# Experience Level API – general overview & features

There are two versions of the API:

* Basic version: namespace ExperienceLevelAPI\_basic  
  Class:
  + ExperienceLevel.cs
* Expanded version: namespace ExperienceLevelAPI  
  Classes:
  + ExperienceSystem.cs
  + LevelModel.cs

## ExperienceLevelAPI\_basic

This is a basic version of the API, consisting of a single class ExperienceLevel.cs, where the Experience and the Level always have a **linear correlation**:

1 × Level = 100 × Experience

The amount of experience points adding up to 1 level could easily be changed by feeding a long value to the overloaded constructor ExperienceLevel(long levelGap), resulting in

levelGap × Level = Experience

## ExperienceLevelAPI

This is an extended version of the API, consisting of two classes and an enum

* Class LevelModel.cs  
  This class defines the way the levels scoring system is set up. Apart from a linear one, levels can have different correlations to the amount of Experience points. These are the implemented possibilities:
  + Linear: a × Level = Experience
  + Polynomial: a × Levelb = Experience
  + Exponential: a × bLevel = Experience
  + Manual/arbitrary: Level thresholds are defined manually with an array of elements, starting with 0, and containing as many elements as there are levels. (We assume to always start with level 0 and experience points 0)
* For readability purposes, these scoring systems are labeled in an enum Upgradetype
* Class ExperienceSystem.cs  
  This class contains all the methods and fields directly related to the experience-level conversion, plus some helper methods to expose values for private fields. The class works in close collaboration with the LevelModel class.

# Basic api – detailed view

## ExperienceLevel.cs – members

|  |  |
| --- | --- |
| totalExperience : long | Total amount of experience gained since start of the game |
| experience : long | Current experience, since last level-up |
| level : long | Current level |
| levelGap : long | Amount of experience in between levels |
| *AddExperience(long) : void* | Add certain amount of experience points When enough points are collected jump one or more levels up |
| *ExperienceDelta(long) : long* | Get experience delta between current experience and arbitrary level value (always returns a positive value due to Math.Abs) |
| *ExperienceLevel()* | Constructor |
| *ExperienceLevel(long)* | Overloaded Constructor, for variable linear progression |
| *ExperienceTillLevelUp() : long* | Get Experience needed till the next level-up |
| *ExperienceToLevel(long) : long* | Convert arbitrary experience value to corresponding level value |
| *GetCurrentExperience() : long* | Get current experience, starting from latest achieved level |
| *GetLevel() : long* | Get current level |
| *GetTotalExperience() : long* | Get total experience gained since start of the game |
| *LevelToExperience(long) : long* | Convert arbitrary level value to corresponding experience value |
| *nextLevel() : void* | Private method invoked by AddExperience to jump to next level |
| *ProgressToNextLevel() : long* | Get percentual experience progress between last and next level-up |
| *ResetExperience() : void* | Reset totalExperience to the last level-up, reset current experience to 0 |
| *SetLevel(long) : void* | Set the level to an arbitrary value, set totalExperience to corresponding value and set current experience to 0 |
| *SubtractExperience(long) : void* | Subtract certain amount of experience points |

# EXTENDED API – Detailed view

## LevelModel.cs – members

|  |  |
| --- | --- |
| multiplier : long | Multiplying factor for level-experience correlation (linear/polynomial/exponential) |
| baseNum : long | Base for exponential level-experience correlation |
| power : long | Power for polynomial level-experience correlation |
| gaps : long[] | Array of manually defined level thresholds |
| Type : UpgradeType (enum value) | API defined type of level Model |
| *GetLevel(long) : long* | Get the current level according to UpgradeType Type |
| *LevelModel(long)* | Constructor for Linear LevelModel |
| *LevelModel(long, long, UpgradeType)* | Overloaded Constructor for polynomial and exponential LevelModel |
| *LevelModel(long[])* | Overloaded Constructor for manual LevelModel definition |
| *XPNeededForLevel(long) : long* | Get the number of experience points to reach a certain level, according to UpgradeType |

## ExperienceSystem – members

|  |  |
| --- | --- |
| totalExperience : long | Total amount of experience gained since start of the game |
| experience : long | Current experience, since last level-up |
| level : long | Current level |
| levelGap : long | Amount of experience in between levels |
| *AddExperience(long, LevelModel) : void* | Add certain amount of experience points When enough points are collected according to chosen LevelModel jump one or more levels up |
| *ExperienceDelta(long, LevelModel) : long* | Get experience delta between current experience and arbitrary level value (always return positive value due to Math.Abs) |
| *ExperienceSystem(LevelModel)* | Constructor |
| *ExperienceTillLevelUp(LevelModel) : long* | Get Experience needed till the next level-up, according to chosen LevelModel |
| *ExperienceToLevel(long, LevelModel) : long* | Convert arbitrary experience value to corresponding level value, according to chosen LevelModel |
| *GetCurrentExperience() : long* | Get current experience, starting from latest achieved level |
| *GetLevel() : long* | Get current level |
| *GetLevelGap() : long* | Get current levelGap |
| *GetTotalExperience() : long* | Get total experience gained since start of the game |
| *LevelToExperience(long, LevelModel) : long* | Convert arbitrary level value to corresponding experience value, according to chosen LevelModel |
| *nextLevel(LevelModel) : void* | Private method invoked by AddExperience to jump to next level |
| *ProgressToNextLevel(LevelModel) : long* | Get percentual experience progress between last and next level-up |
| *ResetExperience() : void* | Reset totalExperience to the last level-up, reset current experience to 0 |
| *SetLevel(long, LevelModel) : void* | Set the level to an arbitrary value, set totalExperience to corresponding value and set current experience to 0 |
| *SubtractExperience(long) : void* | Subtract certain amount of experience points |
| *UpdateLevelGap(LevelModel) : void* | Private method invoked by nextLevel to update levelGap for newly acquired level |

# potential features & improvements

* Define and use a maximum number of defined levels
* Define a starting level and experience points (in constructors)
* Validation for array of level thresholds
* Experience delta always returns positive value, due to added Math.Abs; could be discussed if this is desired behavior
* Add event when next level has been reached to ease displaying message

# Demo application: ball ‘maze’

I included a small Unity project where the user has to roll a ball and touch blue or red bars to gain or lose XP.  
The scores and statistics as calculated by the class (in this case the basic one) will then be shown in the UI.  
Simply use the arrow or wasd keys to move the ball around the level.  
  
Screenshot:  
  


# Test cases

Included in the Visual Studio solutions are also Test projects for each Solution (basic and extended).   
An excel sheet with the Experience-Level scales is also provided to make it easier to verify the assertions.