UNIVERSITY OF CALIFORNIA

SANTA CRUZ

GAME RANDOMIZATION: EMERGENT GAMEPLAY COMMUNITIES

A thesis submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

In

COMPUTATIONAL MEDIA

By

Celeste Clark Jewett

September 2021

The Thesis of Celeste Clark Jewett is Approved

_____

Professor Sri Kurniawan, Chair

_____

Professor Nathan Altice

_____

Professor Michael John

_____

Peter Biehl
Vice Provost and Dean of Graduate Studies

TABLE OF CONTENTS

LIST OF FIGURES

22. *Super Mario RPG Randomizer Tracker*
23. *Final Fantasy IV Free Enterprise* twitch livestream race
24. *A Link to the Past and Super Metroid Combo Randomizer* dev official apology on Discord
25. Zfg list of most popular Youtube videos by view count

ABSTRACT

GAME RANDOMIZATION: EMERGENT GAMEPLAY AND COMMUNITIES

By Celeste Clark Jewett

In the second half of the 2010s there is a new trend emerging amongst the gaming community. The video game randomizer is a project taken on by hobbyist developers who are fans of a particular game and wish to extend its lifespan; to make the old new again. Some of these randomizers have become so popular that entire communities are beginning to form around them. In this paper I put forth a taxonomy for video game randomizers that categorizes them by degree of change to the base gameplay brought about by randomization. I identify a tipping point where the transformation to gameplay becomes fundamentally different from the base game and wherein the community forms. Through interviews and research I construct an ethnography of these communities to identify what other communities randomizer developers and players belong to and compare and contrast those communities.

Acknowledgements

I would first like to thank Professor Nathan Altice. As my advisor, I found your writing feedback invaluable, and he motivated me to do thorough and thoughtful research. Without his guidance this simply would not have been possible.

I would like to thank Professor Michael John for his willingness to offer critical feedback on short notice. Your comments helped this thesis solidify into a more coherent whole and for that I am deeply grateful. I'd also like to thank professor Steve Whittacre for helping to guide me through the initial parts of my research and offer me advice on how to go about conducting ethnography.

I would like to thank my colleagues. Jared Pettitt listened to my long diatribes about randomization and was the first to suggest that it could be the subject of my thesis. I'd like to acknowledge Tamara Duplantis for offering feedback for my work and putting me in contact with several community members of the *Final Fantasy* randomization scene. I would like to thank Cassandra Ravenbrook for offering to keep me honest on my personal deadlines with this project.

Finally, I would like to thank my friends and family. Thank you to my girlfriend Haley Phoenix for helping me stay sane and giving me feedback when I needed it. Thank you to my girlfriend Lauren Terwiliger for letting me set up my unreasonably large pc in her home and supporting me while I gritted my teeth and did the hard work to complete this. Thank you to Chelsea and Chris for being there for me when I needed to gripe about the

lack of sleep and proper nutrition associated with this herculean task. And thank you to my mother, father, and brother for putting up with me going on about one of the most esoteric subjects possible.

GAME RANDOMIZATION: EMERGENT GAMEPLAY COMMUNITIES

INTRODUCTION

One night in December of 2019 I found myself live streaming on Twitch to an audience of about a dozen people. I had just recently started streaming a Zelda game every Monday evening. Tonight I found myself in the bowels of *Ocarina of Time*'s leviathan Lord Jabu Jabu. This leg of the adventure features our hero Link escorting the fish-like princess of the Zora People out of the sea creature which devoured her. Tonight's stream had a catch though. The locations of all the items in the game had been randomly shuffled, so while I had possessed the boomerang required to solve many of the dungeons puzzles I lacked the iconic sword Link uses to vanquish his foes. There were reasonable work-arounds as I proceeded, avoiding enemies or using unconventional weapons like pots and sticks to vanquish any foes that might oppose my righteous quest. My momentum came to a screeching halt when a big Octorok stood between me and the next room. It was seemingly impervious to any of my tools. The boomerang, the sticks, and the slingshot had no effect on this vicious creature. Desperate to avoid perishing in this gooey dungeon, I grabbed the only thing left I had not thrown at the beast, which in this case was the Princess I was escorting out in the first place.To my surprise, and the delight of the Twitch audience, the Royalty's scaly body found purchase against the monster, and was the key to victory.

This moment of emergent physical comedy was made possible by a program known as a randomizer, a program that changes predetermined aspects of a game according to a randomly-generated seed. In *The Legend of Zelda Ocarina of Time Randomizer (ootrandomizer.com)* the randomizer primarily changes item locations, which dramatically alters the player's progression through the game. Other games' randomizers may relocate key items to change progression, edit character statistics to require players to develop new strategies, alter the physics of certain areas to change the way that a player engages with an object, or alter less mechanically impactful things, such as changing cosmetic elements or where songs might be played during the game. Choosing what should be randomized in a game is an important design question for the randomizer developers. Which elements of the base game are selected for randomization by the developer will determine how different the randomizer is from the game it randomizes. In extreme cases, such as the *Lufia 2* and *Earthbound* randomizers, combining existing game elements with map randomization and character progression have produced radically different gameplay experiences.

Randomizers exist in a supercategory I am calling "User Iterative Content". This category includes ROMhacks, mods, and tools that change an existing computer game. The creators of User Iterative Content do not profit directly from their creations and it is an iteration on software that has been created by someone else.

Randomizers fit within this supercategory because they exist to randomly modify games but may or may not be mods or external programs.

I assert when a randomizer approaches a sufficient degree of transformation to be considered a distinct gameplay experience it begins to take on a life of its own outside of the game's existing community. Instead, a distinct randomization community begins to form around this transformed product. These new communities are often made up of the developers of the randomizer, ROMhackers who have significant knowledge of the base game, speedrunners and content creators who use the base game as a source for their online presence, and enthusiasts of the base game seeking to experience one of their favorite games in a new way. These players will drive feedback for the randomizer and will suggest changes that are often incorporated into subsequent builds of the randomizer in an ongoing, recursive development cycle. These communities, if not carefully curated, may fall victim to the unfortunate trappings of many communities surrounding games, including sexism, homophobia, racism, transphobia, and a general culture of gatekeeping.

The developers choose which game to randomize based on a number of factors, like what aspects of the game can be shuffled or altered randomly, and the popularity of the base game. Considerations include character statistics, enemy abilities, item locations, but can be as simple as the color palette of the player character’s outfit. A developer may also have a personal attachment to a game, for

example if they've played it for a long time since they were young. When choosing to randomize *Super Mario World*, developer Author Blues had already been deeply involved in the game's ROMhacking and speedrunning community. While designing the *Super Mario World Randomizer* he drew upon his knowledge of ROMhacking but wanted it to feel "fundamentally different" from a traditional romhack *(Authorblues)*.

> "There's a whole 3rd version of the randomizer that was supposed to come out that
> attempted to create new levels on the fly. It took snippets of levels to create new levels
> on the fly."
>
> -Authorblues, Super Mario Randomizer Developer

During the development of the game he attempted to create a randomizer that would be able to generate new levels out of pieces of existing levels. As of April of 2021, his Randomizer approaches this by combining levels and "sub levels" (areas in the game that are separated by a loading zone) randomly with one another.

Authorblues' beginnings amongst romhackers and speedrunners is not an uncommon origin for randomizer developers. Much of the technical knowledge to develop one of these programs is similar to that which is required for ROMhacking. The speed at which a development team can create a randomizer is increased by

building upon the base game's ROMhacking or modding foundation. For players, extensive knowledge of the base game will be valuable to successfully finishing an instance of a randomizer. This causes speedrunners to gravitate towards this type of experience. Highly virtuosic play of the randomizer will often give rise to a randomizer racing community. Since the first user-created randomizer in 2012, the creation of a racing scene has become more and more ubiquitous, so much so that racing scenes will often develop in tandem with a randomizer's development.

ROMhacking

Since many randomizers utilize knowledge from ROMhacking, it's important to know what ROMhacking is and what information is pertinent to randomizers. The term "ROMhacking" originates from the words "ROM" (Read Only Memory) and "hacking," as romhackers need to hack the ROM data of a game to alter its contents *(Sanchez, 170)* ROMhacking is a type of data substitution: one locates data in memory and replaces old values with new ones. The popular site Romhacking.net describes ROMhacking as "Modifying the data in a ROM image to achieve such purposes as playing the game in a different language than intended, creating new levels for old games, or maybe playing with a different skill level than intended." ROMhacking falls within a supercategory I refer to as "User Iterative Content". User Iterative Content indicates that a game has already been released by the developer but is being changed in some way by a player usually in a way that continually builds on itself. In addition to ROMhacking I consider mods and Randomizers to be

User Iterative Content as well. Since ROMhacking involves taking a finished product and substituting data for some desired result, it is content that is being changed to better suit the user's desire.

When a randomizer is meant to randomize a ROM file it is almost always an external program. These programs accept ROM files to randomize and output the randomized game as another ROM file. These files are usually able to run on their native hardware, but are most accessible by running on an emulated version of the game's console. Emulators in this case are virtual environments that can run these games on contemporary platforms, usually a personal computer *(Pinchbeck, Anderson, et al, 3).*

A randomizer that accepts a ROM image to generate a new randomized version of a game must know the location of the data it wishes to randomize. It then substitutes new data in accordance with the parameters selected by the user. Therefore, extensive ROMhacking of a game provides a useful knowledge base for a would-be randomizer developer. A good example of this is the Shin Megami Tensei Nocturne Randomizer. The development team wanted to randomize the dialog options during the "Demonic Bargaining" sequences of the game. However, no documentation of this system exists. The team then had to find the memory addresses for all the negotiation text. This task took the team weeks to complete, whereas writing the algorithm to randomize this text data would've taken a team

with adequate documentation only a few days*(PinkPajamas)*. This relationship between Randomization and ROMhacking can go both ways. Before working on the *Wind Waker Randomizer* LagoLunatic worked on randomizers for *Castlevania: Dawn of Sorrow, Castlevania, Portrait of Ruin,* and *Castlevania, Order of Ecclesia (*All of which have been aggregated into a single randomizer "*DSVania Randomizer*.") *(LagoLunatic, github.com)* In his words "Those games had barely been hacked before so I basically reverse engineered them myself to make it."*(LagoLunatic)* The connection between ROMhacking knowledge and randomization tends to draw ROMhackers to the randomization community, and many randomizer developers have their backgrounds in ROMhacking.

Each randomizer looks at different parts of the ROM, but the most important information to a randomizer developer is where things exist in memory. In-game data is referenced by "pointers" that tell the game where something is in memory. For example, the treasure chest containing Link's sword in *Ocarina of Time* is associated with a particular memory address that points to the actual data that is Link's sword. When the game wants to retrieve the sword within the chest it is given a hexadecimal pointer to the sword's value in memory. These pointers are often stored within long lists of pointers called "pointer tables." A randomizer then substitutes the values of these pointers in some meaningful way. For example, the chest that would usually reference Link's sword may now reference some other item

in memory, like a bomb or some nuts. In order to create a randomizer for *Ocarina of Time* that is able to meaningfully substitute this data a developer must have access to the addresses of all places where items can be found and to the addresses of all items.

Shuffling around pointers to indicate where things are in memory is one form of data substitution randomizers may employ but a different approach is required if you are randomizing values within a game. Randomizer developers will still need to know where things are in memory in order to locate the values they would like to have modified. It is here where pointer tables are significant. These tables contain many pointers that have the same type of information. This can be invaluable when trying to locate a specific type of data. For example, in *Pokemon* there is a table of pointers that references every move in the game. This table makes it easy for ROMhackers to quickly find and change data for those moves. This information also eases the burden on randomizer developers by having all the data they might want their program to change in one place.

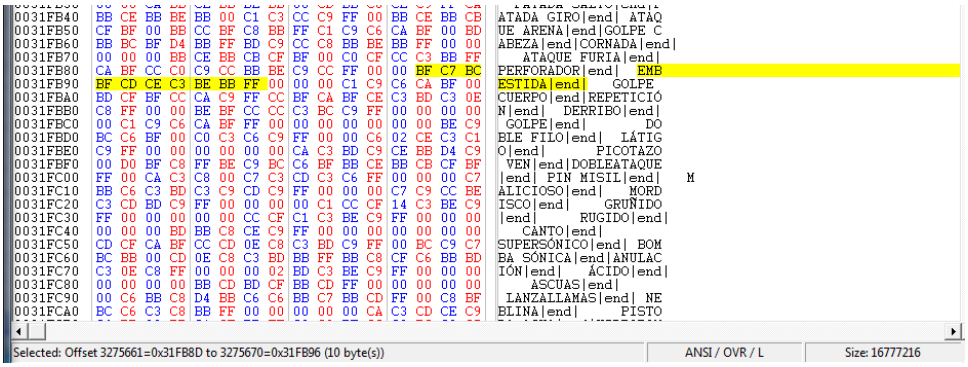


Fig 1: A ROMhacking utility for Pokemon Fire Red[10] lists the pointers for all the moves on the far left. It displays the data associated with that move in hexadecimal. On the right the tool displays the move's name (in Latin American Spanish). Here, we have highlighted EMBESTIDA (Tackle) and all the pointers associated with this attack.

The Broad Experience of a Randomizer

When a player wants to use a randomizer, they will need a ROM of the original game as well as the randomization tools for that game. A randomized game can usually be played on the original hardware they were intended to run on, but you will need some way to make the game data available to the randomizer program. This is most commonly achieved via a ROM file. The randomization program can take several forms, such as a standalone .exe file or a web application.

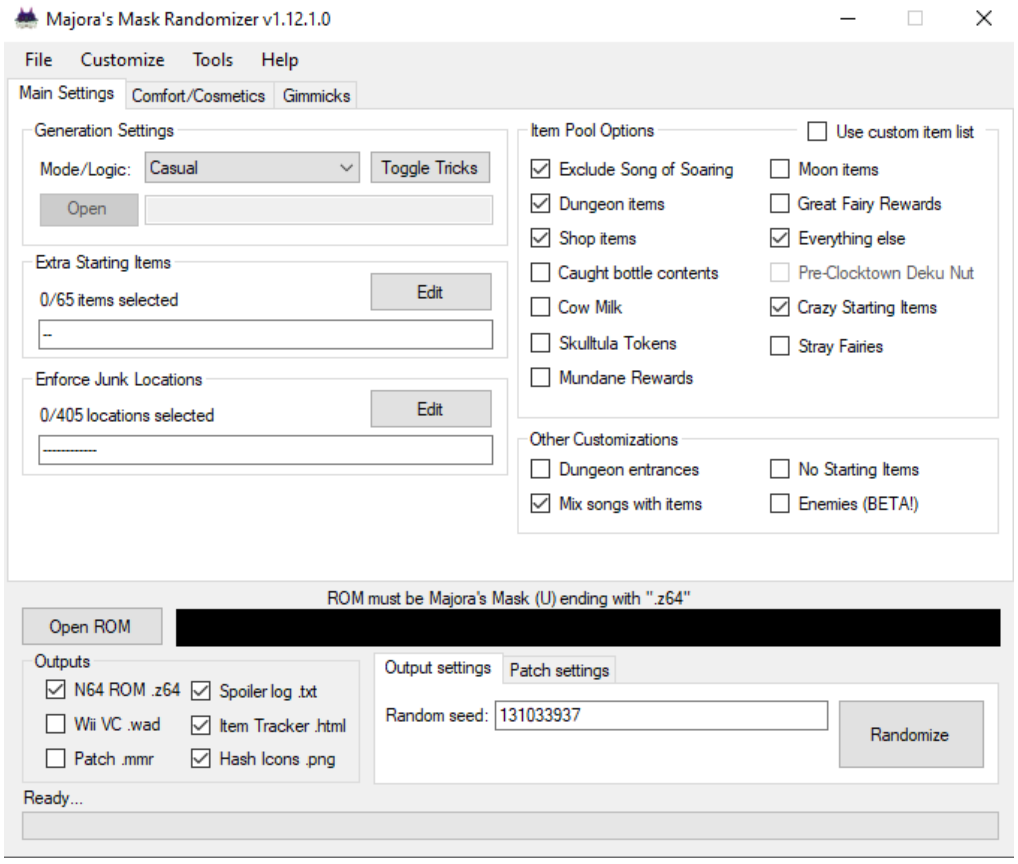


Fig 2: The user is greeted with this upon booting the Majora's Mask Randomizer(ZoeyZolotova, github.com). There are tabs that deal with the primary logic of the randomizer, ie, the settings that will be randomized. In subsequent tabs you can change things that do not directly impact the gameplay, such as the hero's clothing color, the UI color, sound and music, and the amount of damage that is done to the player. Beneath this, the player is allowed to edit the logic of the game. Where items will be allowed to show up and in what order depending on their ability to execute complicated and unintended tricks. Beneath that are the settings required to locate the ROM image and where and how to output the randomized game.

Randomizers generally allow players to choose what options will be changed. The available settings in these randomizers often cater towards the speedrunning community so there will be allowances for varying degrees of difficulty. This challenge may be exemplified by highly proficient play within the expectations of the game.. A randomizer might have extra settings that increase the power level of adversaries, or add hazards to the world that must be avoided. Another way a randomizer will ratchet up the challenge can be through changing the program's logic. Harder logic settings often require a player to perform tricks, exploits, and/or glitches that allow for progress to be made in spite of the barriers originally designed to impede progress. It is not uncommon for randomizers to include other options for the player that are less integral to gameplay, such as aesthetic changes like mixing up music or UI or quality of life changes such as mapping commonly used inventory items to otherwise unused buttons.

In Figure 2, we have the Majora's Mask Randomizer. The first thing the player sees in the top left is the logic setting. The available settings are:

1. **Casual:** The randomizer will shuffle the locations of where items are found in a way that will allow the player to progress without the execution of unintentional exploits or glitches. This setting is ideal for those already familiar with *Majora's Mask*.
2. **Glitched:** The randomizer shuffles item locations with the expectation that the player can perform many glitches and exploits. For *Majora's Mask*, this includes "Bomb Hovering" which allows Link to move infinitely vertically as long as he has explosives and the Hidden Owl Statue glitch, which immediately opens up all fast travel nodes in the game.
3. **Vanilla Layout:** Items are not shuffled. This option exists if players would like to take advantage of other aspects of the randomizer such as aesthetic changes or quality of life improvements, or if they would like to participate in one of its challenge options. These options are found under the comfort cosmetics tab and the gimmicks tab of the randomizer respectively, see figure 1.
4. **User Logic:** This option allows for the manual setting of item locations. This is often used for a curated experience intended for someone other than the user. This is also commonly used for randomizer races.
5. **No Logic**: A total randomization. Items are shuffled with no guarantee that the game can be beaten without the use of extended glitches. In Majora's Mask it is possible to take actions that will directly manipulate the reference

memory of objects. With highly specific glitches you can change the destination of a loading zone to be anywhere in the game. No Logic's lack of progression assurance means that this may be the only means of completing a seed.

Next is Starting Items. This allows the player to manually select items to begin the game with. If the player selected any logic setting other than No Logic these items will be taken into consideration for the logic. Below that is Enforce Junk Locations. This guarantees that a location will not hold a progression item. This is helpful if a player does not want to engage with a specific aspect of the game for whatever reason.

The section on the right includes item categories that the player wants shuffled.

1. **Exclude Song of Soaring:** The Song of Soaring is not required to complete Majora's Mask or the randomizer, however it does cut down on a lot of the travel between destinations. Thus, leaving it in its original spot will theoretically remove a lot of potential tedium.
2. **Dungeon Items:** Any items found within a dungeon. These most commonly include keys, maps, compasses, and some type of bow and arrow upgrade.
3. **Shop Items**: Any item sold in a shop. *Majora's Mask* contains several shops where the game's currency (rupees) can be exchanged for items.

4. **Caught Bottle Contents:** If a player has a Bottle Item, they can use it to pick up a variety of substances. This setting randomizes what each substance is. In other words every time a player might scoop up some fresh water they will instead get a potion.
5. **Cow Milk:** When the player learns how to play “Epona’s Song” they can play it for cows to receive milk. This option means that a cow may give an item instead of milk. This also will cause some other item location to contain cow’s milk.
6. **Skulltula Token**: In *Majora’s Mask* there are several monsters known as Gold Skulltulas that draw golden tokens. Collecting 30 of them will yield a prize. This option shuffles the golden tokens into random locations and causes the golden skulltulas to drop items other than tokens.
7. **Mundane rewards** are a category of rewards that mostly include large sums of money and expendable items like ammunition or bombs. This setting shuffles these rewards into the potential item pool.
8. **Moon Items:** Certain items are located in the moon, the game’s final location that is inaccessible until the player has completed all other dungeons. This shuffles the moon items into the pool. The moon items are never necessary to complete the game normally.

9. **Great Fairy Rewards:** Throughout the game there are 5 great fairies which yield 6 rewards if you collect all of their stray fairies. This mixes the rewards into the pool.
10. **Everything else** puts all rewards not already specified into the pool. This includes most sidequest rewards, mini game rewards, and questline items outside of dungeons into the pool
11. **Pre Clock Town Deku Nut**: Before the player enters Clock Town in the base game they will obtain a Deku Nut. The randomizer starts the player in clock town for logic reasons and in the example here there is no way to backtrack to before Clock Town, thus this option is grayed out.
12. **Crazy Starting Items:** In the base game the player starts with 2 heart containers, the deku mask, the song of healing, a sword and a shield. This shuffles these four items into the pool causing the player to potentially start with only 1 heart and no arms or armor, with six random items in their place.
13. **Stray Fairies**: As mentioned above, there are 5 Great Fairies that will yield an item after returning their stray fairies. Each Fairy has 30 stray fairies located in each of the game's dungeons except the first one which only has 1 stray fairy located in Clock Town. This shuffles all fairies into the item pool.

Beneath this are other customization options. From here, you are able to shuffle which dungeon entrances lead to what dungeons, start the game with 0 starting

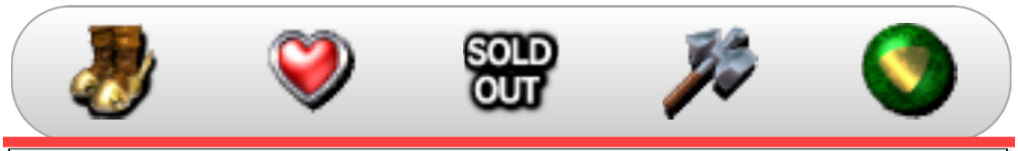
items instead of 6, include songs learned into the item pool, and randomize the locations of enemies.

Once a ROM image has been designated and the settings of a randomizer selected, the player is given the opportunity to input a seed. Seeds are values that the randomizer uses to generate a predetermined outcome for whatever is being shuffled. The seed is made available to the user in order for multiple people to play the same iteration of the randomized, transformed game. This is particularly useful for one of the biggest randomizer niches, competitive racing. By using a shared seed value, participants in the race are guaranteed the same randomized game, enabling competitive play.

Fig 3: The Majora's Mask Randomizer, in addition to changing the locations of items, allows for an increased presence of random enemies. This will make the generated ROM considerably more difficult than the base game (Majora's Mask, Clock Town)*. Here the player is met with a number of optional challenges. Their starting health has been randomized to a single heart and an enemy known as a Beamos has been added to North Clock Town, an area which typically has no monsters and is considered a safe place in the base game.*

Once a file has been created, the player is free to play it as they would any other game. While the actual experience of playing any specific randomizer will vary as widely as choosing to play any game, there are certain commonalities amongst these games post-generation. It is not uncommon that alongside the newly randomized file a "spoiler log" will be created. Spoilers are a colloquialism that refers to a surprise being exposed before the intended time. In the context of

randomizers this refers to aspects of a randomizer like enemy stats or item locations. These often take the shape of .txt or .html files documenting all the changes made to the base game. If a player gets stuck, it can be difficult to rectify since the base game usually does not have any accommodation for randomized elements.The spoiler log can be referenced by the player to progress in case they become stuck. Streamers who play live for an audience may hand the spoiler log off to a moderator or friend to help keep the show running smoothly. The other thing that is created on generation is a seed hash. This is a code that represents the seed and can be shared to other players who want to play the same randomized instance of the base game. This can be especially useful for racing communities to easily distribute instances.



South Clock Town		
Clock Tower Entrance	☑	Don Gero's Mask
Clock Town Scrub Trade	☑	Letter to Kafei
Postbox	☑	Swamp Skulltula Spirit
South Clock Town Final Day Chest	☐	
South Clock Town Straw Roof Chest	☐	

Fig 4: (top) The seed hash for the Ocarina of Time randomizer uses a code represented by items available to the player in the game and is created when an instance of the randomizer is generated. If this code is given to the randomizer before generation, it will generate the same instance. (bottom) A small section of the spoiler log from a Majora's Mask randomizer instance. The top title represents a discrete map separated by loading zones. The left column indicates the areas where items can be (often referred to colloquially as "checks"). The right column indicates the item that is in that location in

this instance of the randomizer. It is blacked out by default but can be revealed by selecting the check box.

Randomizer Taxonomy

When considering the design of a randomizer, it's useful to examine which aspects of a game are being randomized and how that impacts gameplay. A common way a game can be shuffled is through data substitution. This is the process by which the randomizer will take relevant game data and modify it based on the random seed generated by the randomization program. The data that the developer chooses to substitute will vary heavily on the mechanics of a game. Developers of randomizers centered around RPGs may, for example, shuffle the stat growth and skill set of a character to present novel challenges and choices to a player. This is seen as the primary way in which that game's mechanics are experienced. Randomizers may also choose to shuffle multiple aspects of a game which will allow for a greater degree of variation on the base game. Going back to our example, while the numbers associated with character power might be the primary thing being shuffled, it's not uncommon to see enemy attributes being changed as well. A player might find a late game enemy showing up much earlier than expected. This kind of change may require adjusting the range of values the randomizer considers acceptable in order to keep the game balanced and the player adequately engaged. A late game enemy that shows up earlier might have weaker versions of the expected abilities so as not to pose an insurmountable threat. Many randomizers

will also purposefully change aspects of a game to better suit a randomized playthrough. A game where important progression items are shuffled will often remove story dependent triggers to open the game world up based solely on the player inventory. These types of gameplay decisions usually exist to take into account the player who has already played through the game being randomized and is seen as a way to eliminate tedious or repetitive aspects of the base game.

An example of a randomizer which includes primary randomization, secondary randomization, and traditional ROMhacking is the *Earthbound* Randomizer by Stochaztic. *Earthbound* is a turn-based JRPG and one of the primary aspects of its gameplay is a battle system where the player must match up the strength of their characters against enemy monsters. By randomizing the stat growth and abilities of their base party, a player may need to develop new strategies for combat that diverge from the unmodified game. The inclusions of a ROM-hacked “Ancient Cave" mode causes all loading zones to be shuffled as well. This changes the story driven game to a more open ended dungeon crawler where the challenge becomes an ever increasing wager on whether the player can successfully punch above their weight class. In order to facilitate this overcoming of odds a key modification has been purposefully authored into the game as well. The text speed can be made to print instantly. In Earthbound, a player's HP is not lost instantly, but rather upon taking damage it begins to tick down like a car’s odometer. This theoretically gives

the player a window of time to react by either quickly healing their injured character or defeating the enemy before their HP reaches 0. In practice the base game with the fastest text printing options enabled can be a slog to advance through and thus it is hard to effectively engage with this mechanic unless the player has been mortally wounded from a very high HP value. Stochaztic's option to print text instantly ameliorates this problem allowing decisions to be executed at breakneck speed. Thus, it is even more viable for a player to take on enemies with massive power differentials if they can make decisions quickly enough.

Below is a diagram I have provided to show roughly the tipping point of where a randomizer begins to diverge sufficiently from a game to develop its own community. I have marked a position in between the *Digimon World Randomizer* and the *Super Mario World Randomizer* as where this transformation occurs. The gameplay of the *Digimon World Randomizer* is too similar to the base game to generate its own community despite reasonable interest in the game itself. With the *Digimon World Randomizer* the player's engagement with the systems and gameplay is not meaningfully different in spite of some different outcomes of play due to randomization. Contrast this with the *Super Mario World* Randomizer where the player will need to start considering strategies for the game that would otherwise not be seen in the base game.

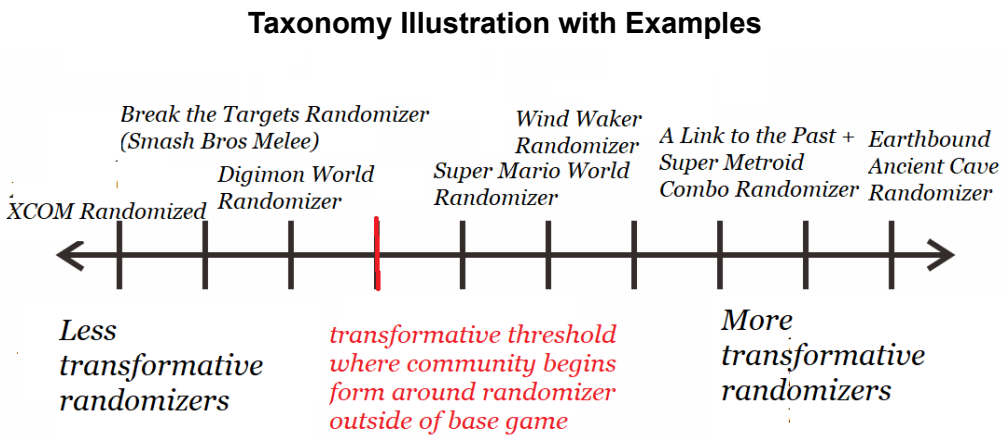


Fig 5: Randomizer taxonomy displayed as a spectrum.

XCOM: Randomized

XCOM: Enemy Unknown and its expansion *Enemy Within* is a turn based tactical/strategy game where the player must defend Earth from an alien invasion *(Wasteland Ghost, nexusmods.com)* The game has two primary modes: a grid-based tactics layer where the player takes the role of the commander and issues orders to a squadron of four to six soldiers, and a strategic “ant farm” layer where they must manage the paramilitary organization’s finances and facilities. In the strategy layer the player is responsible for selecting technologies to research and weapons to build, allocating energy, securing funding from the world’s nations, and ensuring that the Earth’s airspace remains UFO-free. Most importantly, in this layer of the game the player must decide which missions to accept. It is not possible to accept every mission presented and thus the player will have to weigh the

rewards of any one mission versus the opportunity cost of missing out on others. The strategic layer also contains the game's failure state. If a significant portion of the world's nations lose faith in you as a commander due to your inability to complete missions and destroy UFOs, the game is over.

Fig 6: The strategic layer of XCOM: Enemy Unknown is colloquially known as the "ant farm." Here you are presented with a cross section of the base with a selection of facilities you can interact with. Toward the top are facilities that are automatically populated in each game, such as the hangar, barracks, and R&D facilities. The bottom two rows of facilities are entirely dedicated to whatever the player would like to construct to aid in their efforts to stop the aliens. As the game progresses, an additional two layers of customizable space open up.

The tactical layer takes place on a grid. In the unaltered game, the player will deploy on each map in a set location while the enemy's exact position is hidden beneath a fog of war. The squadron must complete a designated objective, be it routing the

alien forces, rescuing civilians, or disposing of alien bombs. To achieve this the player will engage in firefights with the aliens, the outcome of which is largely based on who is able to take the superior position on the battlefield. Players and aliens alike also have special abilities to supplement their skills with firearms. The player's soldiers specifically are sorted into classes which will determine what types of skills they will learn on the battlefield. The sniper class, for example, may learn skills that allow them to increase their maximum effective firearm range as well as a special skill which allows them to target vital points on the alien lifeform's body. These skills are learned through successfully completing missions, so a veteran soldier will become a highly valuable unit. Soldiers in this game that die in combat are permanently removed from the game, thus making your soldiers a precious resource.

Notably, the expansion *XCOM: Enemy Within* includes options for randomization even in its unmodified form. When beginning a new game, the player has the option to select several options from a "second wave" menu. This allows the player to randomize the contributions nations will make to XCOM, the skills each soldier will learn regardless of class, and the stat growth of each soldier.

The XCOM randomizer mod, known as *XCOM Randomized*, exists on the end of the taxonomical spectrum that represents the least transformative degree of

randomization. Unlike many randomizers, XCOM Randomized is a mod that is integrated into the base game rather than a program that shuffles the game data before run time. There are no configurable options for the user to decide what is randomized at any point.

XCOM Randomized primarily changes the position of key elements on the map of the game's tactical layer. In *XCOM*, when you accept a mission, you deploy a squadron of 4-6 soldiers to complete an objective. In the un-randomized game, each map has fixed elements that a player can anticipate as soon as the map is revealed. These elements include starting location, enemy spawn placement, resources, and enemy reinforcement drop points. In the un-randomized game, the player is able to easily set up ambushes in places they know the enemy will appear. Because positioning is so vital to the game's tactical layer, establishing an advantage before an engagement with hostile aliens can end a conflict before it starts. *XCOM Randomized* makes these exploits much harder to take advantage of, thereby forcing players to rely less on their ability to memorize key map locations and more to strategize around new situations and mitigate risk

Fig. 7: In the unmodified game, the player-controlled covert operative unit (selected above) would be spawned much further from the squadron. However, in this example, XCOM Randomized *has spawned them in close proximity to the player's squadron.*

XCOM Randomized is minimally transformative. It only affects a part of the game and it does not fundamentally change how the game is played. It focuses the gameplay more on the key skills needed to be successful. Players will still progress through the game in more or less the same manner as a vanilla playthrough. Many aspects, like the alien statistics, weapon statistics, times in which you encounter aliens, and skills soldiers learn are not changed.

Super Mario World Randomizer

Super Mario World is a 2D platformer for the SNES.The majority of the game is spent trying to navigate Mario through a series of challenges by running and jumping over obstacles such as pits, spikes, and lava. Mario is also impeded by adversaries who can dispatch him with a single hit. To mitigate his fragility, Mario can find power ups within the level that will allow him to take an additional hit, fly, shoot fireballs, or ride on a dinosaur named Yoshi whose long tongue allows him to eat many of Mario's enemies. Progression through the game is indicated via Mario's traversal of an overworld map. The map here does not include the same platform-centric gameplay but rather serves as a metaphor for level selection. New paths on the map open up for every level Mario completes, and divergent paths may be opened if the player is able to find secret exits in certain levels.

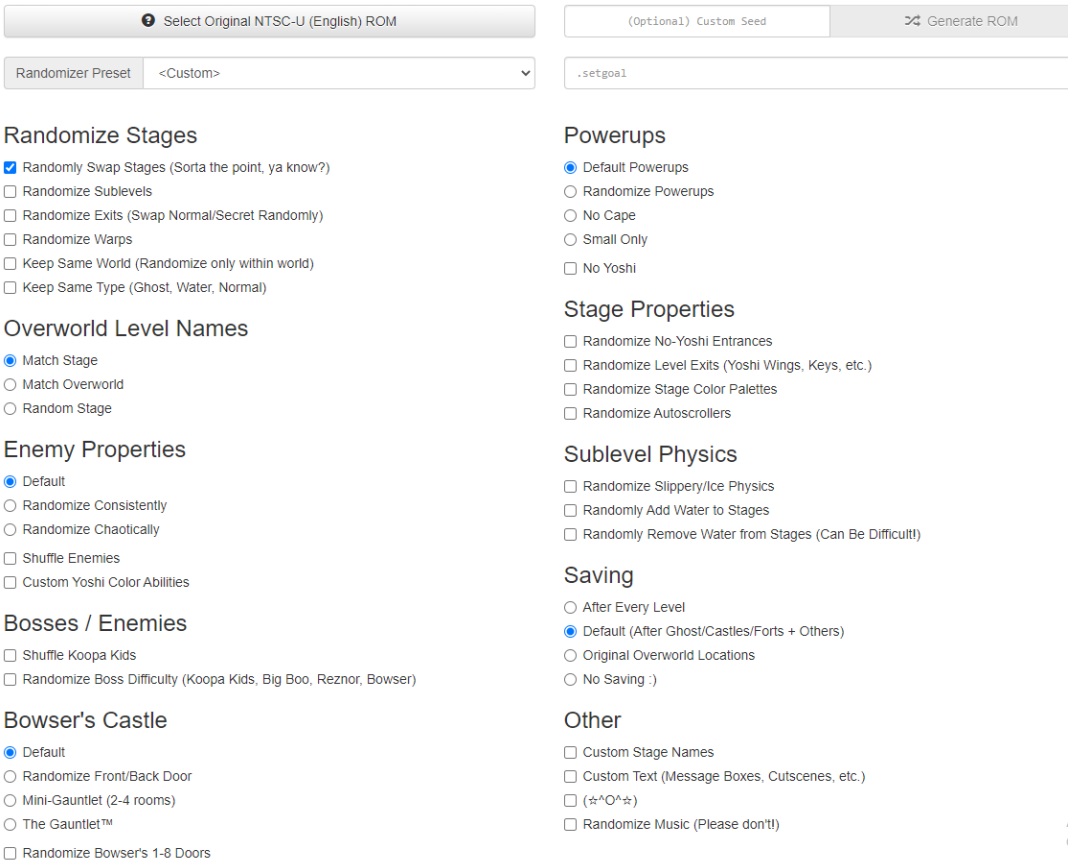


Fig 8: This screenshot shows all of the Super Mario World Randomizer *options* (Authorblues, github.com). *The options change the progression of the player throughout the levels, what enemies they will encounter within those levels, what powerups will be in those levels, and certain physical properties of both levels and sublevels.*

Super Mario World Randomizer is both browser-based and downloadable. This randomizer shuffles the order of the levels, the properties of enemies, the physical properties of levels (the friction of the floors, whether or not the level is filled with water or not, if the level is an auto-scroller or not), and so on. While layout aspects

such as platforms, pits, elevated land, doors and checkpoints of any given level are the same, the shuffling of obstacles in a level will inherently change the way a player must navigate through a level to make it through to the end. Another significant change is the shuffling of sublevels. While the levels within loading zones remain the same, the composite of a level that is created by sublevels will keep the player guessing whenever they enter a pipe.

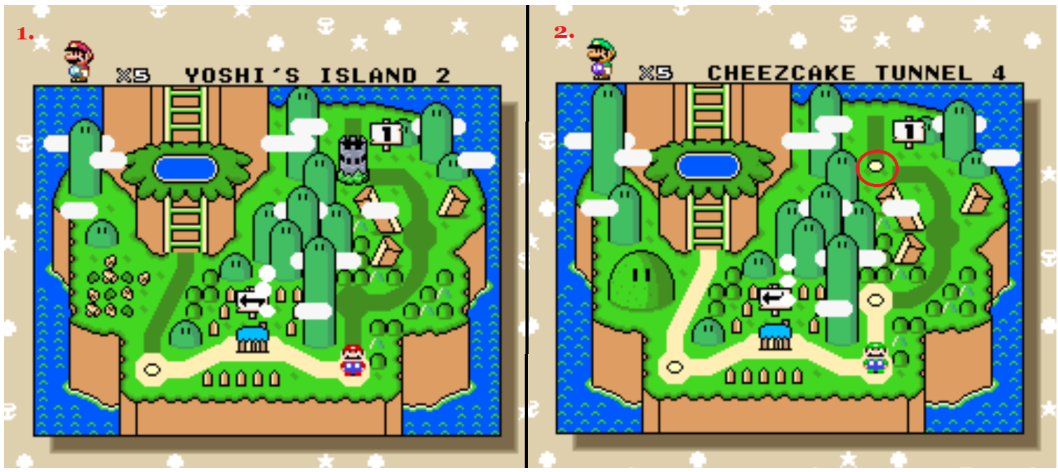


Fig 9: The overworld map in Super Mario World *(left) vs the* Super Mario World Randomizer.*(right) remains largely unchanged however a few slight alterations can be noted.*

Of note in the randomizer is the ability to randomize miscellaneous options. Some are cosmetic, others less so. In Fig xx we see an “Other” category. These options are as follows.

1. Custom Stage Names: In Fig (xx) we can see a side by side comparison of the same stage. On the left is the unchanged stage name, Yoshi's Island 2. On the right this same stage is referred to as "Cheezcake Tunnel." This stage name is meant to imitate the absurd naming conventions of *Super Mario World's* Food Place structure.
2. Custom Text Box Messages. Changes the normal messages from hitting the text boxes in the game to custom ones.
3. *(☆^O^☆):* Enables Pogyo Mode. Pogyo refers to the Japanese Super Mario World Speedrunner Pogyo, and this game mode contains inside jokes as nods to this community member. Changes include
    - Random objects thrown by hammer bros
    - Random items on Fishin' Lakitu's fishing line
    - Random Ball & Chain and rotating platform speed.
    - Several numerous small cosmetic changes such as sprites, and sound effects.
4. Randomized Music: Changes what music plays where. Might be considered grating to some people.

Fig 10: In this picture, Mario is swimming through a level with a P switch. Typically, a player would be expected to use the p switch to convert the coins into solid bricks to allow mario to traverse over the lava. The fact that the randomizer has allowed the player to keep the p switch and use it for a later area allows for emergent strategies to navigate other parts of the level successfully.

Amongst the options available in the *Super Mario World Randomizer* is the option to change physical properties of sub-levels. Physical changes to levels include the friction of the floor, if the level is filled with water (or drained of water if it was

originally filled), or if the level's camera is constantly moving in some direction (known colloquially as an auto-scroller). These changes may drastically affect how the player will have to traverse the level. A level that is filled with water will allow the player to move vertically without the restrictions imposed by gravity, thereby trivializing any pits that may exist below. Conversely, a water level drained of fluid will be very demanding of the player. There may be enormous gaps or towering cliffs that will require highly competent play to overcome. This is transformative to the experience of *Super Mario World* to a degree, but ultimately the game remains a linear game centered around platforming challenges.

The Legend of Zelda: Wind Waker Randomizer

The *Legend of Zelda: the Wind Waker* is an action adventure game that was initially released on the Nintendo Gamecube. The game focuses on the player character Link as he attempts to save his home from the return of the evil warlock Ganondorf. To complete this goal, the player must navigate a vast sea and seek out ancient treasures to properly arm themselves against Ganondorf. In its unaltered state, the game's critical path is well outlined by NPC dialogue. Where the player must go at any given time will be directly referenced by the story and if ever the player needs reminding they can consult with their talking boat. The treasures obtained by Link will expand his abilities in how he is able to traverse the dungeons and other perilous islands that dot this vast ocean. Link may find a grappling hook to scale

cliffs and swing across gaps, a “Deku Leaf” to help him glide and move vertically through wind currents, or a hammer to smash boulders that would otherwise impede his path.

LagoLunatic’s *Wind Waker Randomizer 1.9.0* primarily shuffles where key progression items appear *(LagoLunatic, github.com)* While Link’s primary quest is to acquire 8 pieces of a magical artifact, gameplay-wise this artifact does little more than act as a glorified key. Progression throughout the game is gated by Link’s abilities to traverse the environment, key quest items, and literal keys for locked doors. Within the randomizer you are given the option to shuffle these according to groups known as “item pools.” An example of an item pool would be “small keys,” the most common key that can unlock only a single door in a specified dungeon. When this pool is included in randomization (a setting referred to as “Keysanity” in this and multiple other randomizers from *The Legend of Zelda* series) it will change how the player progresses drastically from the base game. When unaltered, dungeons in *The Wind Waker* are meant to be completed in a single visit, since all the keys for a dungeon can be found within that dungeon. When Keysanity is enabled, a player may be expected to go in and out of a single dungeon multiple times. A player may find a small key for a dungeon in the overworld, visit that dungeon to find a randomized progression item, leave the dungeon to use that item

elsewhere in the world and thereby uncover more small keys for the aforementioned dungeon.

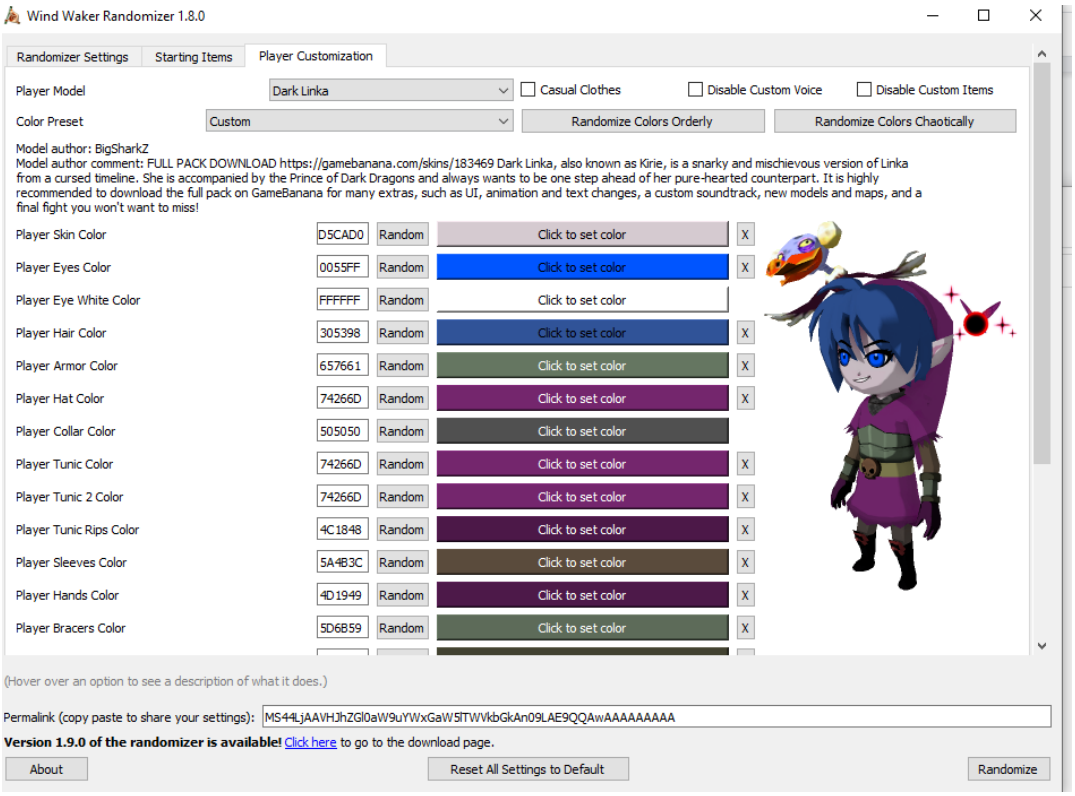


Fig 11: Many randomizers in the Legend of Zelda *series allow for cosmetic alterations but none are as robust as the randomizer for* Wind Waker. *The character cosmetics allow the player to import custom models that often have multiple outfits; not only that, but many if not all color values on a character's palette can be altered.*

Wind Waker and many other games in *The Legend of Zelda* series are prime candidates for randomization because of the particular interactions programmed into the game. Items tend to have multiple uses. The Deku Leaf functions primarily as a glider but also can be waved to generate a burst of wind that can move light objects and stun certain enemies. In the unaltered game a player is likely to have the same tools in the inventory at each point of progression across playthroughs. In the randomized game, a player may never have the same combination of items in any given situation across instances. However the multi-use nature of many of the game items encourages the player to seek out emergent interactions in order to progress further through the game. An evocative example is the bottle. Empty bottles are used primarily to store substances such as potions or soup or even small fairies. However, in the *Wind Waker* in order to put something in the bottle Link must swing the bottle in front of him to scoop it up. Strangely enough, swinging the bottle also reflects certain projectiles. Having the bottle in conjunction with the Skull Hammer will allow the player an unorthodox opportunity to defeat a boss called "Phantom Ganon," whose lightning projectiles must be reflected back at him in order to render him vulnerable to attacks.

Fig 12: Within the settings of the Wind Waker Randomizer there is the option to completely remove the sword from the game. Here the player (referred to as "Fox Only") is about to confront Phantom Ganon, which is typically only beatable with the Master Sword. The Randomizer includes a hack that allows the player to also defeat him with the Skull Hammer, as indicated by the King of Red Lions (a talking boat).

Super Metroid & Link to the Past Combo Randomizer

The Link to the Past / *Super Metroid* Randomizer is an uncommon type of randomizer known as a combo randomizer. This is a randomizer which combines two different games. For this specific example, these two games are able to be joined together thanks to a lucky coincidence in how the memory of these games

are stored. The SNES includes 256 memory banks to store data. After memory for the WRAM, CPU, PPU, Joypad, and Auxiliary has been accounted for, the remaining banks are dedicated to ROM memory from the cart *(Arsenault,113)*. Developers for SNES games must assign data from their game these specific addresses in memory. It just so happens that the memory banks used by *The Legend of Zelda: A Link to the Past* has 0 overlap with the memory banks used by *Super Metroid.* This allows these two games to be composited. The developer for the Super Metroid & Link to the Past Combo Randomizer explains it thusly.

> "The Super Nintendo's memory bank...is split up into 256 segments. *Link to the Past* and *Super Metroid* were both coded to occupy different groups of these segments, perhaps coincidentally. If they had overlapped, they wouldn't be able to share a ROM file, but because they each took up different parts of the SNES memory, they could work side by side." *(Tewtat, Kotaku.com)*

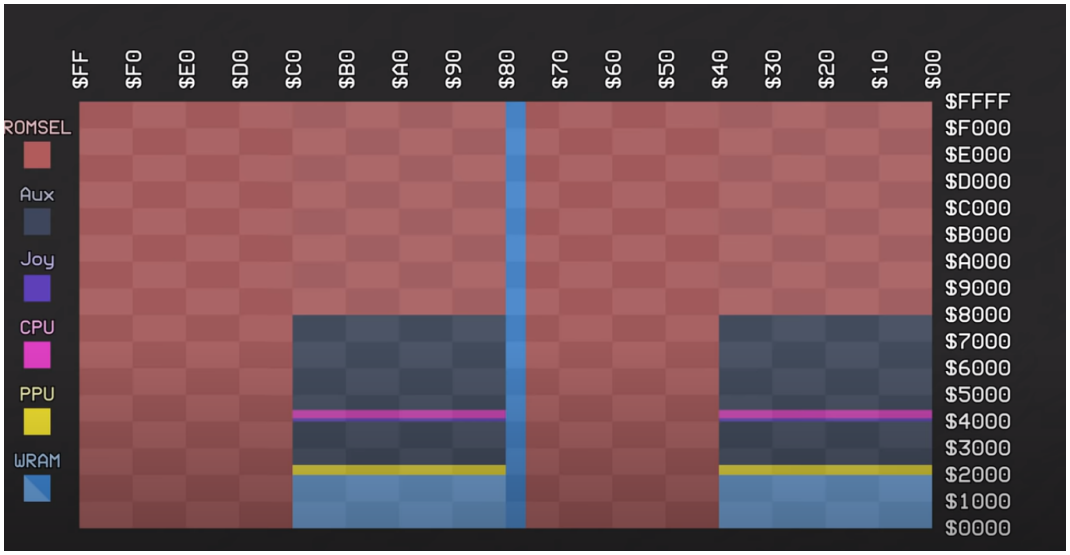


Fig 13: A visual representation of how the memory banks of the SNES are allocated. Each tile represents a bank of memory that itself contains a range of hex addresses. The red sections are entirely dedicated to accessing data from the cartridge. (Retro Game Mechanics Explained, youtube.com)

These two games also fit well together due to certain gameplay similarities. While disparate in both tone and fictional genre, both games involve exploring a fairly open ended world in search of items to continue progression. The item seeking nature of the *Wind Waker* also applies to *A Link to the Past,* an earlier entry in the same series. Notably, *Super Metroid* also requires the player to search for upgrades to aid progression, but is played from a side scrolling perspective, unlike *A Link to the Past*. This places more emphasis on navigating vertical spaces and thus requires a player to grapple more with forces like gravity and momentum.

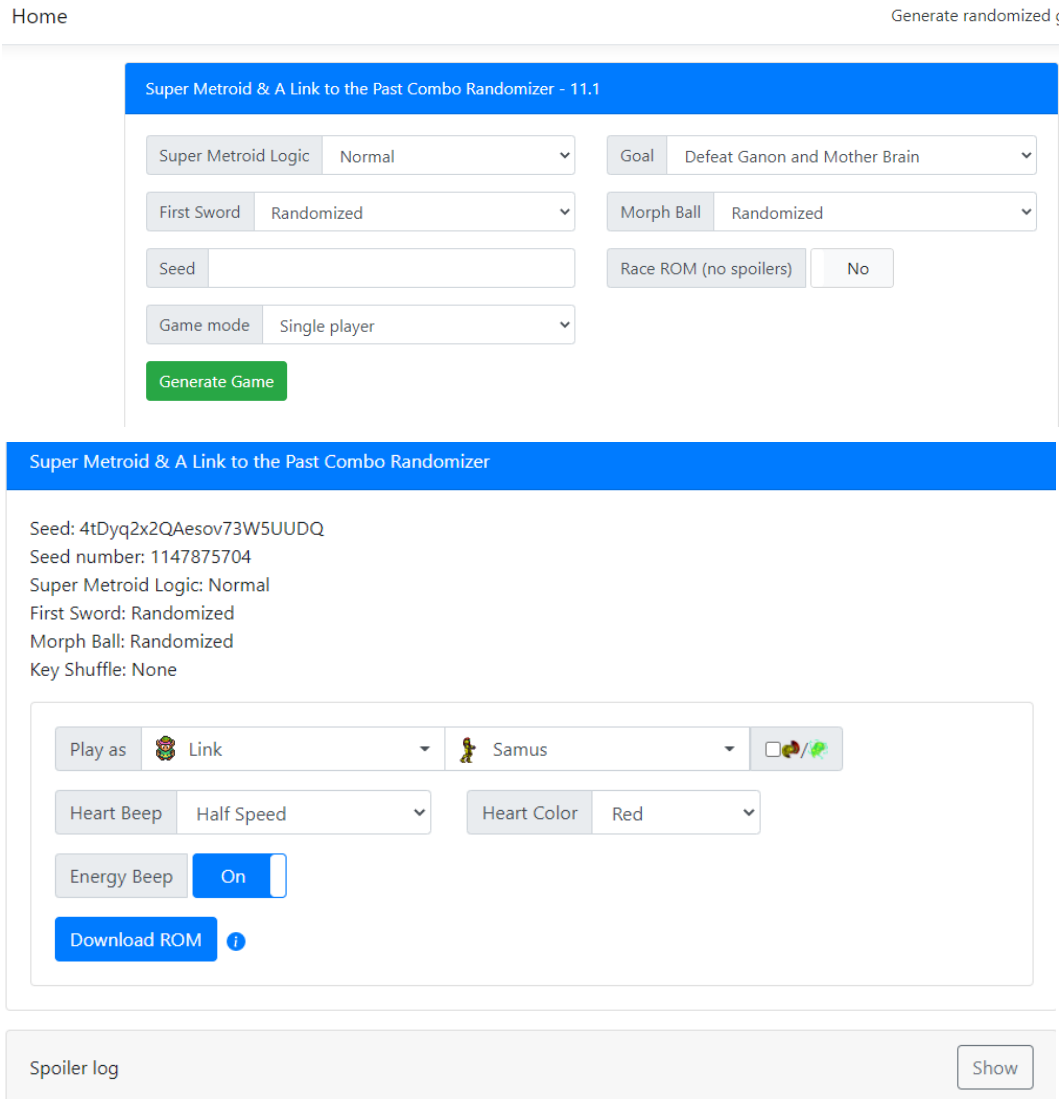


Fig 14 These two screens make up the entirety of the Super Metroid & Link to the Past Combo Randomizer. *The above screen includes the settings for the game's logic. Logic once again determines if the player will be expected to use any extended gameplay techniques to progress. The first sword and morph ball may appear, at the player's discretion, in totally random locations, easily available locations, or in the player's starting inventory. These items are required for much of the progression throughout these games. The second screen is small aesthetic changes, allowing the player to change their sprites in the game as well as UI elements* (tewtal, samus.link).

The randomizer not only shuffles items around in both the worlds of Hyrule (the setting of *A Link to the Past*) and Zebes (the setting of *Super Metroid*), but shuffles them between these two worlds. This expands the world of play considerably and requires the player to keep track of many more gates on progression from important items. As of version 1.1, the locations where the player can move between the top-down world of Hyrule and the sidescrolling world of Zebes are fixed. Nevertheless, being able to switch between these two playstyles is important to this randomizer's gameplay literacy.

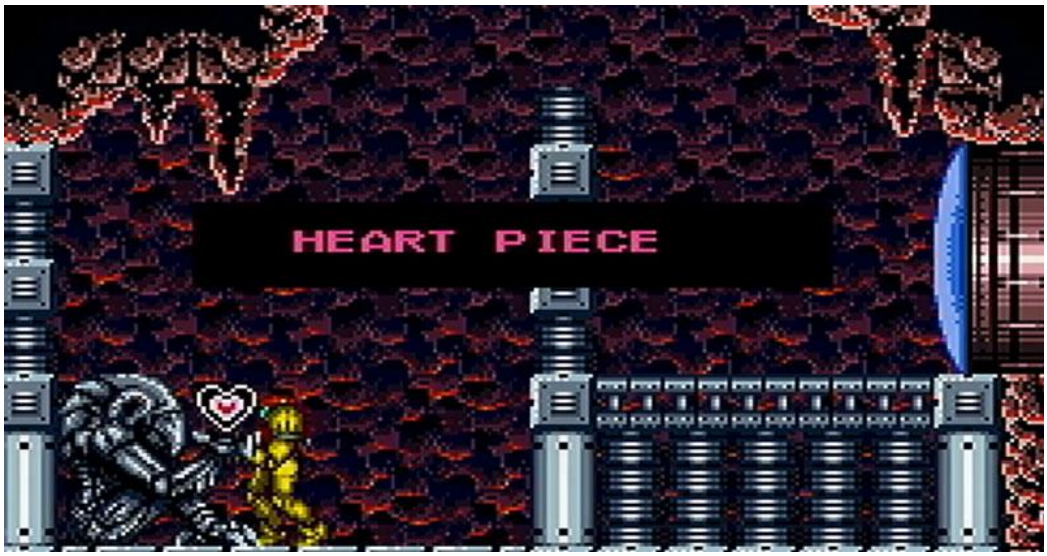


Fig 15 A heart piece is a health upgrade for Link specifically. Here we see Samus finding it in Zebes, indicating that items normally intended for Link can be found in the gameplay portions involving Samus.

Fig 16: This fortune teller's house is one of three connection points between Hyrule and Zebes. The areas chosen are considered inessential, such as a fortune teller or a map room.

Earthbound Ancient Cave Randomizer

At the extreme transformative end of the spectrum, The *Earthbound* randomizer changes many aspects of the game, and also helps to bring mechanics of the game to the forefront to seriously change player experience. *Earthbound* itself is a JRPG with a linear story. JRPG stands for Japanese Role Playing Game, and while the exact meaning of the term is nebulous it generally refers to any Japanese influenced game with largely menu driven combat and character progression. *Earthbound* stands out as unique amongst its peers for taking place in a setting akin

to modern America, which on release in 1995 was considered novel for the JRPG genre. The player's power grows steadily throughout the game to scale with the increasing deadliness of the enemies. The randomizer allows for many gameplay and aesthetic changes. Character power growth and enemy power are randomized. Moreover, the skills the enemy can employ are shuffled so a player cannot necessarily expect a given enemy to have its standard capabilities in a standard playthrough.

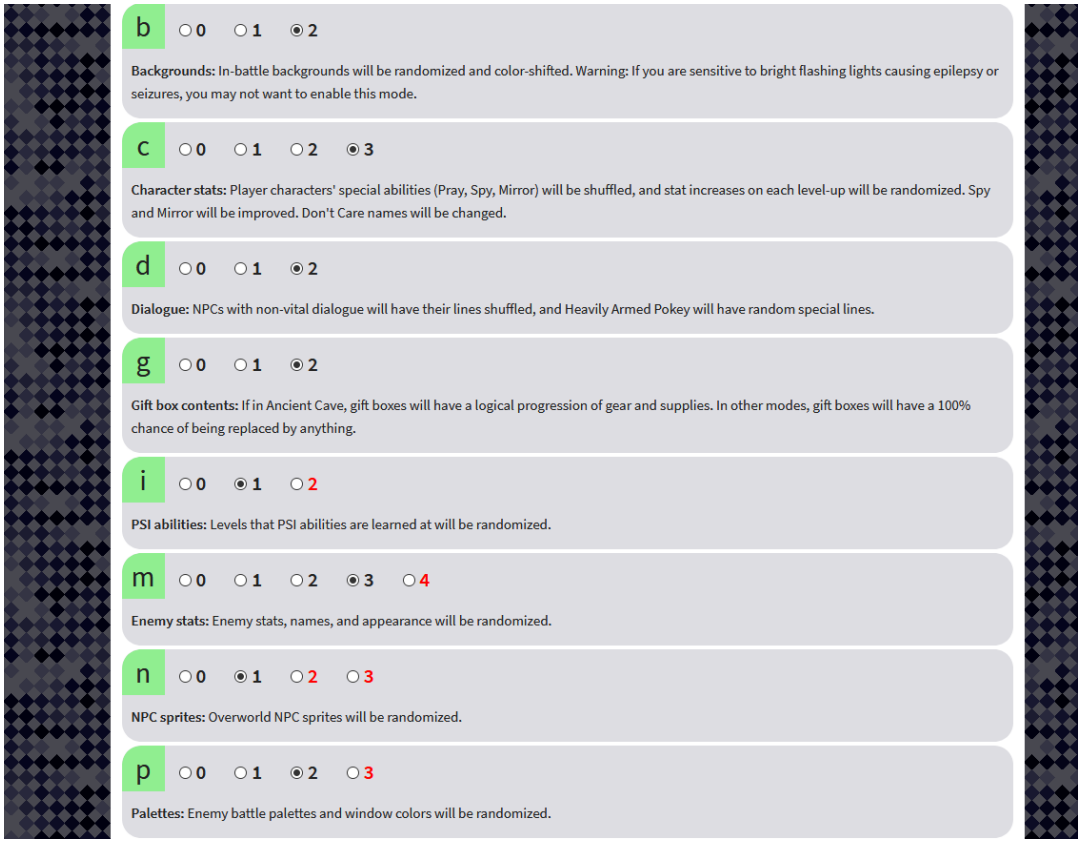


Fig 17: A selection of options from the Earthbound Randomizer Web App. Red numbers indicate a more drastic degree of randomization but also may cause the game to be unstable by inducing unbeatable game states and game crashes (Stochaztic, earthbound.app)

Map entrances and exits are also shuffled, meaning that an exit from one map will correspond to a random entrance from another. This means that the world of *Earthbound* is deconstructed and rebuilt into a massive labyrinth, giving the game more of a dungeon crawler feel. (Dungeon crawlers are differentiated from JRPGs like *Earthbound* because they place a greater emphasis on experiences like exploration through a nonlinear environment and resource gathering and management.) What's more, the world is divided into 9 distinct layers, with each layer containing an upper limit to enemy power and quality of items the player can find. The many paths within each layer will converge on a single exit that will lead to the next layer, which is guarded by a boss monster. However, the player may also take “shortcuts” to skip multiple layers if they find items that would be considered “key items” in a standard playthrough. For example, the “Pencil Eraser” is an item which gates progression in the base game. In the context of the randomizer, finding the Pencil Eraser and using it to open up a path can transport a player several layers deeper into the dungeon than what would otherwise be possible following the standard path. The developer of this randomizer, Stochaztic, took inspiration from the *Lufia 2 Abyssonym' Randomizer's* world design.

Fig 18: Rainy Circle is a Sanctuary typically visited in the base game with 3 party members after completing Fourside. Here, we see a party of four visiting this Sanctuary, indicating its appearance out of order due to the random level construction. It's also worth pointing out that the rain droplet sprites have been randomized here causing a bunch of junk sprites to fall into the water.

Fig 19: The Earthbound randomizer also allows for a high degree of aesthetic changes including character models and sprite palettes. If you are familiar with this game you likely remember the surreal and psychedelic battle backgrounds. These too are randomized which due to the sheer number of possibilities in conjunction with the way these backgrounds are constructed will virtually never result in seeing the same background twice.

Earthbound is notable for its colorful, moving, patterned battle backgrounds. In the base game there is a predefined set of these backgrounds created through compositing multiple layers that animate on top of each other. The simple moving shapes and colors are generated in game and thus can also be randomized. This

allows the Earthbound randomizer to create over 50,000 possible backgrounds across the many possible seeds of the program, the vast majority of which do not appear in the original game. This is one aspect of aesthetic randomization that exists in this game, along with menu color palettes and music tracks. Aesthetic randomization exists in multiple randomizers, but it is separate from the type of gameplay element randomization that might cause a randomizer to mutate from its base game.

An important aspect of the *Earthbound* randomizer are the purposeful and non-randomized game tweaks. One of the most significant is the “Ludicrous Text Speed” option. This allows text to print in a dialogue box instantaneously. This is a quality of life change that also has an important gameplay implication. One distinguishing feature of the Earthbound battle system is that HP is displayed like an odometer. When a member of the player’s party takes damage, it will tick down over time rather than be subtracted instantaneously. In the standard version of the game, this theoretically gives the player time to react if they take mortal damage to either try to

end the battle quickly, heal the dying party member, or escape before the counter reaches 0. In practice, this is often impossible due to the time that it takes the battle text to print even on the fastest setting. In contrast, the option to instantly print text allows the player to navigate through their options at breakneck speeds. This may

give them the opportunity to execute several turns before their life reaches zero. By utilizing their PSI powers to continually heal, thus expanding their Hit Points by levels of magnitude beyond what their low level characters actually have, or by punching far above their weight class with powerful psionic moves, they can defeat enemies that would seem to outperform them simply with a side by side comparison of stats. A player might also be more likely to try niche battle options like “Spy” or “Mirror” to gain an upper hand. Spying on an enemy will reveal its weakness which can allow a player to defeat something that might be too resilient to beat without this knowledge. “Mirror” will allow a party member to transform into an enemy which could magnify their power significantly at lower levels.

The Earthbound randomizer also by its nature involves a risk-reward system with its economy. In the base game, players do not pick up the money used to buy items directly from monsters. Instead, after a monster is defeated, your dad will deposit some amount of money in your bank account which you can withdraw at an ATM. You are also able to deposit money at an ATM if you wish. This is important because if you are defeated in battle, rather than pushing you back to your last save you are revived at the last phone you spoke to your father at with half of the money you were holding at the time. This danger is easily mitigated by only carrying money when you intend to spend it. In an unaltered game, the player will know where the ATMs are since the map is always the same. Most vendors are in the same place as

an ATM but critically there are vendors that are on maps apart from an ATM machine. This fact is vital to the metagame of the Earthbound Randomizer. The inventory of each vendor is randomized, and some vendors could be selling powerful equipment that would be very useful to the player. The issue is that the player has to weigh that reward with the risk of carrying around the money to potentially afford it. Rooms which might contain machines are also randomized, so choosing not to carry around life-altering sums of money could cause you to miss out, but being defeated with that money in tow might be detrimental if you come across a room with both a vendor and an ATM. The extra layer of decision making turns *Earthbound* into a game of risk management with semi-permanent consequences.

Community

To get a complete understanding of randomizers it is important to understand the community that exists around them. Randomizers, being a subset of User Iterative Content, are not created for profit, and therefore the manner by which developers iterate on their design is distinct from that of a commercial product being patched. Randomizers can be enjoyed by anyone, but the designers who are often enthusiastic about the game themselves are targeting other enthusiasts. These enthusiasts make up a vocal part of their community and build events around a randomizer, e.g., competitive races, in order to display their skill at a particular randomizer for the entertainment of an audience. This community focus on skill and competition feeds back into the design, and this feedback tends to be the primary driver of innovation, as Wind Waker Randomizer developer LagoLunatic explains:

> People suggested a number of improvements to make it play more smoothly which I usually would implement. One example is a Race Mode option suggested by people who race Wind Waker randomizers against each other, which makes the randomness slightly less unpredictable so that tournaments can rely more on player skill.

While users and developers of randomizers become involved in the practice through many avenues, there are some common points of entry, especially for the most active community members. These entry points are the speedrunning community, the ROMhacking community, and other randomizer communities. On the whole, there is not a single unified community built around the concept of third-party game

randomizers. There are online spaces that serve as knowledge pools, but randomizer communities tend to form around single games.

Community overlap across games is common, especially between games from the same franchise or similar genres. The developer of the *Ocarina of Time Randomizer*, Testrunner, began his work in the *Legend of Zelda: A Link to the Past Randomizer* community. These two communities share a lot of members, some existing in both spaces at once while others spend most of their time in one then moving on to another. This commonality will sometimes be reflected in the design of the randomizer itself as well. Testrunner and LeggyStarscream (the latter being the developer of *HMSJayne,* the *Final Fantasy 1* Gameboy Advance randomizer) have attributed the randomizer communities they were involved with in the past to be primary examples of design for their own projects. As LeggyStarscream describes,

> (I) worked on two randomizers, both the NES FF1 and the GBA FF1 randomizers. For both of these I worked as part of a team. The FF1 community is more older and established, whereas I was a 'founding member' of FF1 GBA randomizer.

One of the most important resources to the developers of randomizers is the knowledge of a game's data that is accumulated by ROMhackers. Many randomizers are created by ROMhackers themselves. Before Stochaztic created his randomizer, to get an idea of how to structure his project he looked towards Clyde

“Tomato” Mandolin’s Earthbound Randomizer, the *Tomato Reshuffler (Mandelin, earthboundcentral.com).* Tomato is known as a prolific *Earthbound* ROMhacker, having authored some of its most robust tools like *PK Hack*, a series of tools to facilitate the ROMhacking of *Earthbound (Nesluver, starmen.net).* Members of a game’s ROMhacking scene are not uncommon as either developers or advisors to a randomizer community, thereby integrating into it. A good example of this is the co-developer of the Super Mario World Randomizer’s kaizoman666. His technical expertise and experience with ROMhacks helped Authorblues rapidly implement features for their randomizer and is now part of the Super Mario World Randomizer Community.

Speedrunning and Randomizers

A very visible part of a randomizer's community tends to be highly proficient players of the game. This often includes speedrunners, many of whom stream regularly to Twitch. These are players who are already highly motivated to play a single game at virtuosic levels, and the familiar-yet-different experience offered by a game's randomizer is something that appeals to them. Speedrunners like Authorblues and PInk Pajamas have developed randomizers for the games they speedrun.

Speedrunning is a type of gameplay where the player seeks to achieve some objective (usually completing the game, although it can vary) as quickly as possible. Speedrunning as a community activity with rules and records can be traced back to the early 90's with games such as Doom and Quake having groups surrounding them competing for the fastest completion time *(Scully-Blaker, 21)*. As speedrunning grew, the partially self-reported records evolved into a well-documented practice on sites like SpeedDemos Archive. More recently, verification of records with video proof has become the standard. Speedrun.com is currently the largest collection of documented speedruns as well as a repository for most games' community's speedrun rules *(Shadodraft, sjorec, et al, speedrun.com)*. One common practice amongst speedrunners is executing gameplay maneuvers that take advantage of unintentional aspects of the game to progress quickly. This can take the form of finding ways to move the player character faster than normally possible, skipping

large segments of story, or breaking the sequence of events as they would normally occur in a game.

Fig 20: In The Legend of Zelda: The Wind Waker *a speedrunning technique utilized is the "Zombie Hover". The player character is able to move infinitely vertically while in the death state but before the game over state. This allows the player to bypass items that are needed for vertical traversal, saving them time.*

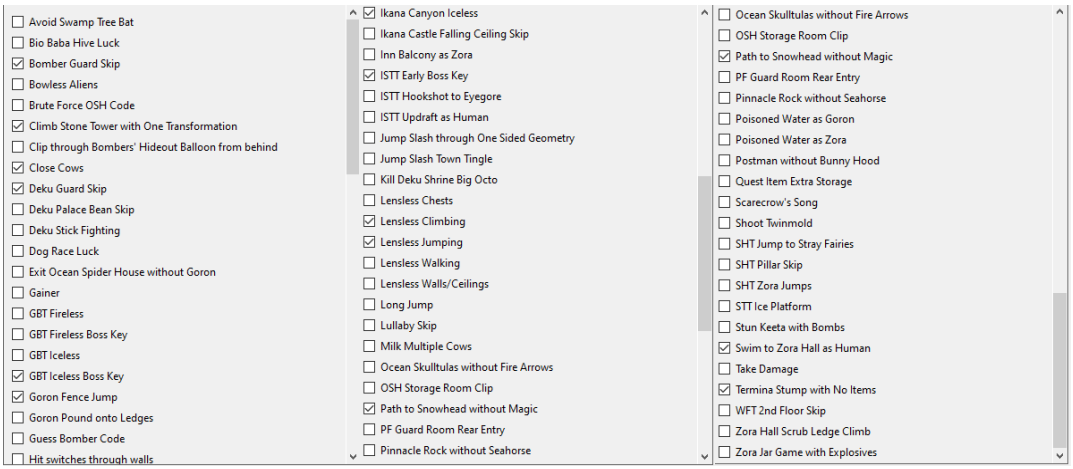
Another important aspect of speedrunning for the purposes of randomizers is the speedrun categories that exist within a game's speedrunning community. These are community-defined rulesets that a speedrunner adheres to within a game. Many games have multiple voluntarily adhered to rules called "categories" *(Boluk and LeMieux, 43)*; some even have dozens. Common categories include:

1. *Any%:* A category where the runner attempts to beat the game as quickly as possible without needing to achieve any other specific objective.
2. *100%:* Broadly speaking, this category requires the runner to complete all optional objectives of a game in addition to the primary one. Certain games have a completion percentage visible during certain parts of gameplay, which will serve as the benchmark for this category. If this is not the case, what constitutes 100% is often community defined.
3. *Glitchless:* Here the player must complete the game without using certain techniques and exploits that are considered “glitches.” These actions are often explicitly defined by the game’s speedrunning community.

Categories may also be combined. The Glitchless category, for instance, may also be subdivided into Any% and 100%, depending on the game.

Fig 21: A non-comprehensive list of all the gameplay tricks and exploits within The Legend of Zelda: Majora’s Mask *contained in th*e Majora’s Mask Randomizer. *Whether the randomizer logic accommodates the players ability to do those tricks can be toggled on and off.*

Randomizers tend to be designed to accommodate highly skilled play. It is not uncommon to see options like figure xx with comprehensive options for a level of execution beyond what the game's original developer intended. Significantly, many of the tricks included in some randomizers also supersede what is useful in many speedrun categories. In the example above, nearly all of the tricks listed are not



required in the Majora's Mask *Any%* and *100%* categories as of May 2021. One trick included in this check list is the Inverted Stone Tower Temple Early Boss Key. By performing a series of side-hops and jump-slashes with very specific timing, it is possible for the player character to land on a small bit of stage geometry which will place them adjacent to the room containing the boss key. If a player can perform this, they can specify it in the settings by checking the box. The randomizer logic will then recognize several items that would otherwise be required to get to that room as unnecessary and omit them from the boss key prerequisites.

Another aspect of randomization that attracts speedrunners is the novelty of each instance. Randomizers will often be featured on a speedrunner's Twitch channel as an alternative to their typical content. This can offer the runner a way to break up the grind of trying to achieve personal bests in their game(s) of choice as well as something different for their audience to watch. Demerine, a streamer and a

member of the *Final Fantasy 4 Free Enterprise* team, sees the “delinearized nature of the FF4 Randomizer as a way to keep coming back to it.”

Racing Scene and Recursive Development

Randomizer races are central in many of the communities. Here, speedrunning, streaming, and development all come together to form a cohesive whole.

Racing randomizers may appear straightforward: two or more people play the randomizer, and whoever completes it first is the victor. This is true in a broad sense, but there is quite a bit of nuance that is left out. Instead of each racer generating their own game instance from a randomizer, a third party will generate an instance of a specific seed and distribute copies of that instance to potential racers. This ensures that no racer is given an edge thanks to a more favorable seed. Another aspect is that communities will have certain rule sets and standards. Some communities even have multiple rulesets. The Ocarina of Time Randomizer racing community has multiple leagues to account for racing skills *(YoshiKyo and Emosoda, ootrandomizer.com).* Like many other competitive events, there will be seasons that include specific rule sets. Rulesets will change in between tournaments in order to keep things interesting. Delmarine, a community organizer for the Final Fantasy IV Free Enterprise Randomizer explains it in the following quote:

> Racing seasons get scheduled by event designers. They do two big events a year that take 2 months to run. Mid-December they start thinking about it.

The first event is a swiss tournament. The summer-fall event has a more experimental format.

Races are typically broadcast on Twitch. Often, the actual players will remain silent throughout the race, with dedicated commentators instead making observations about the race as well as making conversation with each other. Additionally, for the sake of the viewers there will be one or more trackers monitoring the race's progress. These participants will employ third party software that indicates what items a player has collected or objectives they've completed.

Fig 22: an example of a tracker for Super Mario RPG Randomizer. *The leftmost column displays the game's stars. Collecting all seven is usually the goal of the game. The next three columns are key items, which unlock areas and useful power ups for the player. The remaining columns indicate bosses which guard stars or key items. An item that is black and white indicates a player has not obtained that item or defeated that boss yet.*

Commentators and trackers are expected to make a time commitment during the race. The qualifications of trackers and commentators can vary slightly, but

generally speaking anyone is allowed to participate in the production side of races. The tournament organizers will put some rules forth for commentators. Anyone who is giving inaccurate information or being inflammatory or too bombastic may be kicked off the stream. A way to ease newer commentators into the practice is to pair them with veteran commentators who will give them space to try commentating. Demerine also points out that “potential commentators should try tracking first before applying to commentate.”

Fig 23 a screen capture of a Final Fantasy IV Free Enterprise *race stream. The design includes elements of the game UI such as font and text boxes.*

The visual layout of racing streams tend to have certain things in common. You can see many elements above in fig. Xx. The majority of the stream is dominated by the actual games being played, with the rest of the real estate being available to

information to help communicate to audience members what's going on. The racer's handles and pronouns are displayed along with their win-loss record. Beneath that are individual trackers that display their progress through the game thus far. Finally a timer is present for each player. If a player finishes, their timer will stop, displaying their completion time. Finally in this example, there are a set of numbers that correspond to the objectives that need to be completed in this race. These objectives were specified for this racing season of the *Final Fantasy IV Free Enterprise Randomizer[29]*. These objectives can be referenced in the bottom center of the screen. Finally, also in the bottom center are the handles and pronouns of the current race commentators.

It's important to understand the link between the racing scene and the development team in most randomizer communities. It is the feedback of this group in particular that drives most of the iteration on a randomizer. Since randomizers are not being sold for profit and tend to be projects taken on by their developers in their free time, they usually have no set update schedule. Changes to randomizers happen because of a desire on the part of the developers, and what gets changed tends to be driven by the requests of the racing community.

> "There is a channel in the Discord called 'Feedback and Ideas.' It is open to everyone, it is a very clear mission statement. The goal of any change to the game is to make an
> interesting and competitive experience."

D
e
m
eri
ne

One example of a suggestion posited by the racing community is a specific “Race Mode” option added into the *Wind Waker Randomizer.* This is a special form of logic more curated towards racing. It does this by making seeds slightly more predictable and therefore participants must rely more “on skill”.

The kind of community that makes up this symbiosis of racing and development can also be impacted by how welcoming the people are. It is regrettably common that many randomizer communities have struggled with things like gatekeeping and bigotry. This can be caused by important individuals within the community holding certain beliefs or pervasive attitudes that are not uncommon amongst hobbies adjacent to randomizers.

> “(*the Final Fantasy I Randomizer*) is the story of old boys in gaming. The initial community was started by a bunch of dudes. It was not properly curated.
>
> -Leggy Starscream”

These attitudes can even cause communities to splinter off and be the impetus for the development of new randomizers. After facing discrimination in the *Final Fantasy I* Randomizer community, LeggyStarScream developed *HMS Jayne,* a

randomizer for the Gameboy Advance Port of *Final Fantasy I.* This game also features post-game content that the original lacked as well as a bespoke compression algorithm for the overworld map data. Upon release, LeggyStarScream and her team chose to look towards the community curation of the *Final Fantasy IV Free Enterprise Randomizer* Rivers McCown to promote an inclusive atmosphere.

> “The most important thing about *Free Enterprise* in their opinion is to make a welcoming
> community. We’ve put a lot of people other than white dudes at the forefront. The community is extremely active. “[30]
>
> Rivers McCown

The *Final Fantasy I* randomizer is unfortunately not an isolated incident. In early 2021 a developer by the name of HalfARebel for the Super Metroid & A Link to the Past Combo randomizer expressed some inflammatory homophobic opinions in response to a submission of a gay pride link sprite to the randomizer. To this community's credit he was eventually let go, but to some the team's response was slow going.

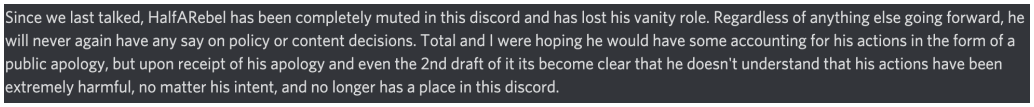


Fig 24: A public apology from the Super Metroid & Link to the Past Combo Randomizer *development team on their official Discord Channel. “Since we last talked, HalfARebel has been completely muted in this discord and has lost his vanity role. Regardless of anything else going forward, he will never again have any say on policy or content decisions. Total and I were hoping he would have some accounting for his actions in the form of a public apology, but upon receipt of his apology and even the 2nd draft of it its become clear that he doesn't understand that his actions have been extremely harmful, no matter his intent, and no longer has a place in this discord.”*

Constant Content

One benefit of randomizers is the potential to keep a game fresh for a much longer time. The likelihood that any two seeds will be identical is exceedingly small, thereby yielding a novel experience with each generated instance of a randomizer. Authorblues points out that his Super Mario World Randomizer is geared towards "people who are really familiar with the game and are a little tired of it." Longevity of play is something that is ubiquitous amongst randomizer players. Demerine has been playing Final Fantasy IV since childhood and believes that these communities are built at least in part on nostalgia. This allows for a symbiosis between content creators who are visible members of the community and generate interest for a randomizer, and the randomizer to continually develop and provide fresh experiences for the creator.

For these habitual players, there is another clear benefit of the ever refreshing content of randomizers. *Metagaming: Playing, Competing, Spectating, Cheating, Trading, Making, and Breaking Videogames* describe how speedrunners are able to monetize their hobby. "While speedrunning has not yet adopted sponsorship models and corporate funding like e-sports, a few speedrunners have started to make a modest living by coupling Twitch's partner program with donation systems borrowed from the GDQs." (*Boluk and LeMieux, 48)* However, playing the same game over and over poses the problem of burnout for both streamers and viewers alike.

Playing randomizers with harder settings is often a donation goal for viewers to contribute towards.

•DATE•	VIDEO TITLE	VIEWS	RATING%	COMMENTS	EST. EARNINGS
2018-12-22	Ocarina of Time Randomizer (max random settings, no l...	1.2M	95.9	2K	$624 - $5.0K
2019-10-23	Ocarina of Time Randomizer - Max Random Setting, Entr...	611.5K	96.7	753	$306 - $2.4K
2019-02-19	Majora's Mask Randomizer	556.9K	96.6	709	$278 - $2.2K
2020-01-13	AGDQ 2020 - Ocarina of Time 100% No Source Requiremen...	501.8K	96.6	664	$251 - $2.0K
2020-03-19	Ocarina of Time Randomizer - Max Random Settings, No ...	498.5K	96.9	644	$249 - $2.0K
2019-03-24	Ocarina of Time Randomizer (With Entrance Randomizer)	458.8K	96.3	586	$229 - $1.8K
2018-11-28	Nimpize Adventure Playthrough Part 1	360.5K	95.4	469	$180 - $1.4K
2020-10-22	Ocarina of Time Randomizer - Max Random Settings, No ...	340.2K	96.8	557	$170 - $1.4K
2020-07-28	The Missing Link (Ocarina of Time romhack) Playthrough	324.8K	96.8	590	$162 - $1.3K
2018-11-29	Ocarina of Time 100% Speedrun in 3:53:33 (With Chat)	290.0K	90.4	170	$145 - $1.2K

Fig 25 Zfg's YouTube analytics. As of May of 2021, Zfg has over 917 videos uploaded to his channel. An overwhelming majority of them are dedicated to speedrunning Ocarina of Time 100%. However, six of his ten most viewed videos are playthroughs of randomizers. On the third column we see that the views these videos have accrued. The most viewed video (another randomizer playthrough) has more than twice as many views as the next most popular video. (socialblade.com).

Many speedrunners will upload playthroughs of randomizers to their YouTube channels or dedicate entire Twitch streams to randomizers. Some speedrunners like Zfg, MajinPhil, and gymnast86 have many randomizer playthroughs among their most popular videos. Even the official Games Done Quick YouTube channel has a randomizer in their top 10 most viewed videos. One reason this content may be so popular is because of its legibility. High level speedruns can be incredibly difficult to parse for audiences that do not have in-depth knowledge of the game. Some speedruns are so thoroughly developed that they require esoteric and highly technical knowledge of the game to fully grasp. This can leave many viewers feeling

alienated when they see an *Ocarina of Time* speedrunner pull the Triforce out of thin air through manipulating memory values in the code by picking up invisible objects. Randomizers allow speedrunners to show off their expertise in a way that is more legible to viewers, since it rewards highly skilled play without incentivizing players to skip all the content with highly specific glitches.

Conclusion

Randomizers are an emerging form of User Iterative Content. They exist in a space that overlaps with ROMhacks and mods, but are not encompassed by either. Some randomizers which boot on game runtime may be considered mods, but many of them are standalone programs. While they are distinct from ROMhacks, the standalone programs tend to be built on existing knowledge bases developed by the ROMhacking community. The technical knowledge required for their development is intertwined with ROMhacking and because of this ROMhackers tend to be involved to some degree in both the development and community.

I have created a taxonomy to classify randomizers based on their potential degree of transformation from the base game. The degree of transformation does not indicate any sort of quality or success of a randomizer, but rather how divergent the experience of playing the randomizer is when compared to the game it randomizes. On one end of this spectrum we have randomization as sort of a gameplay enhancement. A randomizer that shuffles a small part of the game to provide a tweaked experience of the base game would fall on this end of the spectrum. On the other end, we have a randomizer that shuffles so much content or enough key aspects of the game that the act of playing the game becomes fundamentally different, requiring a significantly different approach to gameplay.

An outcropping of the speedrunning community that pulls particular focus in randomizer communities is randomizer racing. The basic concept of racing involves players of a randomizer trying to complete a stated objective as quickly as possible using the same seed. These events are often spectated on Twitch and involve casters. Game randomizer racing as spectacle is an outcropping of the speedrunning community to some extent, but the special role racers take as ad hoc QA and secondary designers makes this aspect unique. Players who race tend to be some of the most avid players of the randomizers and are the most vocal about suggesting fixes and features to the randomizer. This creates a unique development cycle between the randomizer creators releasing a new patch and the racers playing it, giving feedback to be implemented in the next patch.

While briefly mentioned in the section of this paper regarding *XCOM: Enemy Within*, the realm of in-game randomizers merits further study. A heavy focus in this paper has been dedicated to programs that are specifically designed to output a unique ROM image based on an unmodified ROM. Randomizers that are integrated into a game and execute on runtime are often considered mods instead of specifically a “randomizer”, and thus understanding their relationship to an established modding community could potentially be useful.

Another limiting factor of the research was the set of games I chose for my interviews. For the most part, I chose to interview developers of randomizers that I had personally played and had significant experience with. I did this to facilitate easier interviews. If I knew the game and the randomizer well, I could ask questions relevant to my research. It would be extremely challenging for one person to interview every randomizer developer, let alone meaningfully play every randomizer in a reasonable amount of time, thus I had to limit my scope. Nevertheless, future research could involve playing more randomizers and interviewing their creators to compare and contrast the information with what is presented here.

Finally, I chose to interview and focus on the most active members of randomizer communities. Developers, organizers, and avid players of randomizers who stream to Twitch only make up part of a randomizer's player base. There are many more people who enjoy playing randomizers who simply do it as a private hobby. While I do address one reason why these players engage with randomizers, it is possible there may be others that are revealed if substantial interviewing and/or surveying take place.

Randomizers are a relatively new phenomenon within the purview of User Iterative Content. While the first randomizers can be traced back to 2012 it was not until the latter half of the 2010's that they saw wider recognition amongst hobbyist

communities within the larger video game player consciousness. Over the course of my research the number of randomizers and communities surrounding them have been growing at an ever increasing rate. This phenomenon is still young but there are many ways in which it could grow and be integrated into other adjacent communities.

Works Cited

The Legend of Zelda: Ocarina of Time, [Nintendo 64] Nintendo, 1998

Amazing Ampharos, TestRunner, et all Ocarina of Time Randomizer, Version 6.0. https://ootrandomizer.com/

Authorblues, Interview, Conducted by Celeste Jewett 12th of December 2020

ZoeyZolotova. “Majora's Mask Randomizer.” *Github*, 1.12.1.0, github.com/ZoeyZolotova/mm-rando

The Legend of Zelda: Majora’s Mask, [Nintendo 64] Nintendo, 2000

Pablo Muñoz Sánchez, “Video Game Localization for Fans by Fans: The Case of Romhacking” *The Journal of Internationalization and Localization* Jan 2009, p. 168 - 185

Nightcrawler, KaioShin, et al. “About the Site” https://www.ROMhacking.net/about/

Pinchback, Anderson, et al: “Emulation as a strategy for the preservation of games: the KEEP project”, *DiGRA 2009: Breaking New Ground: Innovation in Games, Play, Practice and Theory,* 2009, Brunel University, London, United Kingdom

Pink Pajamas, Interview, Conducted by Celeste Jewett 25th of January, 2021

LagoLunatic, “DSVania Randomizer”, Github 1.4.4 https://github.com/LagoLunatic/dsvrandom

LagoLunatic, Interview, Conducted by Celeste Jewett 14th of January 2021

Pokemon FireRed, [Gameboy Advance] Gamefreak, Nintendo, 2004

Earthbound, [Super Nintendo] HAL Laboratories, Nintendo, 1995

Wasteland Ghost, “XCOM Randomized.” Nexus Mods, 4.0, https://www.nexusmods.com/xcom/mods/499?tab=description

XCOM: Enemy Within, [PC] Firaxis, 2k, 2013

Super Mario World, [Super Nintendo] Nintendo, 1990

Authorblues, kaizoman666, “Super Mario World Randomizer” 2.2
https://authorblues.github.io/smwrandomizer/

The Legend of Zelda: the Wind Waker, [Nintendo Gamecube] Nintendo, 2002

LagoLunatic “Wind Waker Randomizer” Github, 1.9.0
https://github.com/LagoLunatic/wwrando

Arsenault, Dominic: *Super Power, Spoony Bards, and Silverware: The Super Nintendo Entertainment System,* The MIT Press, 2017

“Memory Mapping - Super Nintendo Entertainment System Features Pt. 09” Youtube uploaded by Retro Game Mechanics Explained 3 September 2019
https://youtu.be/-U76YvWdnZM

Schreier, Jason “Inside the Wild New Mash-Up of *Link to the Past* and *Super Metroid”* Kotaku, 15 May 2018, G/O Media,
https://kotaku.com/inside-the-wild-new-mash-up-of-link-to-the-past-and-sup-1826049172

Tewtal, tarthoron, et al “Super Metroid Link to the Past Crossover Randomizer” 11.2 samus.link

Stochaztic “Earthbound Randomizer” v48 earthbound.app

LeggyStarscream, Interview, Conducted by Celeste Jewett 17th of March 2021

Mandelin, Clyde: “Earthbound Reshuffler”, 1.5,
https://earthboundcentral.com/reshuffler/

NESLuver: “PK Hack” http://starmen.net/pkhack

Scully-Blaker, Rainforest: *Recreating the Accident: Speedrunning as Community and Practice,* 2016, Concordia University, Master's Thesis, Spectrum Library, https://spectrum.library.concordia.ca/982159/11/Scully-Blaker_MA_F2016%20v2.pdf

Shadodraft, Sjorec, et all: *Speedrun.com,* https://speedrun.com

Boluk, Stephanie and LeMieux Patrick: *Metagaming Playing, Competing, Spectating, Cheating, Trading, Making, and Breaking Videogames,* University of Minnesota Press, 2017

Demerine, Interview conducted by Celeste Jewett, 22nd of March 2021

YoshiKyo and EmoSoda, co-hosts: "Community Patchup - Next Steps & How to Get Involved (feat. TreZc0_)", *The Gossip Stone Podcast,* January 22, 2020 https://ootrandomizer.com/podcast

McCown, Rivers and SchalaKitty: *Final Fantasy 4 Free Enterprise,* https://ff4fe.com

McCown, Rivers: Interview conducted by Celeste Jewett, 23rd of March 2021

Social Blade: ZFG's Youtube States, https://socialblade.com/youtube/user/zeldafreakglitcha accessed 17th of May, 2021