plan what you can afford, to use time to each part. Try not to make big steps. Make it small but constantly. With the relative time you need, you will think of a game that you can develop, maybe not the perfect game for sure, but hey it works and YOU build it.

For example: You want make a game with armors that the character can put on. Than not make a plan to build full sets with hundreds of pieces, instead make armor as dummy and try to make a script for it with the possibility for more armor pieces. After the script is working and the dummy fits, the graphic-designers (if you are not alone) can do what they want with it, but it has to be the same parameters as the dummy. If there are no resources, you already have one piece of working armor. Later when the game works right you can make the rest of the armors.

The most problem is, that you nearly always alone or in a small team. The team is nearly always this small that every person has a specialization and sometimes a minor skill which can support another part of the team. As less you can manage this skills and as lesser people you are, as harder it is to make a change of this plan. The result nearly always will be that the game never reaches the finish line. You and your team will run in a circle of fixing bugs on the old plan and get overflowed with the new bugs of the new features plus the changes of the new features that has to be developed. This costs time and more time and more time till a point where nothing more goes and the game is cancelled. Really sad, when this happens and all had canned prevented with a simple stick to the plan.

RULE 4: No expansion of features during developing

It's very important that you can see some progress in your game, to not get stuck and lose the way to the finish-line. When you lose it, you will make more than you planned and this can be fatal. Some minor changes don't hurt, but make a plan and stick to it till it is finished. No special features during developing, wrote it down for later, but stick straight to the basic functions. You can't afford to easily make a new feature like AAA-studios do, they have 10 man only working to adjust the code for this, and you *take a look around*? Right, you are alone.

Keep in Mind that an experienced developer can prevent work with foreseen subroutines and binding-in of some (now) "senseless" placeholders, but I think you don't have this level now and it is hard to do this right. This comes after years of scripting and nearly never can be learned.

A lot of new features cost much time and sometimes need a fatal process of restructuring your engine. If you are an experienced developer this isn't this critical, because you split up subroutines wisely and modifiable, as beginner you will sometimes have to kill the whole engine and write it new, cause a change can causes a chain-reaction in code and you easy can lose the thread, if this happens the bugs in your game will overrun you.

RULE 5: Work in accordance of your scripting language

At first think about what you know and what you need to learn. If you want something in your game and you don't know how to do this you need to find a solution and then build it, but always keep in mind that you must follow the rules of the scripting language you use. If you make something that is not conform to this rules your game will causes many bugs or kill itself, maybe only simply this functions will not work even the code is right and you never will get a understanding what you did wrong. On examples ("EXA001 Jumping Cube") you will find an example, which maybe helps for the understanding why we do the hard way

RULE 6: Math mostly isn't the solution in dynamic-systems

When you need a feature of your game, than think by the planning how you possible can solve it. If you are in development with the logarithm you will try to solve it with math, but math isn't always the best solution.

For example: You want build a shooter like game, where you can see the bullet fly's. The bullet authentically will fall down, turns, splitters etc. You can calculated that all mathematically, but be careful that this calculation has run again and again 100 times per second. This causes that your PC is too weak for this massive flow of data, so you have something to do. Think about, if it's absolute necessary to calculate it, or would it reach to make it look like the bullet flies authentically. Maybe you don't need ballistics, maybe a sway of the weapon would to the trick. There are a lot of options to make it faster.

I recommend: As more dynamic, as more graphic solved
=lesser calculation & lesser traffic in network

There are so many data to calculated behind all what the user sees, try to keep it simple and try to hide this "false" flying bullets. In a dynamic system is the screen refresh process already part of a calculating algorithm and when the graphic already refreshes will the bullet fly. The refreshing rate is the relative point in a millisecond where the bullet stands. Change the refresh rate will case wrong bullets fly faster or slower. Find the right time and calculate the right position and daily it with refresh the bullets seams to fly right, but is mathematically totally wrong misplaced. Or you make it more simple, like in most games, that the bullet is "too fast" to be seen.

RULE 7: Think if a team will benefits you, before team-up

I really don't suggest that you try to get a team with no knowledge of team leader skills or a good solid knowledge of (indie-) game development. On the circle of indie developers you are only an absolute beginner. That guys who knows what to do, want to get paid or work on a pro-team. You nearly never can find a good solid team with only an idea. Some of them even try to get in your team to steal your code/engine. There are a lot of platforms which searching team-members, nearly always the same, guess why? (more in chapter

4 about this) If you expect to learn in a team, than you will get very disappointed about it, small-teams are too busy with their stuff and have very rarely time to teach you anything. The result is, that you leave without learnt any and maybe get frustrated

If you try to lead a team, take care of what they know, what they can and maybe what they want to do. Not always your team shares your enthusiasm to your game, or kind of ideas to make something your way. Take care that you always know what's going on and with what software which parts are created. It's kind of useless when you got an absolute pro-designer, but he is working with a 10.000$-Software that your team don't have. If he left the team you lose nearly all models and have very much work to get the old models remodeled, updated and functional again on another modelling program. Rule of thumb: nearly always 2 times switch of modeling software kills nearly every studio.

RULE 8: Fear the Online-Option for you first game

Networking is a special thing; every scripter (should) know the basics of programing a network connection, but to get this working right for an MMO is quite different. The problem is that you need a lot of parameter. A simple Online-Option cans double the scripting time of the game. Plus you can't be sure that you can build it, after all, if you never done it before. Network connections and constant data streams are always risky and you have to be a good developer to solve complex issues if there are any. Online Games uses a very wide spectrum of tricks to hide network failures and package losses, techniques that a beginner don't know and that nowhere stands so you can read them.

RULE 9: Never trust a GDKs/SDKs

Software-Development-Kits (short SDK) and Game-Development-Kits (GDK) are very nice things, as long as they work with you and not against you. All free versions of them (that not come from an AAA-studio) have a very bad documentation. Most of them you will never find the solution for you problem. Sometimes if you have a problem, you can have luck and someone in the forum will help you out. Mostly you only can except the bug or failure and can do nothing about it. As developer they companies expect from you that you can solve the problem of your own. As experience programmer you could, as beginner you will die hard. Stay simple and try not to implement every single SDK that possible is useful for your game. Stay in your league, make small steps that you know what you can expect and go forward with this. Especially files of class DLL have sometimes yearlong development time in it and they are already compiled. Even skilled programmers don't can rebuild or understand all of them. Normally everything what is official published from NVidia (no Betas) and Microsoft works as expected, and you very rarely see a bug.

Chapter 2 - Power of Knowledge (Basics)

Earn basic Knowledge

I recommend trying to make a simple game, which maybe have something to do with the game you want to build. The key is to get a basic version for a test, not the get a full game. Maybe a ball that jumps so you can check a basic 2D-physic is more than a 3D-model which falls through the ground and you don't get why? With a basic model you can gain understand for what you doing. If you use an SKD you only will learn that the handbook says it should stop, but doesn't. A lot of people make failures because they don't know the basics (e.g. hitboxes, physics).

Maybe try to script a 2D-Game before reach for the stars on a 3D-Engine. To master a 3D-Engine or make one self is nearly impossible as beginner, you really need so much understanding and testing to know what to do. It's not the puzzle-parts which are the most problematic; it's more the math behind all them. (e.g.: EXA002 Cube) Most people don't know what complexity of the problem, when you built a wall in a SDK and the player runs through the wall.

Why the hard way when a SDK can do it?

The SDK already has all physics pre-developed; some of them have advanced physic calculations, like ballistics. But what if you want modify them. The prebuilt version not includes the possibility to modify them. Some of the SDKs can be modify but you not will understand why a change of a value make sometimes really bad issues and where this issues comes. The result is an engine what crashes your game; you fall through grounds or run through walls. With your own engine you think out of this box and change all like you want and maybe this can give you a really great possibility to make the game you like and not the (maybe bad working) copy you already seen. You have the change to fix bugs and really can solve them. With the prebuilt SDK you nearly never can do this, because the SDK is to complex, built by a team of developers and you need years to really understand the whole structure of the code (if you can get it), mostly you not even have the source code and you will cruelly die trying without results.

Chapter 3 - Preplanning

Preplanning for a game

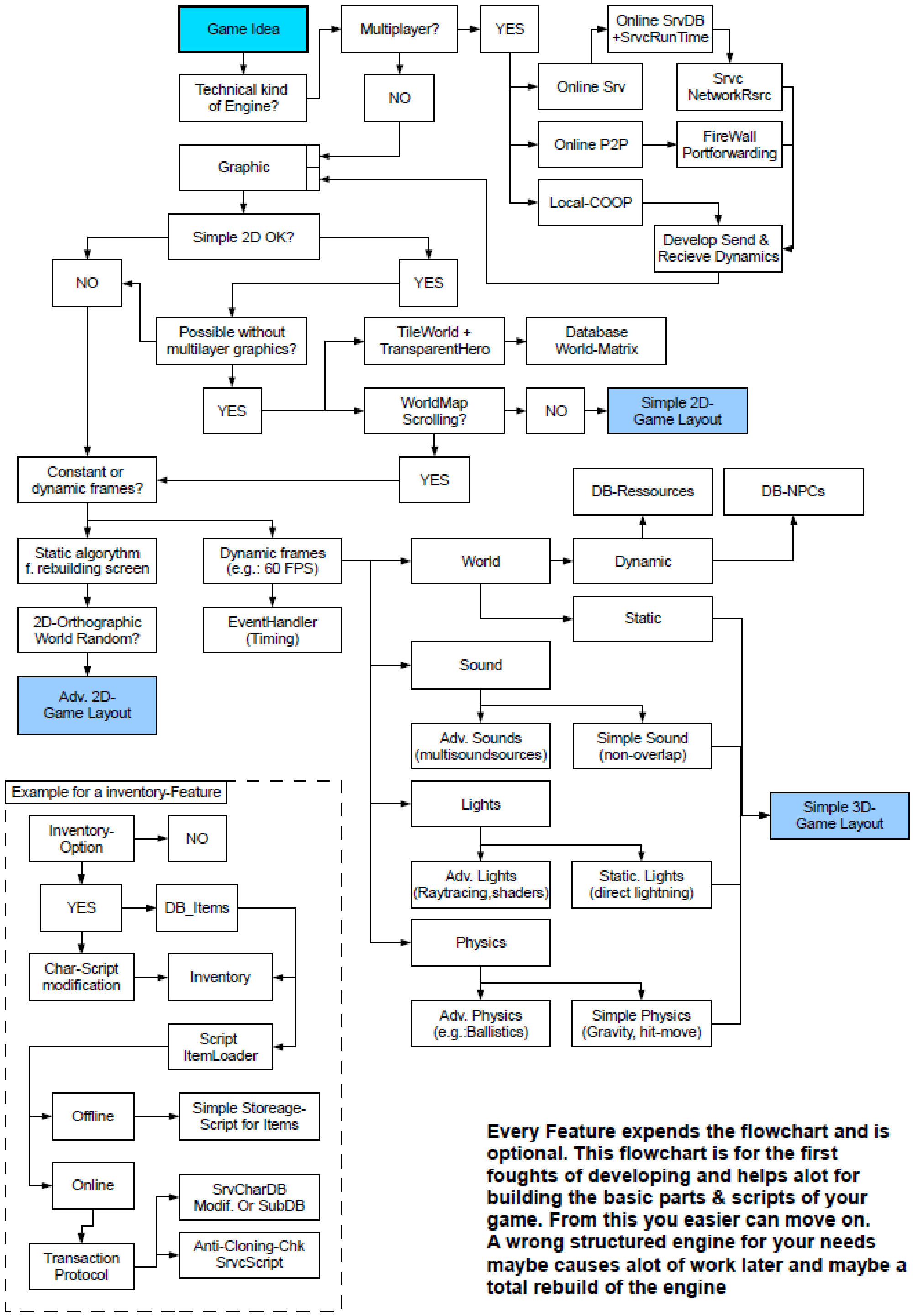
I read much about different kinds of development with different engines; over all nearly every engine (besides a pre-built completed SDK) have the same standards. For better understanding for the development I made a flowchart which should help beginners to step over the first failures (that we make too and you have or will have).

IT-short forms

Srv - Server; Srvc - Service; DB - database

The problem of multiple graphics on same layer

The Question "Possible without multilayer graphics" I tried to explain on Example EXA003. You will find this problem quiet hard to solve on the beginning. Because you really need carefully script the entire world with or without this kind of graphic display, if you make a failure on this, you will get bad screen fractures and sometimes false displayed pictures.



Game Idea
Technical kind of Engine?
Multiplayer?
YES
NO
Online SrvDB +SrvcRunTime
Online Srv
Srvc NetworkRsrc
Online P2P
FireWall Portforwarding
Local-COOP
Develop Send & Recieve Dynamics
Graphic
Simple 2D OK?
NO
YES
Possible without multilayer graphics?
TileWorld + TransparentHero
Database World-Matrix
YES
WorldMap Scrolling?
NO
Simple 2D-Game Layout
YES
Constant or dynamic frames?
Static algorythm f. rebuilding screen
Dynamic frames (e.g.: 60 FPS)
2D-Orthographic World Random?
EventHandler (Timing)
Adv. 2D-Game Layout
DB-Ressources
DB-NPCs
World
Dynamic
Static
Sound
Adv. Sounds (multisoundsources)
Simple Sound (non-overlap)
Lights
Adv. Lights (Raytracing,shaders)
Static. Lights (direct lightning)
Simple 3D-Game Layout
Physics
Adv. Physics (e.g.:Ballistics)
Simple Physics (Gravity, hit-move)
Example for a inventory-Feature
Inventory-Option
NO
YES
DB_Items
Char-Script modification
Inventory
Script ItemLoader
Offline
Simple Storeage-Script for Items
Online
SrvCharDB Modif. Or SubDB
Transaction Protocol
Anti-Cloning-Chk SrvcScript

Every Feature expends the flowchart and is optional. This flowchart is for the first foughts of developing and helps alot for building the basic parts & scripts of your game. From this you easier can move on. A wrong structured engine for your needs maybe causes alot of work later and maybe a total rebuild of the engine

Chapter 4 - 3D: The Beginners Execution

Dynamic frames are very hard to handle, without good team. As soon as you hit this point, you are beyond 1.000-3.000 hours for your engine if you build it complete new.

Often Teams skip a few hundred hours with using an open-source engine with basic adjustments, but as soon you need only a minor modification of this, you easy can lost the won time again.

This point is king-sized entry on math and scripting - very hard to handle. You get a lot of scripts and Open-Source for this, tons of books, examples and freeware, but to get it work and put all right together, you really have to know what you are doing.

Simple the basics you need to learn for this are gigantic. Bases on what you want to build, you need to know exactly what can be done in what manner, to what time and what is not. Every single failure in calculation causes hundreds of failures per second. Even if you catch all errors and hide them, your display-frames will drop drastically. You often know this on bad developed games, where the high-end PC isn't good enough to get it stable.

The next thing is that you really have to know programming well. This is not simply put snippets together and give it a try, very simple (sometimes with only one code) you can shoot the whole system.

Mostly this kind of displaying is not very popular in indie-development, because you nearly always only can fail to make a good game. If you use a Prebuild SDK for this, you can make something, but to make it good you have to make all new and this is hard.

You have not very much option. Direct3D (also the complete pack known as DirectX) and OpenGL are the most used, simple you can find a good base to build you first models with that. To get them together will be harder.

Now you really fast find the difference between 2D and to a high dynamic environment. Now you have to think in numbers, and timelines instead of code-lines. You not so easy as before can import a character, because there is no import. You have to build one, with the full spectrum of function. There is no import-function for animation, you have to make all yourself and every single step must be in the right place and in the right time-step or it fails. That requires a lot of knowledge, skill and time.

If you start a dynamic frame environment is like open the eyes, sitting on a river. We don't can stop the flow of the water, same as the stream of the frame set. We can only close the application, like closing the eyes. If we put something in this stream, we need to load and place it.

For Example: You build a world with models on it (like a house) and want to put it in this dynamic frameset. You have to load the saved model to coordinates X-, Y-, Z-axis plus the angle or reference points. This you have to put on a loading screen, because the script only has this one action. Remember that this whole model has no collision detection and you can run/fall through it, if you not scripted it.

If we want join the world and we want do this with a character, the character have much data in it. The player must be a model with a zero-point near the ground, but over the ground. If it is zero the model "player" will no detect collision and fall in the ground. If you tried SDKs you already know this phenomenon by some SDKs. The character must now join this world on a spawn point.

If you done all right, you now have nearly 1.000-2.000 lines of code, some hours/days/weeks worked and a simple model standing on a simple world. Now it can move, no lights, no shaders, no materials, no effects and absolute no features. Only the basic to make the next steps for you game, so you can start the design of the world.

Only after you finished this nearly unbearable project, you can start to make some models, paint them, and start to create a world. Remember that this all isn't possible yet. For everything you have to script the parts. The script parts for loading textures on a building, for 3D-geometric paths to detect the hitbox in the world and so on.

Maybe this all reads quiet frustrating and I really must say my first project with this kind was, but I don't want to make you false hope. If you really go to 3D with all features you have to do this.

If you go 3D on an SDK/GDK you have the first parts all finished, but you not can go any further. Everything that makes your game, your game is stuck in the pre-build piece of software. If some bugs come up, you have to except that. If you want make a new feature and the engine dislike this, you can't do anything against it.

That's the money making trick of "free" Engines. To make it good you need to pay a lot of money, for there development time.

If you make the big failure as beginner to grab an Open-Source-Engine I must disappoint you again. These Engines are not competitors to Unreal and so on. Free Engines are so bad documented that others makes money again, with "helping" (=selling), the missing information. Ironic that this the same people who not keep records.

Chapter 5 - Team & Rolls

What rolls exists for game-development team?

There are many different views, as larger a team get as more sub-rolls a team have. Take care that every one of them is some kind of a specialist with own skills and own programs, I recommend a standardized list for my teams than seams quiet helpful, maybe this will help you too. Please take care that is a very rough prognosis for nearly every game and not exact for a specific type of game. With special feature this can turn completely

	Skill	**Time required**	**Available free**	**Available paid**
Animator	+	-	-	++
Engine Designer	+++	++++	----	++++
Graphic Designer	+	++	+++	+++
GUI Artist	++	+	+	+
Lead Game Designer	o	o	o	o
Level Designer	+	+	o	+
Modeler	-	++	++	+++
Scripter / Programmer	+	++	o	+++
Sound Artist	+	++	+	+++
Storywriter	--	-	++++	++++
Tester	--	+	++++	++++
Tutorial Engineer	--	+	--	o
Database Engineer	+	+++	----	+
Webmaster	-	+++	+	++++
Learning / Teachers	--	++	o	++++

My experiences with team-work and recruited rolls

Animators: Animators are very rare. The most of them come from art-schools and some of them a very quick by animate your models. If there are not too many and too complex animations, I recommend to buy them as service when you need them. A model that is OK designed and need a walking animation (like a character) can be animated by a good animator in nearly 1 hour; self a good modeler can't do this under 8 hours. Some of them will work for free, when you use their name as main animator. This seams quiet not be a hard job to get an animation to run, but mostly I have very good experience with this eccentric kind of people. I often could really fast can implemented there animations because during he made to animation in/with my model I could implement it and fast get a nearly seamless (and because that) fast implementation to my games.

Engine Designers: Short - There are absolute no free Engine-designers that will help you. Even if there will be one, they never will build your engine for free. Most of people who say they are Engine-Designers have only experience in using an engine. You can use someone of them if you want use a SDK and need some experience. This can come in handy if you want use parts of the Unreal-Engine and you found someone that have the knowledge to give advices so you can build your engine faster.

Tutorial Engineer: Understated and very important. I good Tutorial can be the card for success, even they game is not complete finished. If the tutorial is right and the first impression is good, you have the first part of building a community. Everyone knows minimum 1 game, that you hate because the tutorial was painful to play and absolute no skip button for it.

Graphic Designers: This people are very easy to find. Nearly everyone that has Photoshop or Paint can make a texture; the problem is to keep them in your team. To get a good graphic design of your game, you need to stay in a line, which gives the game its look. If you change to often the game will look like a patchwork and seems odd. Depends on the size of your team, you should make on graphic designer to the leader and this one should stay in team as long as it takes to develop the game. Other designers in your team should try to copy this look and make it similar to it.

GUI Designers: Seams not really important but I suggest giving someone, that have experience on this a try. Sometimes I was really impressed how handy some people are by creating a better GUI then I and this in quicker time. The GUI is always a hard part of the game and it's always hard to say how to come to a good GUI. Some say there should X-buttons and bars on the screen and give this information to a GUI designer who creates a layout for it. The layout takes than the graphic designers and gives the GUI your style. Others say we make the possibility so the player can adjust his GUI (sometimes completely). Others make the GUI very last with the testers of the game. Most of them work for free, not much software requirements. Sometimes Paint is enough for a layout.

Lead Game Designer: Only recommended if you buy a lot of stuff for the team need and someone to coordinate the processes. Mostly this is your main-job in a team. I think you know self that it's not wise to give the lead out your hand for your game.

Level Designer: Sometimes good, sometimes bad. Normally it's recommended that the graphic designers work close with the level designers, this gives the game depth and atmosphere. If you want a close atmosphere for your game it would be wise to take someone of them in the team. As single person you nearly never will spend much time on this detail, especially if you wrote a code.

Modeler: One of the really hardest to choose, simplest to get and often really quick replaced persons in your team. Nearly every "Modeler" has his own program and there are very much modeling-software on the market. Some free, some very expensive. Whatever you search, you will not really get that modeler you need. If they use the professional Tools the models will be hard to import. The most problem is that you have a very large problem to put an awesome model to your game. Every modeling-software you have to need an adapter for your engine. Often you will get freeware modeler that learn with Software like Blender, but this is mostly every very use-less for your Game, beside you program in Blender compatible en-gine. The most problem is that you very hard can connect with this software, because programs like Blender are for art and cinematic and not for developing games.

Programmers: They are (self-chosen) gods and everyone needs them. The truth is that there only 2 kinds of programmers in team-search, that one that can script and want get paid; normally they are long there and make good money with this. And the other kind are beginners like yourself or guys who try everything to get free codes and stuff for their own game. A very small part of programmers really will help you out, by given you simple code snippets, but as soon you need more work from them, they are gone.

Sound-Artist: Take care of them very well and recheck every sound you use twice. Some of them use stolen tracks from the internet. Some of them think they are doing the right thing with mixing tracks with own sounds and republish it as there sound. This really can cost you, when you for violating license of the origin owner. Maybe you shouldn't recruit one of this, till it's really necessary. Some bands give tracks for free; over sounds for regular uses can be downloaded too. Take care to have a warrant of the owner to use this tracks

Storywriter: Short - there are on every edge. You always will find a person who has an idea for a game. Sometimes there is no special storywriter in the team and often, the lead designer wrote the story and quest itself. Storywriter can be useful if your game have a lot of story branches and ability to choose options, but therefore you more need a (book-) writer then someone in a forum. This people are mostly rare and they need work close with scripter and graphic designers together. With this combination, they game will more an artwork.

Tester: Mostly easy to find, but very hard to get good people that you can trust to make to game go further. Most of the "Tester" is kids & students that want play a free game; it happen quiet often that the feedback of this user group is not helpful for your game. If you want a tester which you can trust, it must be one in the team, which makes the same routines more than 100 times before release, has a checklist and wrote bug-reports. Or you program a system which tracks tester and give you a report what they have done and if they like it, but this solution is very time intensive to develop.

Database Engineer: Short - There are no free ones. You will never find a free database engineer, because they are all working on projects for companies. Every database that you will use is on SQL-base, what nearly every company has, or on simple self-build databases, like LIB, DLL etc. which make every developer itself. If you want make a really good fast database (e.g.: for an MMO), than you will have to read a lot and build itself or pay someone that makes it for you.

Webmasters: Some of the Webmasters can be very useful, but my experience is that mostly the free Webmasters make a site on a free homepage-tool and after a few days/weeks there interest is too low to go forward and the site orphaned. I don't recommend using a cheap designed, free site, when you want sell your game or stay in touch with the community. Sometimes when games get popular you get a good chance, that some of your fans, will help you out with Web-design and you can host them self. It's hard to say who you can trust on this. But for a successful game it necessary to have a good Homepage and Forum. More important is to say "an ACTIVE Forum" where really even once a day one checks for questions of the community. You will get to a point where you not have time to be really every day in your forum and therefor you need a webmaster.

Teachers: On longer project times it's necessary to conserve the information and teach this to newer members, when a team-member change. If this happens often you need someone to take care of this. I not recommend that you teach them until it's only you which have the required information. You have too much to do to model, make bug fixes or patches.

Chapter X - Examples

EXA001 - Jumping Cube

I have a simple cube and want to make it “jump”. First of all declare what this should do? Some SDKs want you to draw a cube a line. Click on the cube, set player and Start. Let’s see what this means for non SDK-built:

- The cube und ground have size and as player a hitbox
  - -> Collusion calculation relative size-point box script, besides that the box will fall through the ground
- Make the world moving
  - -> create timer/ticker etc. (equal you script language) that rebuilds the scene every X-Millisecond. Besides that your box will move nowhere even if your simple physic script is right.
- The player
  - -> starting point is relative to zero
  - -> Key bindings have to be set. Keep in mind that every key which should cause a movement have to be bind manually. We are not speaking of configuration-files or something. Only basics arrow movement or in our case the pressing of space to jump
  - -> add gravity script / correction axis of box movement. Keep in mind that we are talking about simple physics and not advanced to box will not turn it only fall simple in one axis. To make it turn you really need deep thinking. We only need for this test one simple command that causes the box to fall
  - -> generate a script which detects to hit of both elements

EXA002 - Brainstorming Cube Hitpoints

For example: A cube has 4 edges, every edge a relative point with a possible hitpoint. If you want make the cube turn your script have to calculate every possible hit every time it moves. If you let them round (integer = 1) you have by 100 pixel 100 possible hitpoints and on a turn in an angle to X- and Y-axis, every X milliseconds. You maybe know right here it start to get complex in math. If you now try this in 3D you have alone on a plain surface 100x100 hitpoints and that multiplicity with 3 axes. If you than try to flip it, you have 3 angles in room coordinates with beginning and end coordinates every X milliseconds. Why we need calculated every point? This is needed if anything hit the surface the script can detect the hit. When not, the hit will be not detected and no script will work to stop the object it hits the surface.

STOP! STOP! STOP! STOP! STOP!

Don't get frustrated at this point, it's all possible, but keep in mind that this is a lot of thinking and it's only a cube without animations. Because of this complexity you always have to think forward and always try it simple. Yes you can calculate everything, but is it always the solution? Sometimes a make-it-look-working is better than a double-value-calculated, because else you nearly always need a server or supercomputer to calculate your game, even if there are only simple blocks.

EXA003 - Brainstorming Multilayer-Graphics

Multilayer Graphics are some hard to understand as beginner, because a lot of programs do this automatically. A Multilayer-graphic-overlay is a system which let you pit several pictures on another and when they are transparent as background you can look through it. The problem in developing is this doesn't works, like often in forum, imagined as glass.

When you try to lay a transparent picture on a gray background, then the surface will by gray. When you lay a transparent picture with the red dot, on a surface and then the transparent picture again on the top, the picture will be gray and the picture with the dot is gone. Why? Simple but quiet not so simple.

In development every graphic object has an ID and a parent. The parent of the first picture is the background. When you set a picture to "transparent" this means 'get the background', but this isn't what we want, we want to red dot and then the background. So we have to make a script we load the master of the picture above the background and then the top-picture with the master of the second picture.

This seams a lot of complexity on it, but its quiet simple and you will use it often, for a 2D-dynamic world. You will need this for the reloading of screen by a moving object. For example a character which is centered on the screen and move around in a dungeon, or a world that should be scrollable.

If you don't use this only one picture can be shown a time. This is maybe known in some games, when the character moves on a tile of a dungeon and the item vanished. Then maybe the developers wanted this.