55 Takeaways from Game School

Nawaf Bahadur

Nawaf Bahadur

55 Takeaways from Game School

Nawaf.Bahadur@outlook.com

Digital copy of the book can be found on the website below.

www.NawafBahadur.com

Printed in Canada

Author's Note

This book is meant to help new students learn the most important topics that every game designer should know. It is also meant to help students coming out of school to refresh the things they have learned. Game design courses cover an overwhelming number of topics and so it is easy to forget useful key concepts.

It all started when I heard in a comment online about Mathew Fredrick's book, *101 Things I learned in Architecture School*. I ordered a copy as soon as I saw how digestible the writing format was. I fell in love with the book so I bought the business and fashion ones. The format was so easy to read and that got me wishing there

was one for game design as well. It would have been very useful when I first started going to school so I turned it into my deep dive project for graduation – and here we are.

It took countless hours of research as well as talking to teachers and classmates. I had archived most of the class notes and presentations over the first few years of the Honor Bachelor of Game Design program at Sheridan College which proved invaluable to fill a bulk of the content. However, there were many holes that could only be filled by talking to the teachers directly to see what kind of takeaways they wanted the students to leave with. All of that got collected into a giant spreadsheet with all of the entries cited and documented. Feel free to contact me if you have any more questions.

Nawaf Bahadur, Level Designer
April, 2018

Acknowledgments

I want to thank Jose Rueda; Michael Brown; Nicolas Hesler; Andrew Carvalho; Max Piesner; Jeff Pidsadny; Cindy Poręba; Michael Assadourian; David Mikula; Raphaël Tétreault; Elise Cole; Laura Onderwater; Seth Byrnes; Daniel Praymayer; Benjamin Scott; Brendan Muir; the Sheridan College faculty; and all classmates that supported me during the creation of this book.

55 Takeaways from Game School

"A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome."

-Katie Salen and Eric Zimmerman
Rules of Play, 2003

Game mechanics are the underlying rules of the game

- Core Mechanic: The most prominent interaction(s) in the gameplay. Core mechanics usually interact with other mechanics.

- Secondary Mechanics: Supporting interactions that help enhance the game or add goals while not being the most important aspect to the game.

The magic circle is an area where the game’s rules have authority over the player.

When inside the magic circle, a new reality that is defined by the game’s rules is created.
When a board game starts, the pieces of paper and cardboard become representations of specific things. When you start a game of monopoly, you become the boat or the shoe, and you're buying lands and trying to build property. Rolling a die suddenly feels exciting.

The MDA framework is a tool used to analyze games. It formalizes the consumption of games by breaking them down into three components.

- Mechanics describes the particular components of the game, at the level of data representation and algorithms.
- Dynamics describes the run-time behavior of the mechanics acting on player inputs and each others' outputs over time.
- Aesthetics describes the desirable emotional responses evoked in the player, when she interacts with the game system

Feedback Loops are situations where part of the output of a system is used as input.

- Positive feedback loops amplifies the output and destabilizes the system. For example, if a player wins, they get more power that then makes them win more.

- Negative feedback loops system the output and adds stability to a system. An example would be giving losing players a reward that would help them win, while the winning players get less or nothing.

Nintendo’s 4 steps for teaching players. AKA kishōtenketsu

1. Teach by introducing to the player the new concept or mechanic in a safe environment.

2. Develop the mechanic more by augmenting its properties or showing the player more enounced aspects of the mechanic.

3. Twist the mechanic and surprise the player with something exciting.

4. Challenge the player at the end with something that will have them feel good about mastering the mechanic.

Game Economy is any kind of system in which resources of any type can be produced, exchanged, and consumed.

Game economies can be closed, as in self contained, or open, as in the economy is extended beyond just the game such as currency purchasable with real money.

There are multiple functions that enables the economy.

- *Source*: Adds new resources to the game out of nothing
- *Drain*: Take resources out of a game, removing them permanently, also called a 'sink'
- *Converter*: Changes resources of one kind into another
- *Trader*: Move a resource from one place to another (and vice versa), according to some exchange rule

Game Juice is what makes the game feel satisfying to play.

Juicing a game is taking a functional game and adding layers of effects of any kind to make players feel connected into the game. Animation and audio are the most common ways to add juice. Screen shake and menu sound effects are examples of that. The juice should compliment and enhance the gameplay experience and not get in the way of it.

Flow is the mental state of which the player is completely absorbed in their activity. Sense of time is lost, and the player is doing it for their own sake.

There are a few things that can be done to help achieve a state of flow.

- Clear Goals. It is clear to the player on what they should do. There is immediate feedback on actions.
- Challenge = Skill. There are things the player can do to stay engaged that requires skill.
- Action and Awareness merge. All unnecessary stimulants get ignored and disappear.
- Loss of self consciousness and feeling as being part of the game.
- Loss of sense of time
- Experience being autotelic – doing it for its own sake

Story is the sequence of events that unfolds within a game.

Traditionally people think that the story is just the cutscene and text added into a game, but it can actually be anything that the designer intended or did not intend. It doesn't have to be a story that the designer added in, it can be a sequence of random events that happen or are imagined by the player while they play the game.

Gameplay is what players do, story is why they do it.

While gameplay or game mechanics can dictate what the player does in the game, story can help players understand why they should be doing those things. Story lends context to gameplay.

Imbedded Narratives are the traditional way of putting story into games. Their defining feature is that they are told to the player by the designers. In a ‘strings-of-pearls’ story method the game goes back and forth between gameplay and story telling.

Emergent Narratives are stories created by the players' use of the game. Emergent narratives arise from the player interactions without primarily being created directly by the designer.

A game can be a story machine. A game experience would be so interesting that a player would want to tell other people that story.

Enable, Don’t Describe

‘Show Don’t Tell’ is good for traditional storytelling, but for interactive storytelling the rule should be, ‘Do, Don’t Show’. Or it should at least be a hierarchy of Do, if not, then Show, if not then Tell.

We should make the players feel like they have the agency to participate and affect the narrative in the game through their actions or decisions. It’s much more effective then having players feel like they are just watching the story.

Serious games are games that are used to do more than just entertain. They are meaningful and serve a purpose like education or tackling social issues.

"Serious games solve problems" - Ben Sawyer

Learn how to code. Being able to test ideas with your own hands is a valuable skill as a designer.

It will make you be able to try things out quickly, helps you better understand how games work, and make you more efficient overall.

“The best teacher is experience and not through someone's distorted point of view”

- Jack Kerouac

Creating small throwaway pieces of software to test out is called prototyping.

Build this quick and dirty to check if the solution or idea are useful or viable to continue working on. The code can always be refactored into something that can be put inside the game if found useful, otherwise throwing it away causes minimal lost effort. Prototyping is good way to search for the fun in the mechanics before making a whole game out of them.

When prototyping, focus on the best aspects of the prototype. If you find an element that feels fun focus on it develop it more.

Don’t get too attached to your original mechanics, if they aren’t fun and you found something that is fun, go with the latter. It will help you make a better game overall.

Pseudocode is an informal high level description of what a piece of software would do or should work.

It usually is written in steps of how an algorithm would act. It is incredibly useful in setting a plan for making a piece of software. Laying out the functions beforehand allows the designer to problem solve before jumping into the implementation.

"A playable sample gives every team member a concrete sense of what the game could be. It gives everyone an experience to give feedback on. It is also both fun and motivating to have a working prototype."

-Nathan Lovato

Game engine is the piece of software where various game parts mix together to become a working game.

Some make their own and many use off the shelf software. The latter has the benefit of saving on development cost and provide an easier way to make games if you're not a programmer.

Spend a small amount of time building a tool that will save you a huge amount of time making the game.

It might feel like it is better and faster to just jump into the game making, but it will save you a lot of time in the long run to make a tool or software that make you work more efficiently over the course of development.

Code refactoring is restructuring computer code without changing its behavior.

- Refactoring can be done to improve readability, improve efficiency, reduce the complexity of the code, etc.
- This makes code more useable and easier to work with.

Version Control is a system that records changes to a file or set of files over time so that you can recall specific versions later. It is a common way for teams to share files and put them in the game. The files are stored in a repository that users ‘push’ to and ‘pull’ files from it.

When pushing to the repository make sure to follow the standard format for commenting.

Here is a standard format:

- Separate subject from body with a blank line
- Limit the subject line to 50 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line
- Wrap the body at 72 characters
- Use the body to explain what and why vs. how

Level design has to be functional, usable, and delightful.

- Functional
  - The design must work
  - Fulfills a need or an objective for the game
- Usable
  - The design must fit the game metrics
  - Intuitive vs Interpretive
- Delight
  - The design challenges and rewards the player
  - Entices the player to continue

Utility Belt vs Tool Belt

Tool Belt:

- Fixes routine problems with known solutions
- Has one tool for each job
- Collection of specific solutions

Utility belt:

- Analyzes problems and looks for solutions
- Flexible, can be used with many types of problems
- Can lead to several solutions to the same problems

You want a utility belt not a tool belt.

“I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail.”

-Abraham Maslow

Metrics are numbers and variables that inform the level design and space. They usually refer to things like sizes of objects and spaces in a game.

Metrics are:

- Building blocks for composition
- Generated by the analysis of the design requirements
- Useful to make things modular and repeatable

A Beat Map is moment by moment description of the gameplay and story events that the player will encounter as they progress through the level.

It also describes the important moods, character development, story, and dialogue within the game. Beat maps usually cross references with diagrams, maps and other documents related to progression.

Challenge is any obstacle that impedes the actions of the player. It introduces difficulty to the game and makes it more fun. It requires skill to overcome.

Careful not to mistake frustration for challenge. The former usually comes from the player feeling cheated while the latter comes from the player believing that their skill is the main factor for winning.

Risk is the exposure to the chance of failure.

Risk makes the player want to overcome obstacles and keeps them engaged. Risk should generally increase as the player progresses and masters the game, but it is not necessarily linear. It has to balanced with an appropriate award.

Uncertainty describes the condition of the player not knowing what to expect.

Pure uncertainty can lead to frustration and should be avoided. If the player doesn't have any indicators on how their actions will lead to consequences or rewards they become frustrated. With no information the decisions become random guesses that feel unfair. On the other hand, having some uncertainty can make the game more exciting and keep the players on their toes.

"You are never directly designing the behavior of your players.
Instead you are only designing the rules of the system."

-Salen and Zimmerman
Rules of Play, 2003

A Greybox is the initial 3D representation of a design that is playable in the game engine.

This is commonly used to provide a playable rough outline of a level before it gets sent to artist to fill up with assets. It focuses on the massing and volumes of the levels, less so than things like lighting and textures.

Modularity is design approach that subdivides a system into smaller parts, called modules, that can be fitted together to make something bigger or to be reused in other systems.

Modularity offers a reduction in cost, flexibility in design, and editability for future iterations. Modular design tries to combine the advantages of standardization with those of customization. This applies to many fields including level design and programing.

User Interface, or UI, is used to describe the way the player and game communicate with each other.

UI has to main components, an input and an output.

- Input is how players communicate their actions to the game.
- Output is how the game displays the results of these actions.

UI comes in multiple categories. The same UI element can overlap multiple of these categories.

- Diegetic: Interface that is included in the game world. It can be seen and heard by the characters.
- Non-diegetic: Interface that isn't part of the game's world. It can't be seen or heard by in-game characters.
- Spatial: UI elements that are presented in the game's 3D space.
- Meta: Representations that can exist in the game world but aren't presented spatially to the player. An example of this are effects that are rendered on the screen like blood splats when damaged.

Keep in mind the 6 Principles of Graphical User Interface, GUI

- The structure principle: design should organize UI purposefully, in meaningful and useful ways based on clear, consistent models that are apparent to the users. Put similar things together and divide different things.
- The simplicity principle: The design should make simple, common tasks easy, communicate clearly and simply in the users language.
- The visibility principle: the design should make options and materials visible and without distracting using unnecessary information.
- The feedback principle: the design should keep users informed of actions, or changes of states or conditions, and errors or warnings that are relevant.
- The tolerance principle: the design should be flexible, reduces the cost of mistakes by allowing undoing and redoing, while preventing errors by tolerating varied inputs.
- The reuse principle: the design should reuse internal and external components and behaviors while maintaining consistency with purpose.

The spiral model is a risk-driven process model used in software production. It guides a team to adapt elements from multiple production methodologies like waterfall and agile.

Playtesting is testing a game to find design flaws or software bugs.

When making a playtest make sure to:

- To have your target player testing the game
- Make sure the software is working and that you have specific questions ready for the testers.
- Make sure the playtesters are comfortable. Brief them on the rules and situation beforehand, and debrief them afterwards.
- It’s preferred not to talk to the testers during play as to not effect the data. Ask them to think aloud instead.
- Analyze your data and make a plan on how to use it.

“I prepare each of my playtest sessions in great detail - I plan exact issues I want to monitor and test. During play, I record relevant data about game flow. Afterwards, I analyze the results and then make necessary or exploratory changes. This becomes the preparation for the next playtest session, during which I can find out how changes will affect the game.”

-Richard Garfield

"When we playtest and iterate a game, we make changes that attempt to adapt the game's form and possible spaces that emerge from it to the psychology and behavior of players. We move the game away from purely being about our own expression to adapt it for an audience."

-Anna Anthropy and Naomi Clark
Game Design Vocabulary (2014)

"If you set the difficulty to easy so you can see all the content, you're doing it wrong”

-Frank Lantz

Waterfall development follows making software through a series of stages. Each phase leads to a more expensive one afterwards.

- The first phase consists of making the plan on how to build the software.
- The software is created in the second phase
- The final phase is putting all the pieces together and testing them
- On a waterfall project, most of the design work are done in the early stages, while the majority of the testing is done in the later stages.
- Testing late can lead to a costly redesign or the product being shelved.

Agile Manifesto is a formal proclamation of values and principles for iterative and people focused software development.

Agile Manifesto Values

- Individuals and Interactions over processes and tools
- Working Software over comprehensive documentation
- Customer Collaboration over contract negotiation
- Responding to Change over following a plan

Agile Manifesto Principles

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Close, daily cooperation between business people and developers
- Face-to-face conversation is the best form of communication (co-location)
- Projects are built around motivated individuals, who should be trusted
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Self-organizing teams
- Regular adaptation to changing circumstances

Volunteering and going to game jams are some of the best ways to put yourself out there and getting to know your peers in the game industry.

The game industry can be like a small village, everyone knows everyone. Making yourself visible in events is a good way to build rapport with industry professionals when trying to break in.

Crunch time is still prevalent in the industry. Burnout is a common and causes a lot of people to get sick and/or leave the industry.

Check companies reputation for crunch time when applying. Make sure to give yourself breaks and maintain a healthy life balance. Remember that you can’t work if you're dead.

Game schools can only teach so much

You can’t expect a game school to teach you everything. This book is a starting point, but it is not enough. You need to have the will to learn on your own.

Nawaf Bahadur is a level designer and graduate of Sheridan College's Bachelor of Game Design program. Originally from Saudi Arabia, he came to Canada in 2013 looking to acquire the knowledge and skills to jump-start the middle eastern video game industry. He worked with Cococucumber as a junior level designer on Riverbond (PS4, Xbox One, Steam; 2018), winner of Best In Play at GDC 2017. He also showcased a game in the Toronto Comic Arts Festival and at Hand Eye Society's Game Curious program.

Game Design Studies

55 Takeaways from Game School Nawaf Bahadur

After 4 years of game design in college and out, I realized how easy it is to forget the hundreds of concepts and lessons we sped through over the course of each semester. This book is meant to help refresh core concepts for past and present students alike. Selected are the 55 most important topics that every game designer should know.

"55 Takeaways from Game School's ability to concisely collect the most important technical and creative aspects of game development is a testament to Nawaf's ability as a designer. The summary of knowledge gained during his formal education shows an understanding of games that is well beyond expectation for a graduate. It's clear that his work experience and vast amount of time volunteering and helping grow the local Toronto game community has imparted on him the ability to recognize the skills and effort required to succeed in the industry."
Andrew Carvalho, Co-founder of Laundry Bear Games, programmer for *Mortician's Tale*

"Nawaf provides an indispensable and definitive resource for young designers or anyone who wants to up their game."
Michael Brown, Game Designer, VR Designer for the Emmy Award winning *Oculus Sleepy Hollow Project*

OAKVILLE, ON HTTPS://WWW.NAWAFB.COM/