

Game economics and balancing

Understanding game design and the games business through math

Aalto University, Game Analysis course

Perttu Hämäläinen 2020

What this is not about

- The usual game and graphics programming math (e.g., linear algebra, transform matrices, quaternions, ballistics, physics simulation...)

Part 1: the business

Today's goal: Get to know Colab/Jupyter notebooks for game simulations and analytics

- Jupyter notebooks are an easy to use browser-based Python programming environment. Colab is Google's Jupyter service with some UI improvements.
- Python and the Numpy & Scipy packages are perhaps the most popular data science and analytics tools
- No need to install anything, the notebooks can be run on Google's Colab cloud service
- Previously, we used Excel sheets, but many tasks are much easier in Python than Excel, and as a university, we need to look forward and introduce students to emerging tools
- If you want to be an economy or system designer in the industry, be prepared to also work in Excel or Google sheets. I'm still providing the course's old Excels for reference.
- Materials here: <https://github.com/PerttuHamalainen/GameAnalysis>



Dynamics, again

Motivation

- Competition is fierce. 1000+ new apps every day.
- Game developers must think statistically and optimize their chances to monetize and stay alive
- Think of chains of random events, e.g., the acquisition and monetization funnel:

$$p(\text{revenue}) = p(\text{eyeballs}) * p(\text{install}) * p(\text{engage}) * p(\text{pay})$$

- Total probability is low if even one factor is low. At every step, you lose a portion of players. For example, optimizing App Store screenshots and icon affects $p(\text{install})$. If you achieve a 10% increase in retention at every step, above, total number of paying customers increases by 46%.

Key concept: The power of recursion

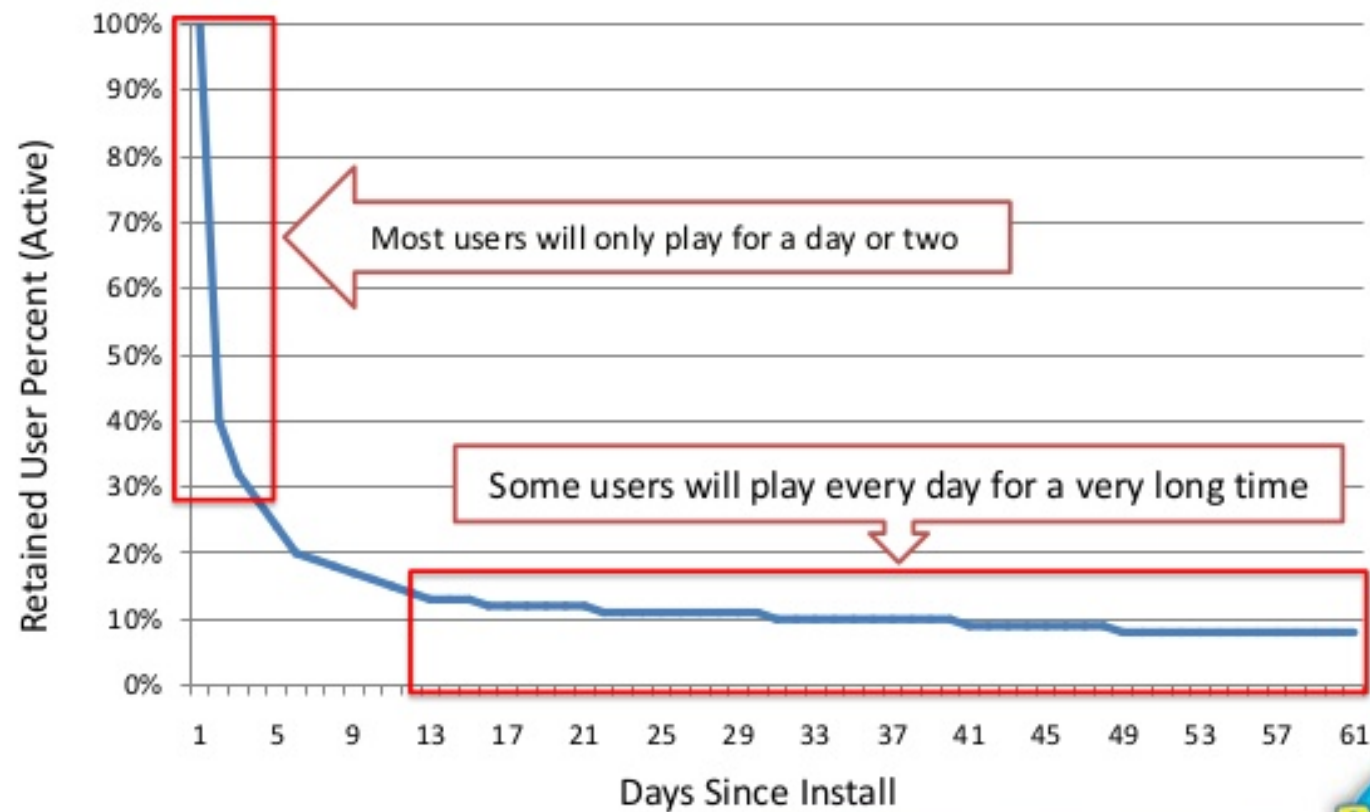
- Due to recursive relationships (feedback connections), small changes in attributes (e.g. retention) can have huge impacts (e.g., on long-term revenue)
- Recursion leads to exponential behavior (decay or growth)



Exponential decay: $u(t+1) = u(t) - p_{\text{churn}} u(t)$

F2P Retention

Tracking a user's in game activity every day since first install.





Exponential decay: $u(t+1) = u(t) - p_{\text{churn}} u(t)$

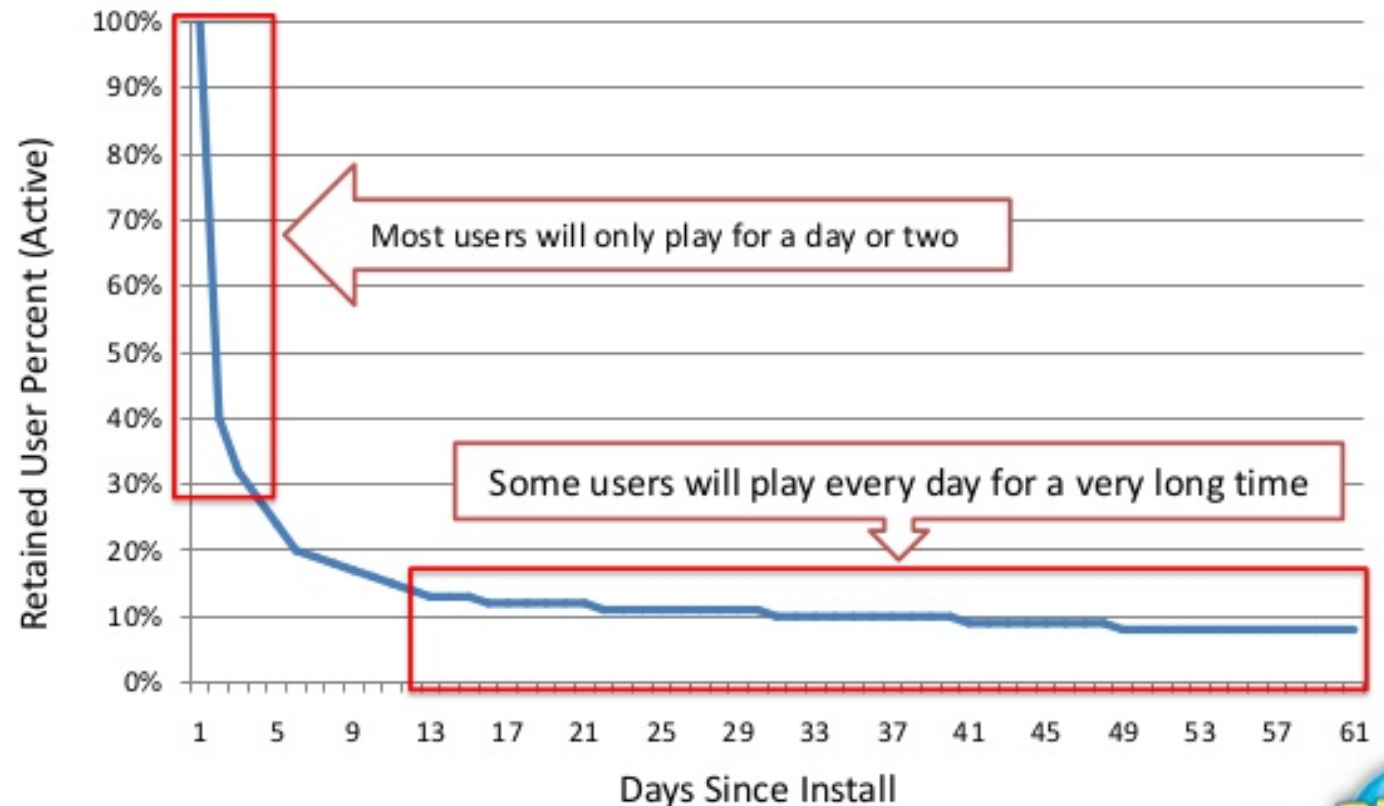
Notation:

$u(t)$ Number of
players at time t

p_{churn} Probability of
player quitting
(percentage of
churned players
divided by 100)

F2P Retention

Tracking a user's in game activity every day since first install.





Exponential decay: $u(t+1) = u(t) - p_{\text{churn}} u(t)$

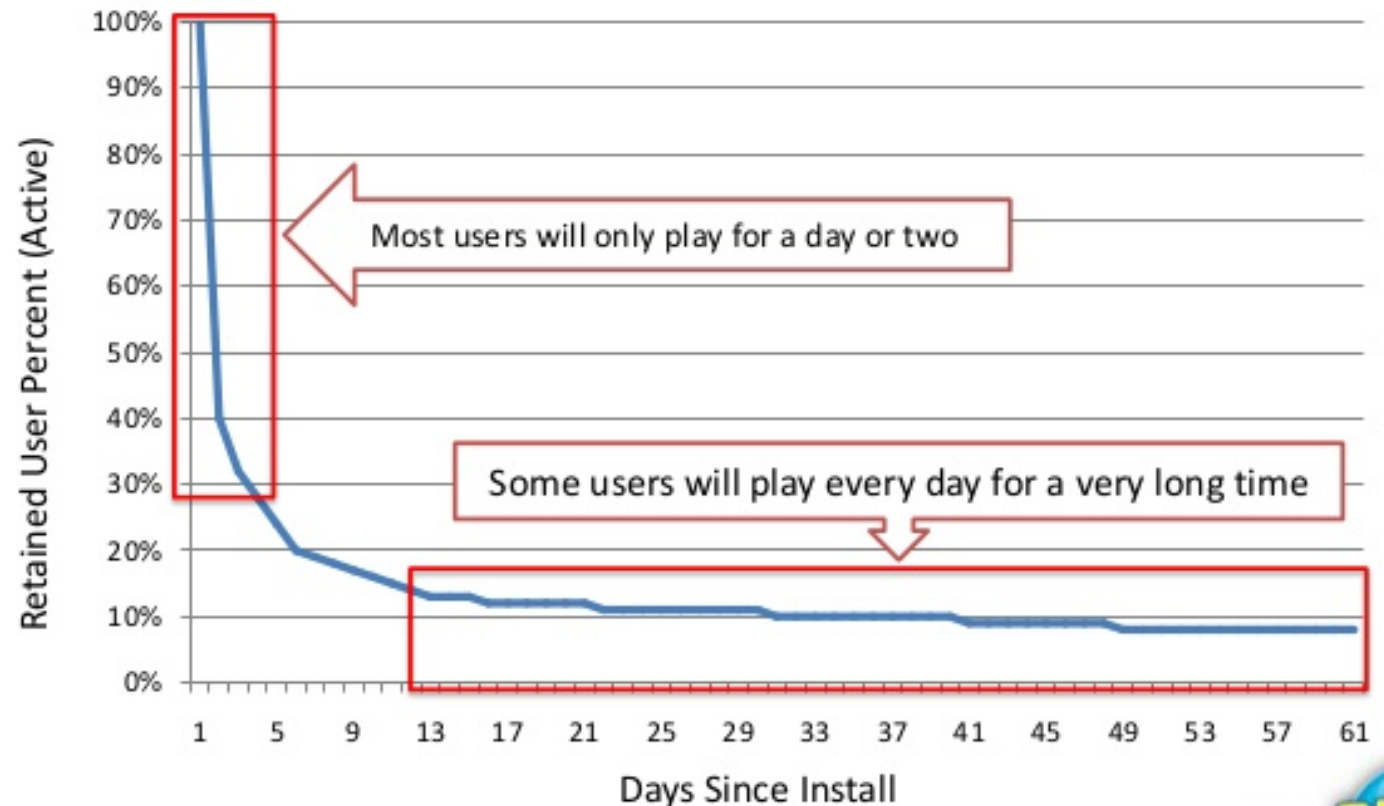
Alternatively:

$$u(t+1) = p_{\text{retention}} u(t),$$

$$p_{\text{retention}} = 1 - p_{\text{churn}}$$

F2P Retention

Tracking a user's in game activity every day since first install.





Exponential decay: $u(t+1) = u(t) - p_{\text{churn}} u(t)$

Alternatively:

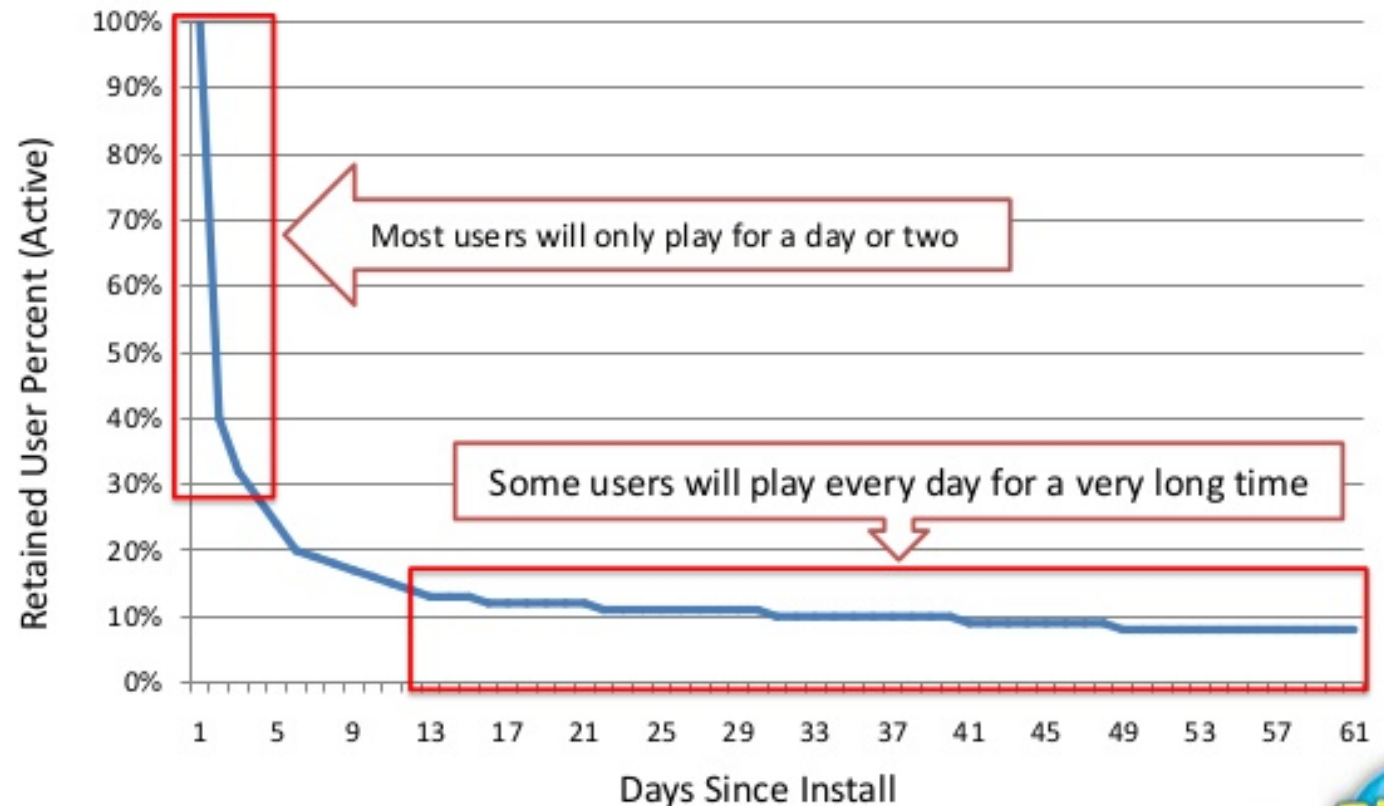
$$u(t+1) = p_{\text{retention}} u(t),$$

$$p_{\text{retention}} = 1 - p_{\text{churn}}$$

Nonlinear because the number of players depends on itself recursively and *multiplicatively*, instead of, say, $u(t+1) = u(t) - 1$

F2P Retention

Tracking a user's in game activity every day since first install.





Model accuracy?

- In real games, churn is often higher for the first day
- Bad tutorial/onboarding can easily make you lose most of your players
- On the other hand, the more time one has already invested in a game, the more committed one might be to continue, provided that the game provides meaningful progress
- If you use this kind of model for making predictions based on your initial player data, measure the churn rate over multiple days, maybe without taking the first few days into account.

Retention outside free-to-play

F2P player churn is dramatic: With zero initial investment, players can easily switch to another game.

Data from paid games can be quite different.

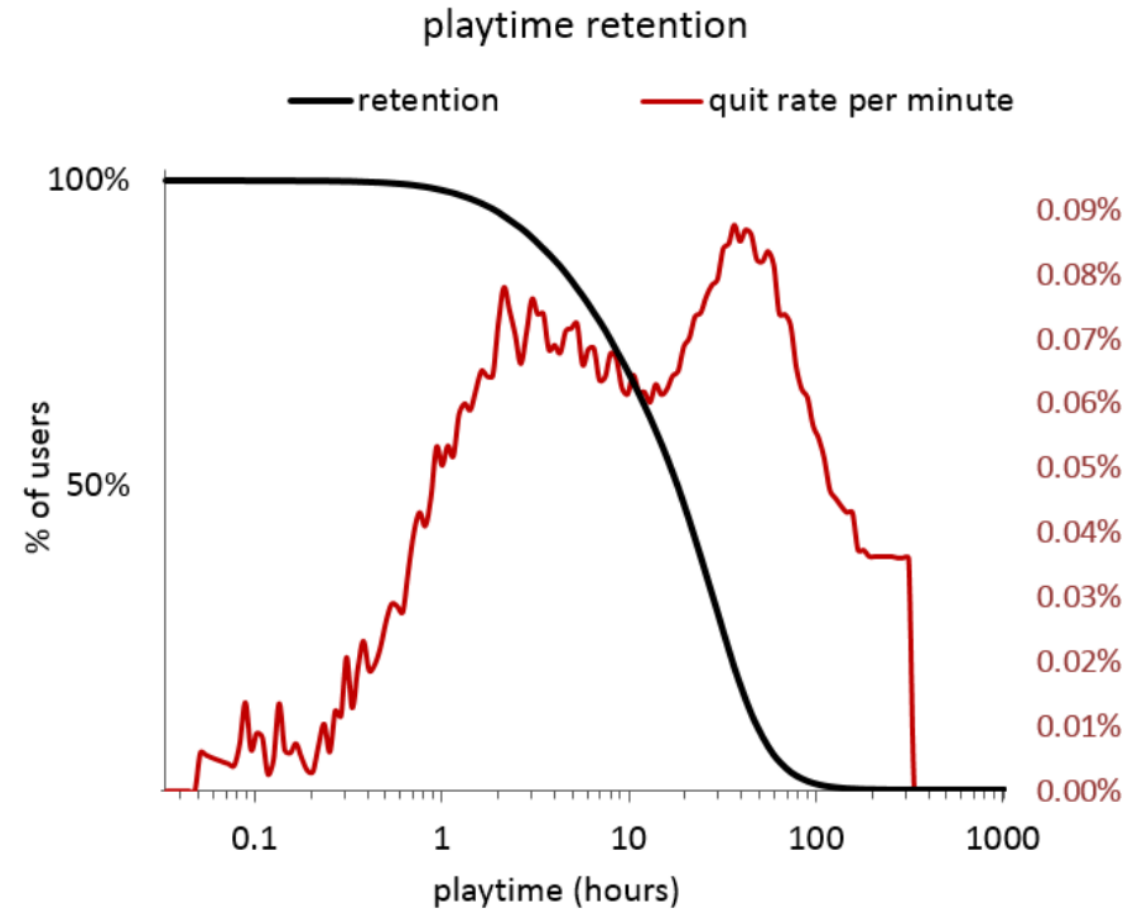


Fig. 4. Far Cry 4 retention in black and quit rate in red.

Churn/retention research

- Viljanen, Markus, et al. "Modelling user retention in mobile games." *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016.
- Allart, Thibault, et al. "Design influence on player retention: A method based on time varying survival analysis." *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016.
- Perriñez, África, et al. "Churn prediction in mobile social games: Towards a complete assessment using survival ensembles." *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016.
- Demediuk, Simon, et al. "Player retention in league of legends: a study using survival analysis." *Proceedings of the Australasian Computer Science Week Multiconference*. 2018.



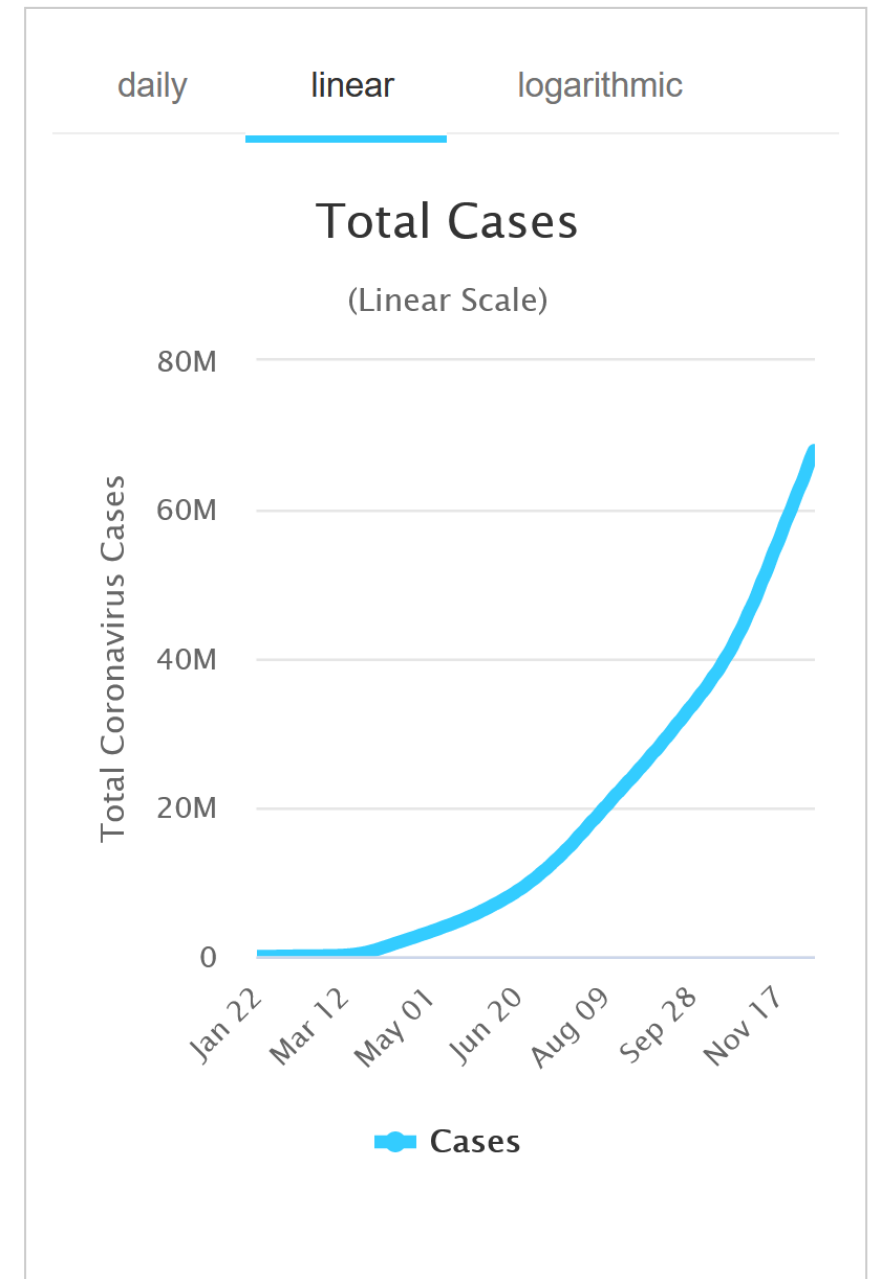
Exponential (viral) growth

- Every coronavirus case infects a number of new cases
- Organic growth of games: Portion of players share on social media or show a game to their friends at school etc.

Equation:

$$u(t+1) = u(t) + vu(t),$$

where v is "virality", i.e., how many new players each player "infects" per day



Source:

<https://www.worldometers.info/coronavirus/?>

Another source of exponential growth in games as a service

- Achieving high virality is hard.
- Combining the following can also produce exponential growth:
 1. Achieve good retention and monetization: At least some players keep playing for a long time and provide a steady revenue stream (e.g., season passes)
 2. Invest a portion of the revenue in acquiring new players



Why does the math matter?

- You can measure parameters like p_{churn} from your player data
- Using the parameters, you can make predictions about your playerbase and company cashflow
- Soft launch: A business practice where you launch the game on a limited market to measure, make predictions, and iterate so that your parameters improve.
- Successful soft launch: Your data and predictions indicate good return on investing in global launch (marketing, servers, support staff...)



Exercise: Simulate retention, virality, and game company economy



RetentionAndVirality.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

Comment

Share



+ Code + Text

Connect ▾

Editing



▼ Basic recursion

First, let's write a very simple simulation of how each player might recruit new players virally with some probability, e.g., by sharing on social media



```
players=1000
viralityProbability=0.1
for i in range(20):
    players=players + viralityProbability*players
    print(players)
```



```
1100.0
1210.0
1331.0
1464.1
1610.51
```

This is the first Python simulation in the RetentionAndVirality notebook. Let's investigate it to learn Python syntax, assuming that you're only familiar with Unity C#...

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

means that the rest of the line is comments that explain the code.
The computer running the code ignores these.

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

This introduces a variable called "players" and sets its value to 1000. Python does not need type specifiers like C#'s "int players=1000"

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

In Python, you don't need to add the ";" at the end of a line.

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

This declares a loop: All code inside the loop is run for 20 times.
In C#, this would be "for (int i=0; i<20; i++)"

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

For loop declaration ends with ":"

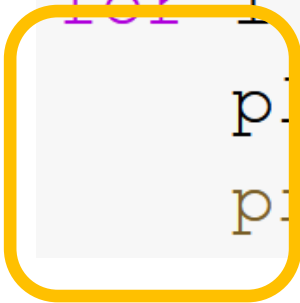
```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```


Everything after the ":" that is intended belongs inside the loop. In C#, the loop contents would be inside curly brackets.

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

A yellow rounded rectangle is drawn around the loop body code, specifically enclosing the lines `players=players + virality*players` and `print(players)`. This visual aid emphasizes that these two lines are the operations that will be repeated within the loop.

This assigns the variable "players" a new value based on its old value and the virality parameter. Mathematical expressions follow standard rules: Multiplication is executed before addition.

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

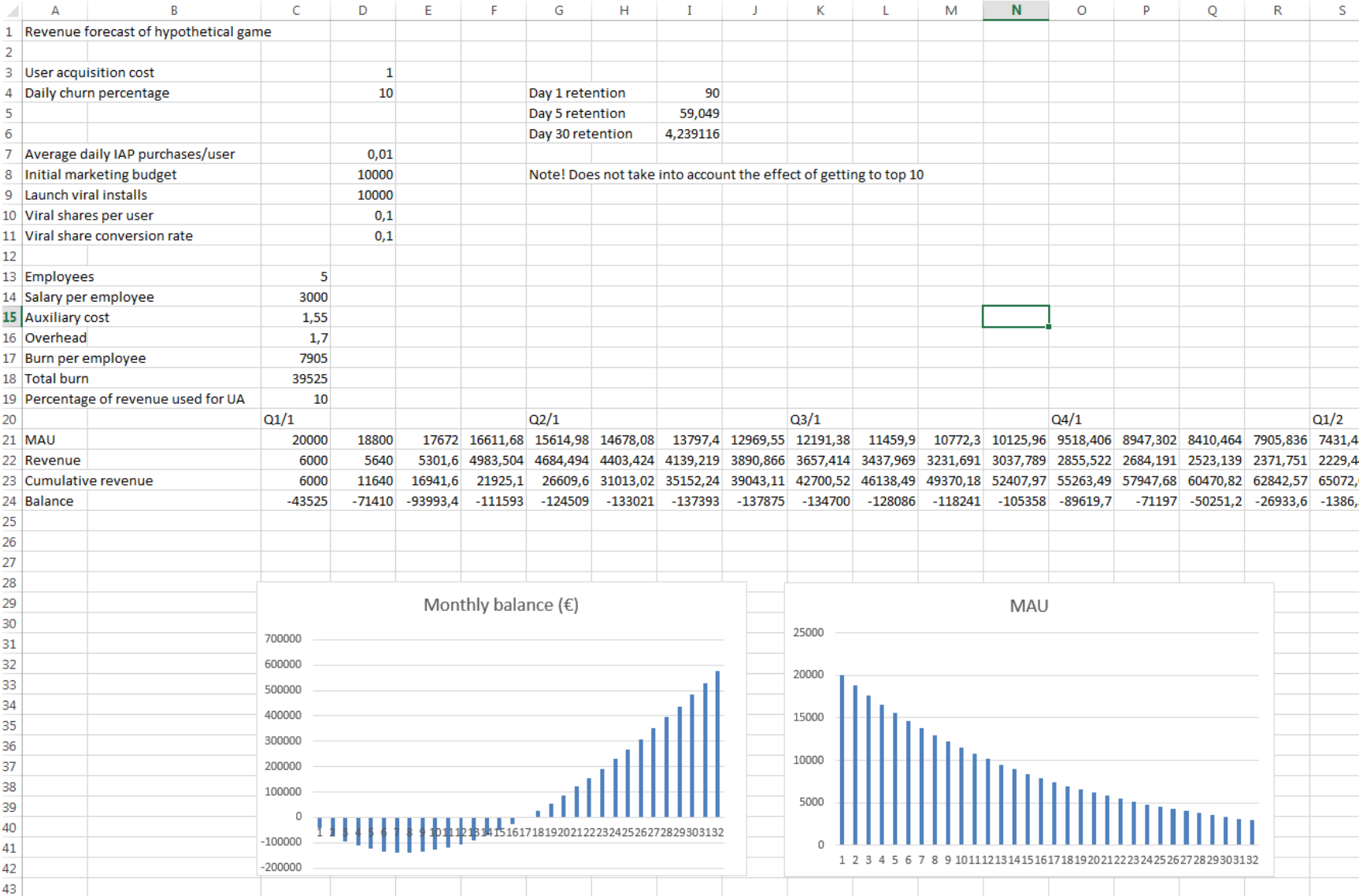
#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

The print function prints things into the Colab output. In Python, print usually works no matter the variable type.

```
#Simulation parameters
#You can choose these as you like.
players=1000
virality=0.1

#Simulation loop
for i in range(20):
    players=players + virality*players
    print(players)
```

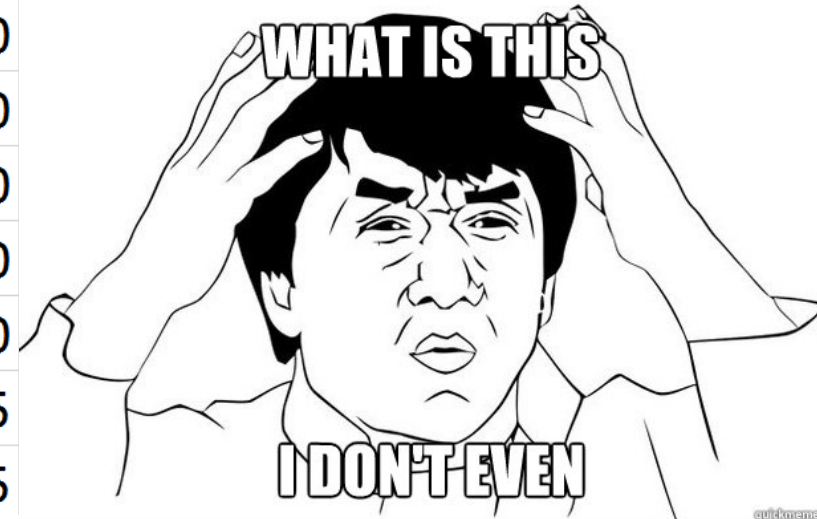
A similar Excel from previous years is also provided in the Github



Why we use Python and Colab now?

- Spreadsheet formulas be very hard to understand for someone who didn't create it. Python code uses human-readable variable names.
- Simple things might be easier in Excel, but Python scales better to complex simulations

		✕ ✓ <i>fx</i>		=I\$6*D22/100+D24/\$D\$3+\$D\$10*\$D\$11*D22	
A	B	C		D	E
MAU		20000			
Revenue		30000			
UA spend		21000			
UA new users		21000			
Cumulative revenue		30000			
Cumulative burn		46575			
Balance		-16575			



Combining spreadsheets and Python/Colab

- Many data scientists do some quick visualization and manipulation of data in Excel
- More detailed analyses in Python or R.
- The Pandas Python package allows one to load, manipulate, and save spreadsheets (we use this in the Clash Royale notebook)
- Python can also directly manipulate Google Sheets using the Google Sheets API:
<https://developers.google.com/sheets/api/quickstart/python>

Break: Work on the Colab notebook

- Everyone goes through the virality and retention Colab in their own pace.
- You can continue tomorrow if you run out of time.
- If you feel confident on your own, you can leave the Zoom.
- I will stay here to help, we can go to a breakout room for one-on-one tutoring where you can share your screen.

Lesson learned: To make it with a small initial budget, one needs to...

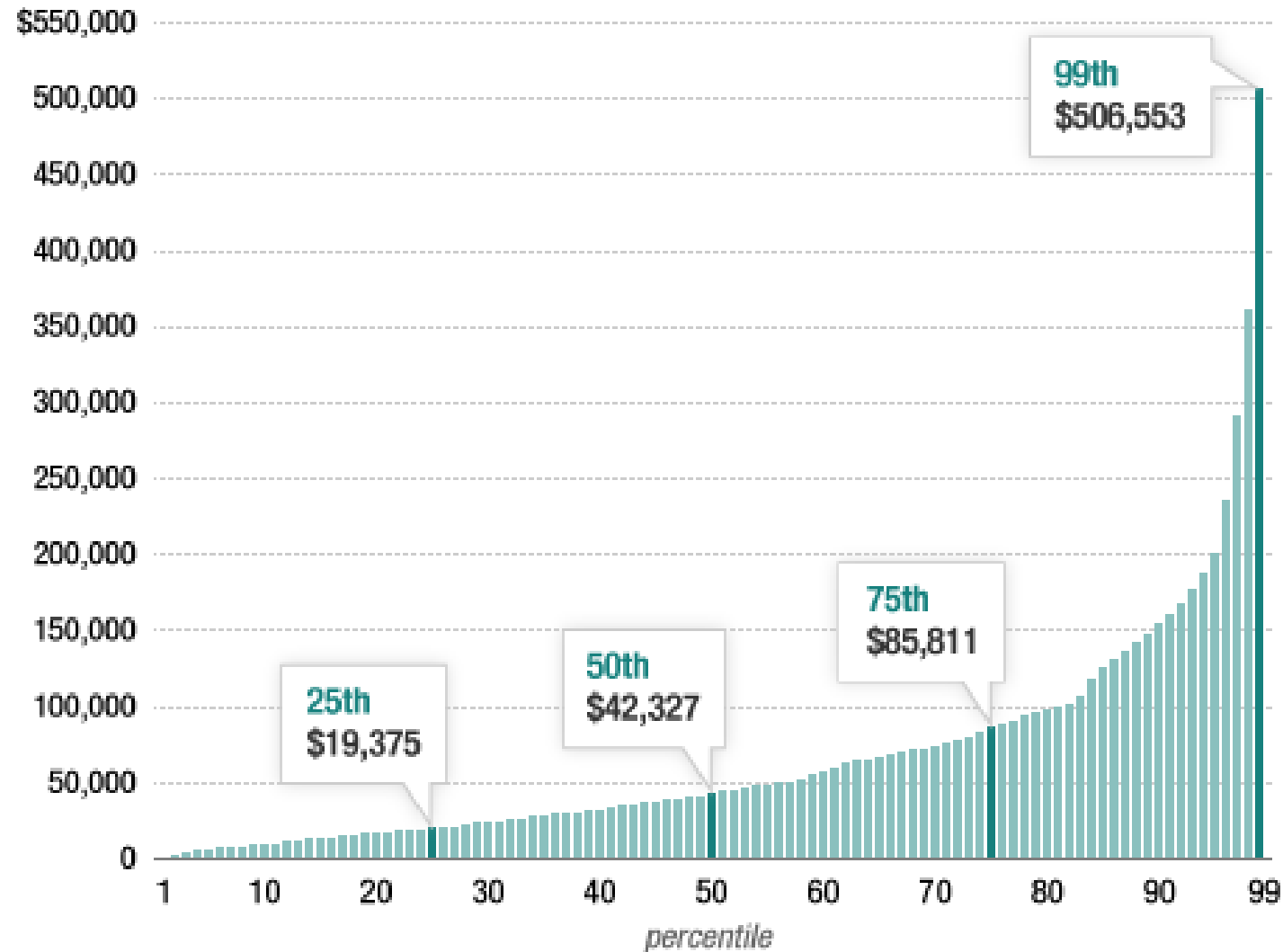
- Minimize burn
- Invest revenue in user acquisition
- Maximize odds of virality and free exposure => create something unique
- For example: Is your game unique and innovative enough to, e.g., get to GDC Experimental Gameplay Workshop or shortlisted for Independent Games Festival? Check the games featured there in previous years.



More on “power law” relations

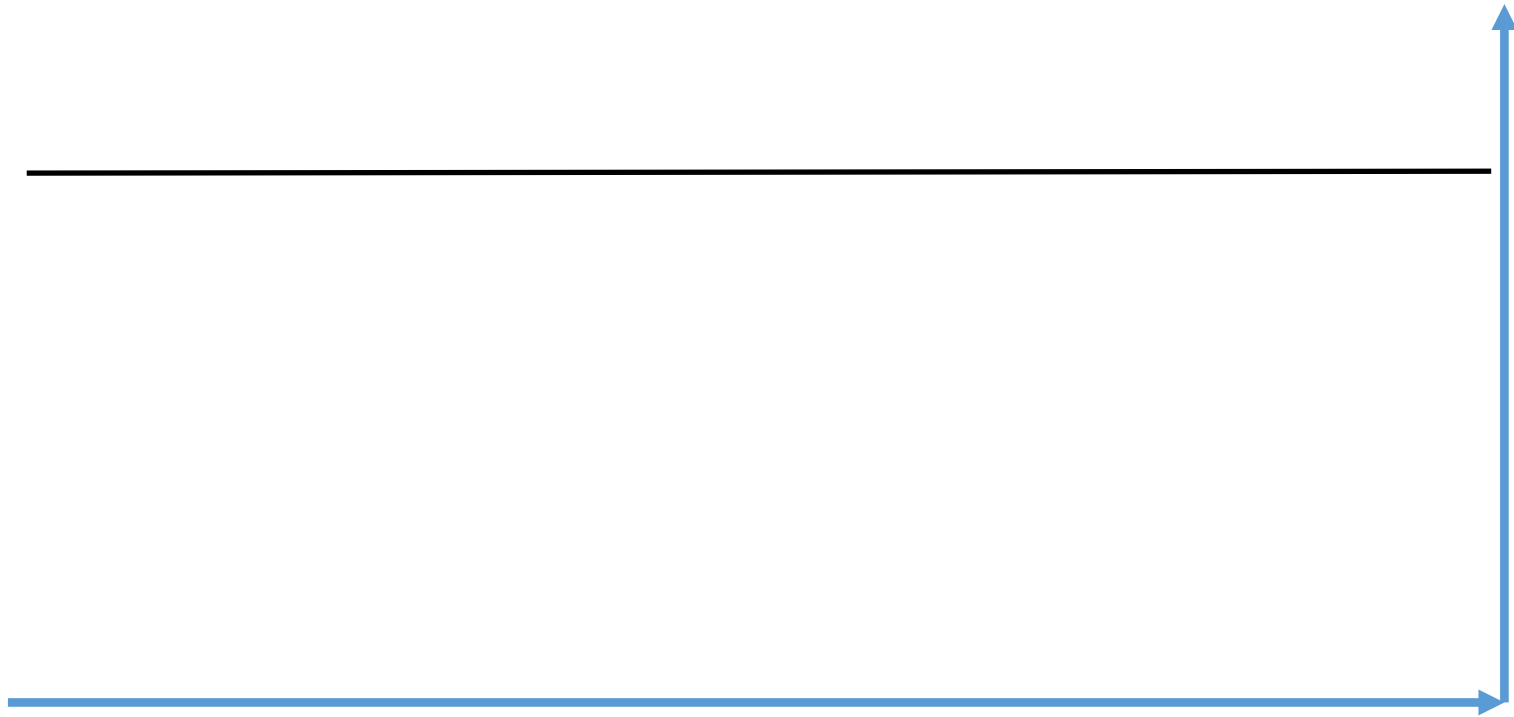


Income distribution looks like a power law. Is there a recursion?

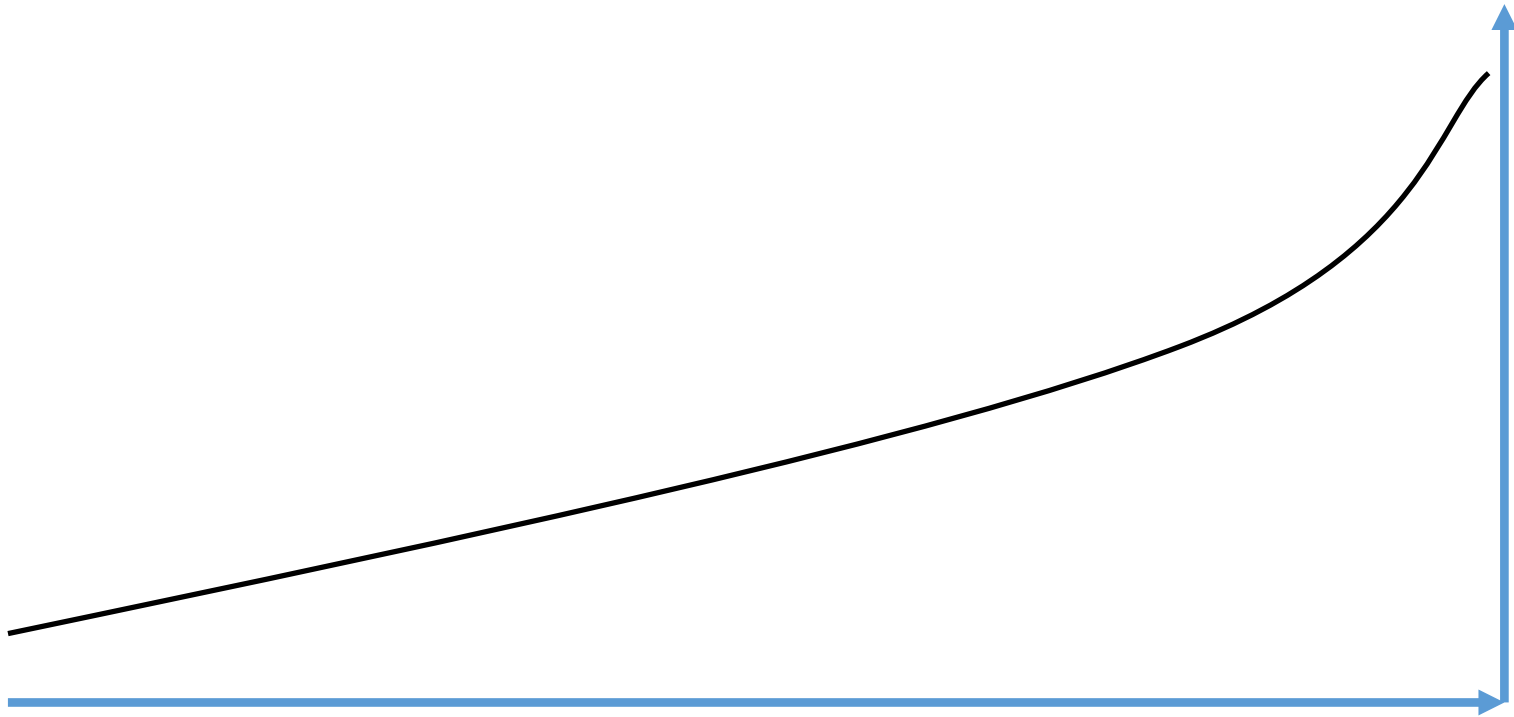




Distribution of wealth through recursion in
Monopoly (game start)

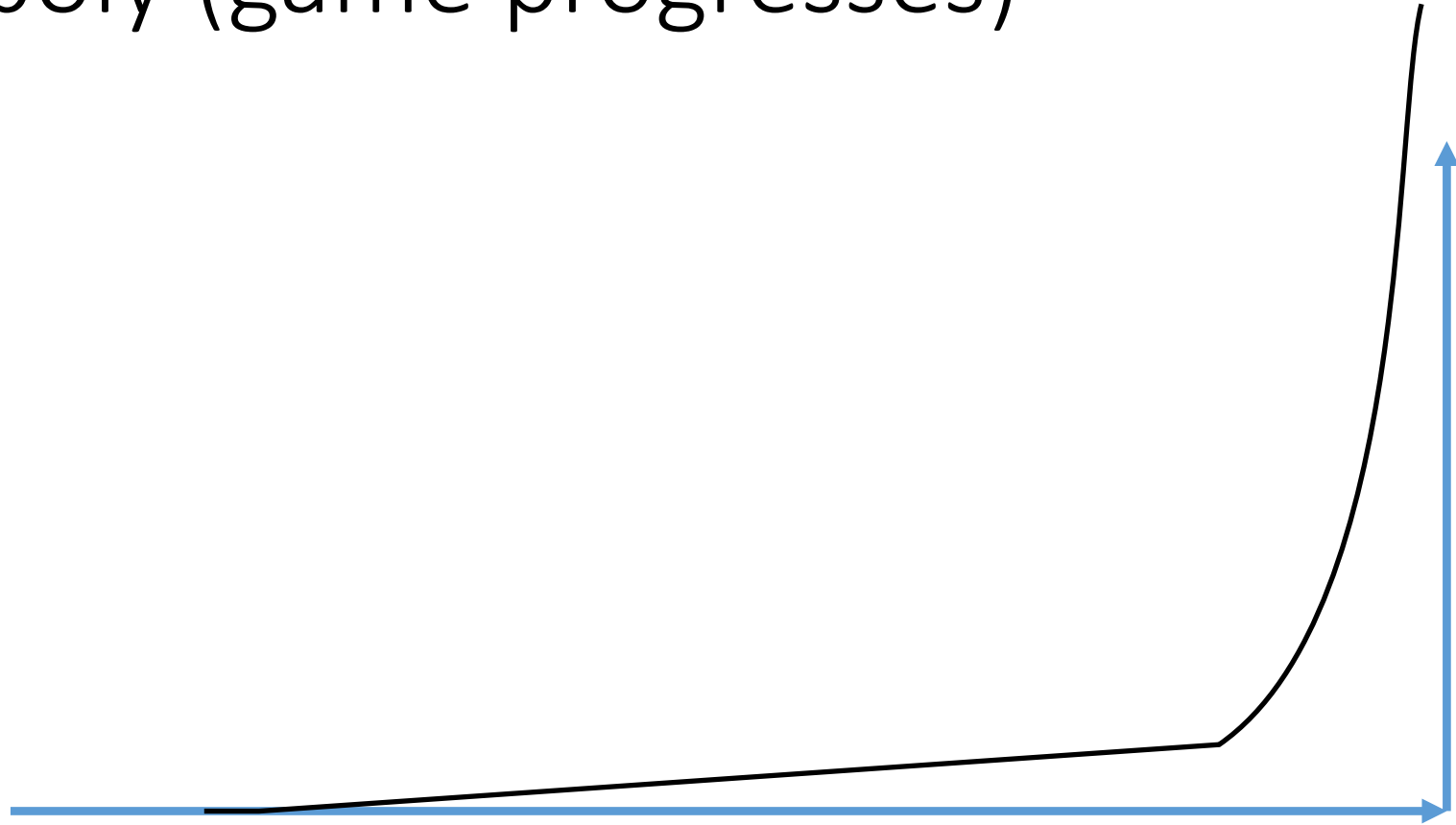


Distribution of wealth through recursion in Monopoly (game progresses)



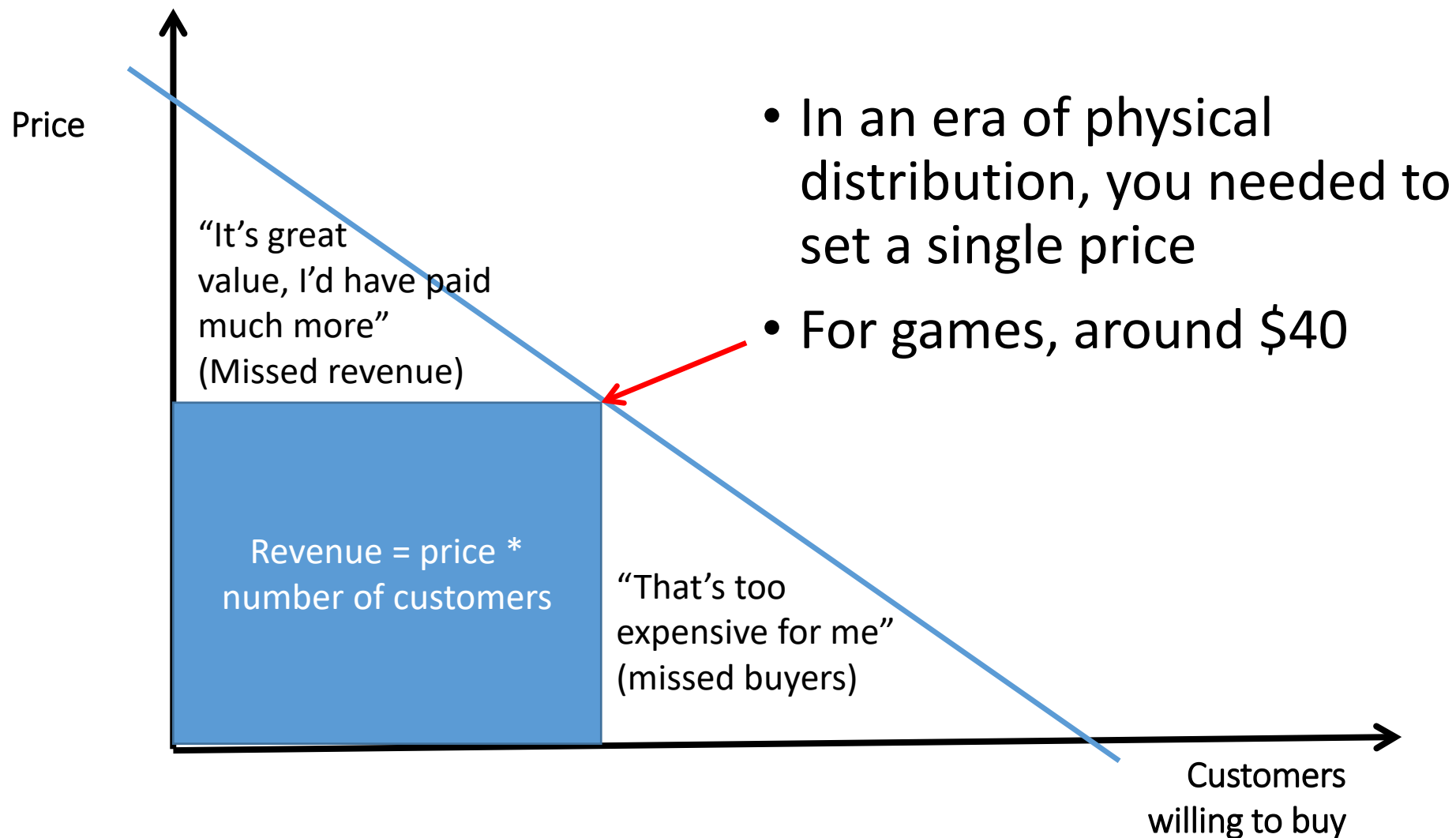


Distribution of wealth through recursion in Monopoly (game progresses)

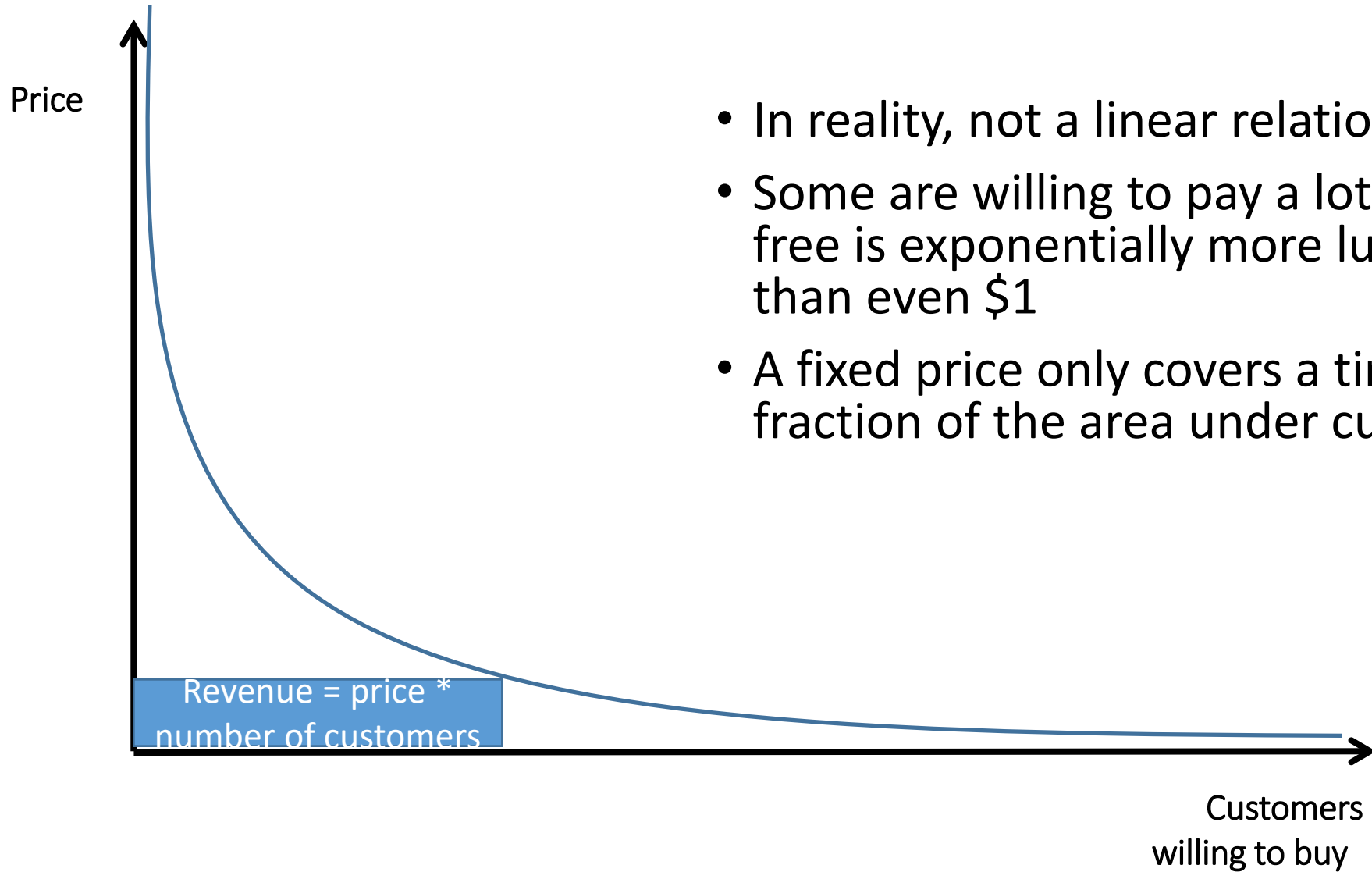


The connection to game monetization

The price/demand curve

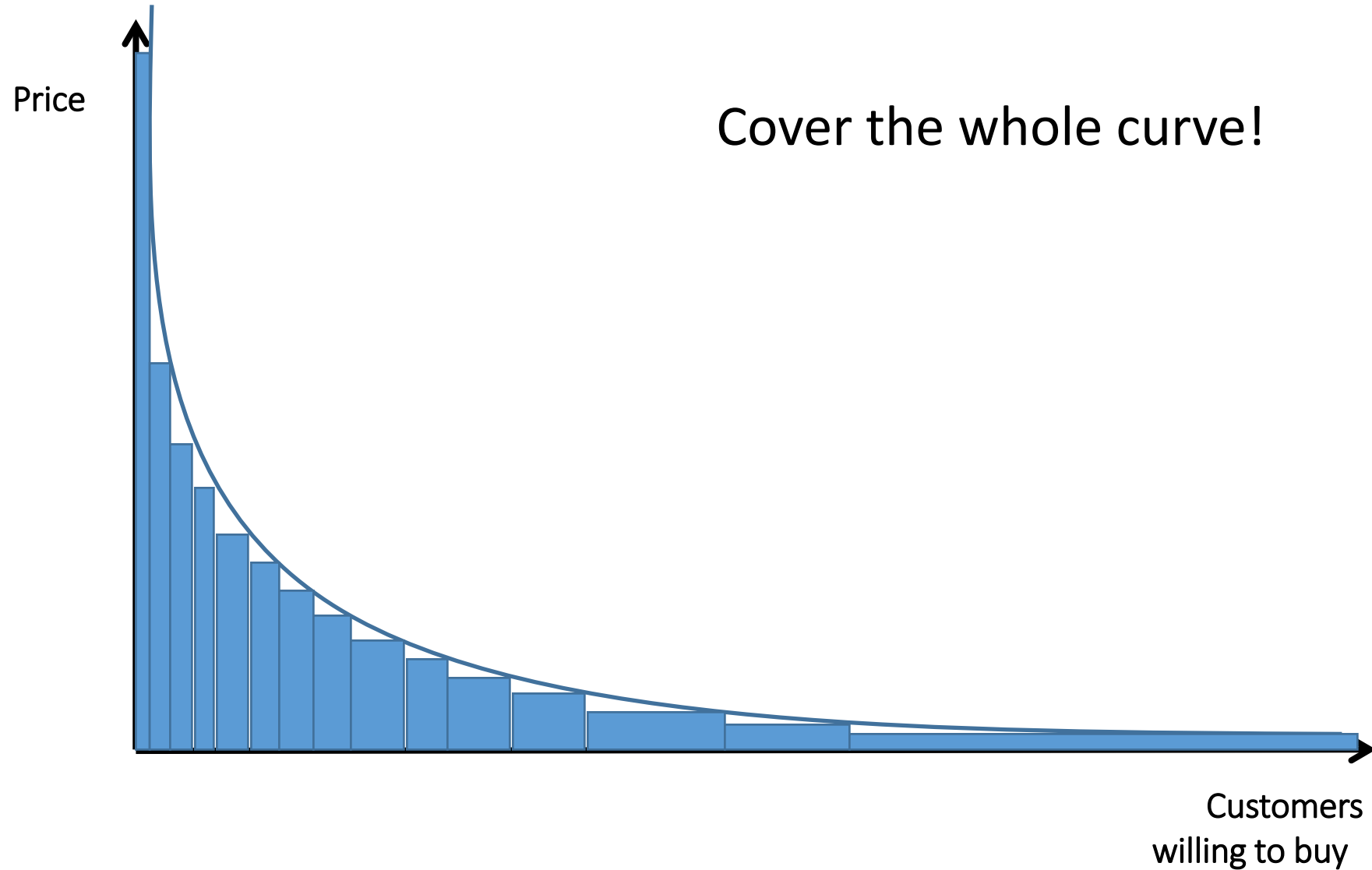


The price/demand curve

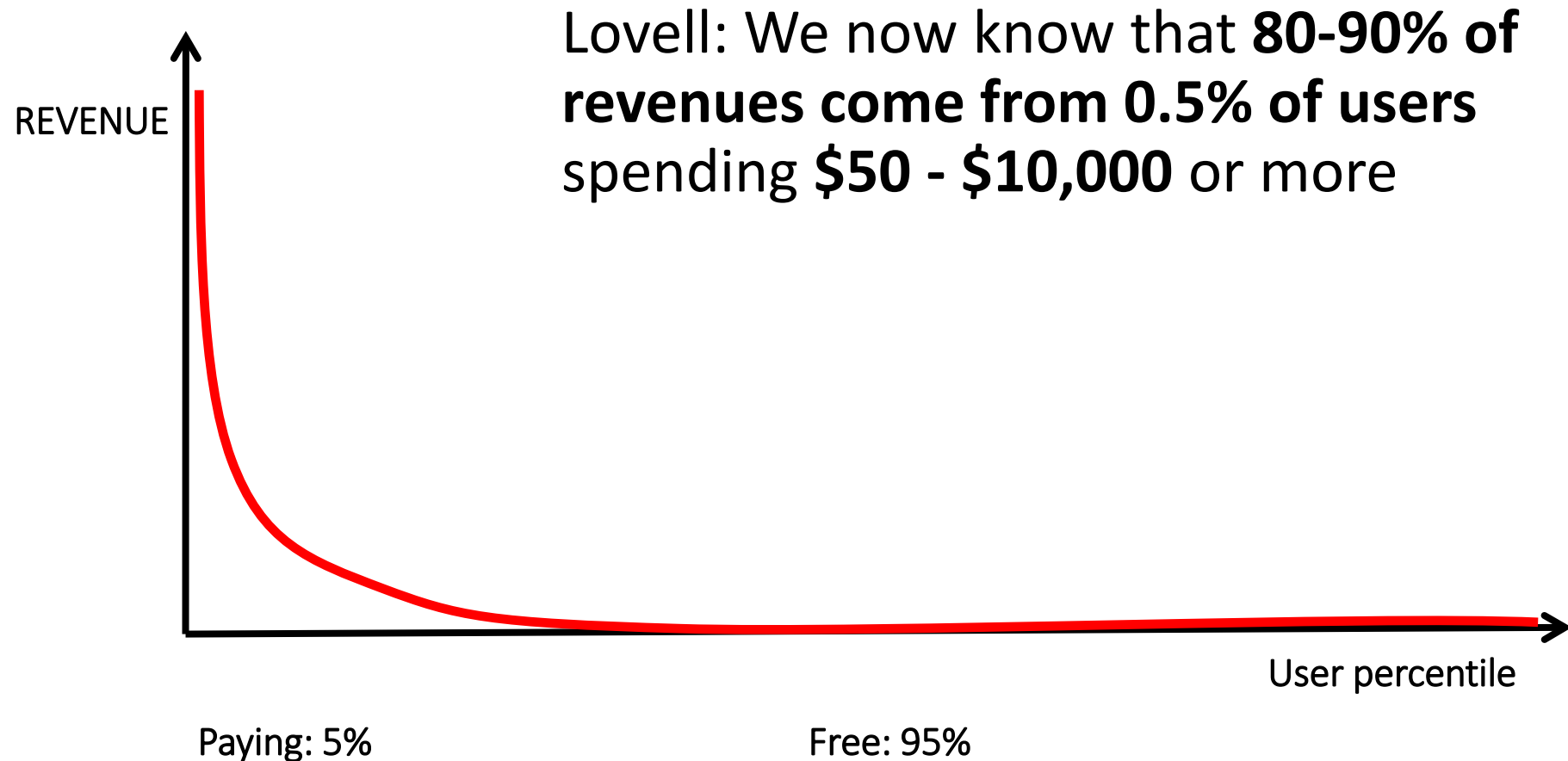


- In reality, not a linear relationship.
- Some are willing to pay a lot, and free is exponentially more lucrative than even \$1
- A fixed price only covers a tiny fraction of the area under curve

F2P: Allowing users to choose how much to pay



F2P: Allowing users to choose how little to pay



Huge differences in players is not news

- 90% of gamers don't finish a game
- Only <10% of purchasers participate in online discussion forums



Other explanation: addiction

- Free-to-play games utilize principles that lead to gambling addiction with some people (more of that tomorrow)
- Combining ethics and economics: Limit player spending, just don't set the limit too low.

Summary

- Power laws and recursion are common in game economics
- Power laws and tipping points: sometimes small changes have huge effects, e.g., change behavior from exponential decay to exponential growth
- Retention, virality (organic growth), and investing revenue for user acquisition are crucial