Solutions for Game Design Problems

Bogdan, Bîrseuan

West University of Timişoara,
Faculty of Mathematics and Computer Science,
Department of Computer Science in English, Timişoara, Romania

Email addresses: birseuanbogdan@yahoo.ro
(Bogdan, Bîrseuan)

Abstract

In a world where a person's attention is the most important currency, finding ways to captivate and engage is the most valuable tool. With the rise of a trend known as "gamification", understanding of game design becomes significantly more important in the creation of software, due to the increased competition for a user's time and attention. This paper focuses on problems and solutions regarding information, replayability, and accessibility in games, and software in general. Most solutions presented can be slightly altered to fit any application that deeply cares about user retention.

Keywords: *games, design, replayability, accessibility, information, variety, randomness, control, ui/ux, user retention, psychology*

1. Introduction

In a world where a person's attention is the most important currency, finding ways to captivate and engage is the most valuable tool. Any application that directly benefits from a user spending time on it, can tremendously profit from implementing game design techniques, and many have already done so, since games are made with the sole purpose of giving the user a fun time, and engaging them as much as feasible, to get them to spend as many hours/days/weeks as possible, while also attempting to not make the user feel regret or consider themselves robbed of their precious time. This trend is commonly known as "gamification". Real world applications of these game design principles can be observed, in various ways, in numerous types of software, such as applications for learning ("Duolingo"[1] for example, being one of the most predominant in this category, implementing systems such as leaderboards, a map-like structure with levels, daily rewards and streaks, quests/missions with rewards, consumables, rewarding progress, etc.) and social media (such as "Facebook"[2], "Messenger"[3], or "Instagram"[4], outright implementing games, usually on the smaller side, being either directly accessible in a section of the application, or indirectly, by sending a singular sport emoticon/emoji to a friend and clicking on the emoji, incentivising them to start playing as well, and engaging in a competition for the highest score, all to keep the user retention high), along other examples. Although this paper contains information which might seem specific to game design, most of it can be applied (sometimes directly, other times with slight alterations) to various other applications, in numerous fields.

2. Design Problems and Proposed Solutions

In this chapter, we describe some of the problems, a few possible solutions, and some related work, together with their solutions, to gaming as a whole, with the focus on the games and genres that are most affected by each problem. These issues have been identified from various games across various genres, although some of these problems might be more relevant to some genres than to others.

2.1. Information

Information, and how it is or is not presented, is vital to any game, its systems, and its mechanics. Some games require wiki pages, spreadsheets, and community tools for quality of life features (such as: "Path of Exile" or "The Binding of Issac: Rebirth" [5]), while others might have a lot of those tools built-in, even if in a slightly less accessible way (such as: [6], [7]). This topic is relevant to any and all games, as the way you deal with information and its availability can make or break the experience for a new player to a game, or to a whole genre, especially if there is lots of it.

Information, or lack thereof, is a big problem in most games nowadays, we will be referring to this as the **"lack of information problem"**. In games as vast in size as the ones that might have to worry about this problem, lack of information can become a big problem quite quickly. When a game is comprised of hundreds of items, tens of characters, and hundreds of secrets and easter-eggs, all in generally diverse environments, keeping track of information you have gathered should not rely solely on the player, yet a lot of games do just that, or lazily implement a "CODEX" (in-game encyclopedia, containing information about the world, characters, or anything that is relevant enough to the game) that is comprised of what can only be classified as an information dump. Although a CODEX in itself is not a bad idea, due to it being a system which generally "discovers" the game together with the player, game creators would often rather let the player remember most of the information themselves, generally due to some of these reasons:

1. They can have a high chance of breaking immersion/flow, depending on the quality of the implementation, and the type of person that is playing the game;
2. Difficulty of creating a customised system for concise storage of facts;
3. Deciding which things deserve a section, and how much information that section should have (too much information becomes an information dump; too little information will require the player to go outside of your game to search for what they want or need to know);
4. They have a bad reputation because they are often poorly implemented.

Not having a CODEX or similar feature results in the risk of a player making game-changing mistakes due to forgetting facts about a game's systems, which they might have already learned, but maybe forgot because of factors such as:

- Not being invested enough in the game at that point in time (or being a "hardcore" player of the game);
- Player has taken a break from the game, or had certain events take up their free time for a longer period;
- Too many things to remember in their real life (work, school, exams, etc.).

This forces the player to try and memorise whatever they can about the game (especially in the beginning, while the player has yet to find out what information is or is not important), which will lead the user to remember useless things, such as the affixes/modifiers of some "common" items, which, in games with hundreds of items, is sometimes to the detriment of bigger game systems which might be encountered less frequently than said items, yet bear a lot more value in terms of the reward for remembering information about it (for example in "Path of Exile", retaining the names and affixes of common unique items, compared to the names and affixes of "essences"). We must also keep in mind that not every user will become a hardcore player of our game to remember all the fine details and mechanics present in it, as previously mentioned, they also often take days or week long breaks from games, due to internal (game-related) or external (life-related) factors. If possible, we want to keep our systems complex, not complicated, usually achieved through careful use of intuitive systems, some which users might be familiar with from their day-to-day life (such as: red = danger, yellow = warning, green = safe; which we commonly associate with traffic lights, but that are also ingrained in humans due to the association to what we find in nature), however, that might not be feasible in every game. Some of the reasons for this might be the game including innovations in the space, or having very abstract systems. In this case, where the player is not offered a CODEX, because of one or more of the reasons stated, the only alternative is having a community wiki of the game[8], or a cheatsheet[9] (once again, usually created by the community) open at all times for reference. This creates plenty of moments where the player has an increased likelihood to think about anything other than the game, which, from our perspective as game developers, is **far** from ideal. Having the player get distracted with their thoughts might also be to *their* detriment, considering the fact that plenty of people look for a break from their real life when engaging in video games.

The problem of lack of information has plenty of possible **solutions**, each with their advantages and disadvantages:

- A display with varying amounts of information, that shows up each time the user is close to an item, or interacts with an item a certain way (has mouse over item, picks up item, etc.). This is not limited to items, they are merely offered as an example. **Positives**: ease of use/development, helps the player by giving a summary. **Negatives**: takes up screen space which can distract the user or mentally take them out of the game, might be too much to read;

- An RPG-style CODEX - generally found in story-driven games such as RPGs (Role Playing Games), but copied to many other genres due to it generally being easy to implement. This specific CODEX can be implemented in multiple ways:

  - The find-implementation is like a book with all the pages scattered throughout the world - while exploring (usually) you find pages (or said game equivalent - tablets, messages, etc.) with information. The information that you gather is in regards to their current area (generally), such as: environmental hazards, area specific enemies, relevant mechanics found in current zone, parts of the story (more or less relevant to the main narrative), etc.;
  - The discover-implementation is like a journal where the character, or their companion (artificial intelligence in futuristic games, or a helping spirit/god/fairy/etc. in games set in fantasy scenarios, friend, etc.) writes or states information that can more or less be observed by the player (*for example when killing a boss, the player likely learned some facts about it, simply by observing*), such as: narrative developments, enemy information (weak spots, "quirks" - fast/slow/strong/weak/etc., place in the world - how they interact with the world and where they fit in accordance to everything else living in the same space), specific zone details, etc. The player unlocking that information could be conveyed through something like an achievement notification;
  - The combination-implementation is comprised of both base implementations.

  Positives: removes the need for a Wiki, eases and speeds up the search for information, and ensures the information is factual. **Negatives**: can mentally take the user out of our game by giving them too much to read/process;

- An iterative/progressive/gradual CODEX where you unlock some information each time the player interacts with the world in one of the ways specified in the previous CODEX implementations - for example: each time the user picks up an item, they unlock some information about that item, the first time might be just the modifiers, the second interaction with it unlocks the item drop conditions (from a boss, on a certain floor, in a certain room, etc.), the third time might be extra information (like being part of a certain set of items, or some specific story about the item, or interesting interactions with other items/things in the world). **Positives**: removes the need for a Wiki, eases and speeds up the search for information, ensures the information is factual, reduces the chance of mentally taking the user out of the game by giving them too much to read/process. **Negatives**: if too many items get details unlocked too fast the user can be overwhelmed with the amount of context switches that they need to perform between said items and simply ignore all the valuable details;

- A combination of the aforementioned solutions. **Positives**: can mix and match for optimal user experience; **Negatives**: much longer development time, needs much more testing to ensure that the systems will work together properly.

If the systems/items/characters/etc. present in your application can be easily separated in categories, a tag system (visible or not) could prove very useful for users, in the case a search box is available to them in any CODEX implementation.

2.2. Replayability

While certain games are made with the intention of only being played once or twice, or every couple of years, once you forget about the story/experience (games like the "Mass-Effect" series[10], "Superliminal"[11], or "Outer Wilds"[12]), the contents of this section are focused on games that wish to add "replay value" to the list of what they offer to the end user. Although even games with no such intent can utilise these to great benefit, especially ones with longer completion times, or risk being left unfinished. Even if a user wishes to continue playing after they have finished the game, if they are given the exact same experience, or very close to it, the vast majority of players will stop.

Games that want to focus on replayability must take into consideration three integral pieces to any game, pieces that we will refer to as "the three pillars of replayability":

1. Variety
2. Randomness
3. Control

To ensure that most users are satisfied with the replayability of the product, these pieces must all be present in some form or another, and must be intertwined, at the very least to a minimal degree, for an optimal replay experience. As previously stated, at times, games that take longer to complete also suffer a great deal if these three pillars are not implemented in a decent manner. Although not all games with high replay value have these pieces interconnected properly, the games that do, offer a much more cohesive feel, while the games that do not, might only offer great replay value because of great, fulfilling gameplay, which, while an amazing achievement in itself, could be improved with some thought and care for this issue.

2.2.1. Variety

Variety describes the difference between elements of the same type in a game. For example, having 2 types of enemies, a boss and a normal enemy, would not qualify as enough variation to fulfill this requirement, however, having multiple types of enemies, such as normal/enhanced/bosses/mini-bosses, each with different attacks, movement patterns, and looks, would fulfill said requirement.

Lack of variety leads to repetitive gameplay, since the player will not need to switch their strategy often and/or think about ways of dealing with their current (supposed) challenge. It can come in many forms:

- Layout/Map/Room/etc. = in the case of a dungeon crawler for example (such as "Diablo IV"[15], which received many complaints about the repetitiveness of the enemies and dungeons), having a new, different dungeon, to conquer the challenge it offers every time you open the game, is the sole purpose of games in this genre, without concerns in the realm of how it is achieved. Even in the case of more story-oriented games, having rooms/houses/cities/planets([Sim], [16], [17], [18]) that differ with every encounter, offers a sense of novelty to what is essentially very similar content in a different package;

- Character = in the case of any game, having varied characters, that are more than just slight stat(data representing a certain *aspect* of a character - derived from the word "statistic", however, in the context of video games, they convey 2 different meanings, as a character can have both stats and statistics in a game) differences, and cause the player to approach an encounter with a different strategy than they normally would, simply because of what their character can or cannot do, can make a run of the game feel significantly more engaging;

- Enemy = having a small pool of enemies will, in any game, quickly become tiring and boring, since the player can use the same strategy over and over again, to deal with the same two or three moves an enemy might throw at the player, thus losing whatever sense of challenge the user had at the start of the game;
- Item = if the game has an item system, making sure that items can be diverse enough is very important. Depending on how many items a user is given throughout a run, and how important of a role a developer wishes items to play, the amount of variance from one item to another should be heavily thought upon. For example, if a developer wants player power to come in a big part from items (assuming player power is even a concept in their game), and they want a lot of those items, then a good idea would be to have a tier system for items, which might dictate rarity and how powerful they can be, or limit what they can modify. They might also use tags as a way to further differentiate between items, impose spawn limitations (such as: boss-only, first floor only, etc.), or pick a certain tag which given items have to include (such as: healing, damage, regeneration, etc.).

Having elements that are different each time you encounter them can greatly improve the appeal of the game, because users want a lot of possible experiences in one game, as it makes the perceived value much higher, in the case where they enjoy the game a lot and wish to play it again, whether immediately, or in a few weeks, months, or years. Some **solutions** for the previously discussed problems include:

- Layout/Map/Room/etc. = whether the game is 2 or 3 dimensional, using a tile system and rule-based random generation, a multitude of pre-fabricated dungeons, or a combination of both for a more fine-tuned, and generally higher-quality experience, will usually yield the best outcome while simultaneously being the simplest way of achieving variety for the space the game is presenting. Tiling for 3D environments can be done by setting arbitrary measurements of space, which fit every possible asset for that area, for each type of space (room, city, etc.), and making sure that division by that measurement results in a whole number for each relevant dimension (a developer might not care about height for example, if every asset is guaranteed to be smaller than every space it can be placed in), and if the asset does not take up the entire tile, it can be further randomly placed within that tile;
- Character = depending on the amount of importance a developer wishes character selection to have, characters can simply differ through (slight?) stat (attack speed, damage, health, etc.) modifications, mechanical changes (flight instead of walking, different attack/attack pattern), addition of mechanics (health regeneration after passage of time, pressing a predefined button throws a bomb, bumping into an entity deals damage without attacking, etc.), or a combination;
- Enemy = to increase variety when it comes to enemies, a developer must make each type of enemy as unique and different from the rest as possible. This can be achieved through enemy tiering (normal, enhanced, mini-boss, boss, etc.), creating a design theme for each different enemy within its respective tier/type, and a number of changes to their stats/modifiers, movement and/or movement pattern(s), attack and/or attack pattern(s), to fit that design theme (for example, an enemy with a shield would require a block chance stat or a reduced damage taken modifier, a slim and agile enemy could use a dodge chance or evasion modifier, etc.). Depending on the importance/rarity of an enemy type, rarer enemies should have multiple attacks/moves;
- Item = should generally be more than just simple increases and decreases: flat modifications (+5 to attack damage), percentage modifications (50% increased attack damage), increased number of usages (has an extra dash), addition of new mechanics (enters stealth when abiding by a condition), changes to core mechanic (flight instead of walking, normal attack changes to throwing damaging consumable which is now in infinite supply), etc.

2.2.2. Randomness

Randomness can be described as the lack of predictability and/or lack of a definite pattern presented by the outcome of a certain action (rolling unaltered dice, flipping a coin, etc.) when repeating it for an indefinite number of times. It usually shows up in 2 generalised ways:

- input randomness = happens before you can make a decision, for example, dungeon generation in most dungeon crawlers happens in a way where the player cannot influence its creation;
- output randomness = happens after you make a decision, for example, after choosing to attack, the attack damage might be randomised, or in a deck builder, the cards you are given are outside of your control in the moment of the fight.

To understand how randomness can be of use, one can look at the following examples, showcasing different parts that are present in most games:

- Layout/Map/Room/etc. = if a game wishes to have multiple spaces, ones where it does not bear any importance whether or not said space is identical (for example the houses in a city must remain the same once generated, but when first starting the game it can be randomly generated), randomness could help solve the monotony of seeing the same space repeatedly, or of using an identical strategy as last time, by introducing variety (as previously stated, the three pillars are/should be interconnected);
- Character = randomness in things like attack damage of a character, usually force the player to have multiple plans to deal with a situation. Being in a situation where the plan is straightforward can become boring if that is all a game gives the user, but being in a situation that only seems straightforward, yet has a decent chance to partially or completely change, while keeping the same relative level of challenge, will keep the player much more engaged in the game, and will feel much more rewarding when done, since the perceived challenge suddenly increased, even if sometimes for mere seconds, and even if the actual challenge level remained roughly the same;
- Enemy = having the player not know which enemy or enemies , and how many of them, they will encounter the next time they enter anything that is not a safe zone (an area which can spawn/contain enemies), and making them adapt and strategise to conquer their current challenge, is fun and engaging if done right. A developer, however, should also be mindful of too much challenge, as that can quickly turn from fun and engaging, to frustrating and enraging. Careful adjustments to the upper and lower limits for random generation of the strength and density of enemies in an area, is heavily advised for the aforementioned reasons;
- Item = items dropping randomly can keep the number of spikes of excitement/dopamine(brain chemical playing a role in the feeling of pleasure and motivation, and helps in the process of learning) higher each time an item is dropped, as there are generally 2-3 spikes:
    1. acknowledgement of an item dropping (this spike can be less effective in cases where there are lots of items dropping at all times, for example in games like “Path of Exile”);
    2. observing the rarity/type of the item (if such systems are implemented);
    3. seeing the actual item (assuming it offers the player value in any way).

  A game developer should be mindful that such spikes are only possible given the player perceives value in the following ways:

    1. the player perceives value in an item dropping, that is, if items are, on average, actually worth something in the game they are playing;
    2. the player perceives value in the actual item dropped, at the time it dropped.

For the aforementioned topics, **solutions** include:

- Layout/Map/Room/etc. = for all such intents and purposes, a combination of the following could provide a big boost to the replayability of a game, by using randomness in combination with these generation methods: tile-based layout, grid-based layout, rule-based layout, noise generation (3D, Perlin, Voronoi, etc. - or a combination of these), weighted outcomes, biome generation, etc.;
- Character = randomness relative to a character can show up in many different manners, such as: attack damage, chance of boosts on certain events, stat randomisation on certain events, random selection of a stat to increase on level-up, etc.;
- Enemy = plenty of diverse approaches can be taken to enhance the enemy variety through the use of randomness, including: slight stat randomisation (with upper and lower limits), picking which enemies to fight in a situation, choosing which attacks/movement patters each enemy uses from a valid pool, etc.;
- Item = some solutions for accomplishing a diverse item pool are adding randomness to: picking item modifications from a mod(modification) pool, selecting a mods number value within a range, having tiers for mods and selecting one, having item/mod rarity/tags which are selected. In cases where the effectiveness of the spike received from an item dropping is lowered due to a large number of items dropping, the effectiveness can be increased through the means of an item filter, which only shows items you *might*, at the very least, be interested in.

2.2.3. Control

Control describes the idea of giving the player the liberty to choose what and/or how to use something that they are presented with in a game, from a decent pool of options, without the concern of whether the developer has thought about all approaches, and ideally without imposing a very limited number of strategies.

The purpose of control is for the player to get creative with their plans, by using the space and mechanics offered by the game, so they can feel prepared for whatever challenge the game could make them face. In some online communities this might be referred to as a "skill ceiling", where the higher the "ceiling", the more (and in some cases easier) a player can showcase their mastery.

One can think about the last time they have lost control, maybe they were on their bicycle going downhill as a kid for the very first time, and went too fast. They maybe recovered, or maybe have not. Any person remembers the feeling. The more dangerous the action, the more terror they feel before the conclusion. This is something that users can go through, and must be taken into consideration. A developer generally does not want their users to feel terrified or scared, unless the game they are creating is part of the thriller or horror genre, of if they are crafting a more hardcore experience, where such occurrences are to be expected, such as the "Dark Souls" series[14], or "Elden Ring"[13]. However, even in such genres, loss of control must be handled with care. If a user loses control too often, they will come to expect it, and its effectiveness as a tool for evoking feelings of fear or terror will become diminished, therefore it should be used sparingly. Although in the case of a non thriller or horror game these emotions could seem out of reach, in certain scenarios a player can go through such feelings. For example, depending on a player's investment level in the game, if they are in a challenging situation, where they can lose a lot of progress, or something they have spent a lot of time to acquire, due to a small mistake, such feelings might be provoked, along feelings of frustration and anger, should the situation go astray. If this is not part of the "promised"/expected experience, a player who is not very invested might lose interest or quit, and a high-investment player might be less likely to recommend the game, if situations such as these are not a rare occurrence.

To achieve control, a developer must present multiple options to the user, without overwhelming them, as that can cause decision/analysis paralysis, and without being too sparse, as that can lessen the feeling of being in control. A good range is usually 2-4 options, depending on the situation. That can be done through outright choosing between options in a run, or the character/map the user is going to be playing, choosing a starting item/bonus/ability/etc., or settings for how the game plays out, such as difficulty settings (can also offer options for fine-tuning the difficulty: enemy accuracy, enemy health, enemy damage, etc.) (in some cases it might be worth *slightly* increasing the reward together with the difficulty, but a careful approach is needed, as in some cases it might lead to a sense of fear of missing out), item drop chances/places/etc., enemy quantity, experience gained (or if character levels are a mechanic at all, assuming this feature exists), space layout, etc. A way to present these options might be through a sliding array of images, or a container holding all the choices in a grid-like pattern.

2.3. Accessibility

Accessibility issues are often overlooked by developers, yet they can be some of the most problematic ones. Accessibility does not only refer to the inability of a user to interact with the product, but is also concerned with its ease of use. These problems can often arise from a lack of user understanding. Developers oftentimes forget to take a moment to try and empathise with how a user might be feeling when completing a certain task. These difficulties can be split in a few general subjects, some of them being:

- Relevant information must be easily found in the application = unless the information the user is looking for is about things such as the inner workings of the application, all information must be clearly communicated to the user, in some way/place in that piece of software;
- Easier/more intuitive user interface controls = moving from the mouse to the arrow keys to the "WASD" keys(or whatever the game control keys are) (or any other order of these inputs), is not only annoying to end users, but also redundant. The necessity of such movements should be limited as much as possible, if removal is unfeasible;
- Abiding by standards / intuitive design = conforming to the various set standards of human-software interaction is *very* encouraged. Making sure that the created software abides to predetermined rules for controls eliminates the possibility of frustrations related to the choice of controls, for example, basic game movement not being changed through the use of the arrow or "WASD" keys, and instead being modified with arbitrary keys, when this move of the controls does not affect other parts of the gameplay. Although certain games can "play" with a user's expectations as part of the experience, one should be mindful of the possibility of frustrations arising due to the created inability of proper use, in instances such as user interface control;
- Less convoluted user interface = a user interface that scales on both axes unevenly, that is, one which has many buttons on the vertical and horizontal axes, can become a source of frustration for users, or might make them feel overwhelmed, in the case that the experience is not intuitive enough, or presents limitations in certain areas, which are unknown to the player, especially due to an improper showcase of the interface (one which does not account for its limitations, if any), can be considered convoluted.

A few approaches for the previously defined topics are:

- Relevant information must be easily found in the application = preferably done through intuitive design, but for more complex topics, text can still work very well, if written in such a manner that most people can easily understand the topic. This text can either be presented through "pop-ups", information bubbles, a manual-type section, similar to the previously discussed CODEX, or other means;
- Easier user interface controls = the necessary buttons should, as much as possible, be within the reach of a single hand, barring the need for excessive motion. Controller support should be added, for users favouring that input device, and for people with disabilities who can remap the controller buttons to whatever special input device they require;
- Abiding by standards / intuitive design = a good start for intuitive design is by abiding by the already set standards, by, for example, making sure that all keys (such as: the function keys, "Home", "End", "Page Up", "Page Down", etc.) work as previous pieces of software have conditioned the user to expect them to work;

- Less convoluted user interface = making sure that elements on the screen have room to “breathe”, as having them tightly packed together not only looks bad, but usually means a developer has too many of them on one screen, and is forced into this position. Proper separation of elements can be achieved by splitting them into different selectable sections and/or scenes/screens.

3. Conclusions

The aim of this paper was to showcase various problems across software, with a focus on video games, and discuss viable solutions, all to improve user experience and satisfaction, leading to a better end product. The specific problems were:

- Information and how introducing various sources of in-app help for understanding its systems or any other relevant information, leads to fewer frustrations among users;
- Variety and how making similar content feel different enough in comparison to each other, makes the user less likely to find the game boring quickly;
- Randomness and how its introduction in systems creates a more unique experience from the same starting elements, leading to less likelihood of a user finding a situation they have already been in as repetitive;
- Control and how mastery of giving and taking control away can improve the user experience, and how its mishandling can lead to a user having negative feelings towards the piece of software;
- Accessibility and how providing an intuitive experience and giving users easy ways to interact with software ends up causing less unfavourable feelings (such as anger or frustration, since software is often expected to run perfectly) towards it.

Given all the problems and solutions discussed, and the vast situations where they can be applied, one can conclude that a good understanding of game design is paramount to making good games, and good software, in any area where user retention is a metric bearing any amount of importance.

Future work could include a deeper dive into some of the issues discussed, or research the way all the subjects interact and affect each other.

Bibliografie

[1] Duolingo. (2011). Duolingo. Accessed: May 15, 2024. [Mobile Application/Website]. Available: `https://www.duolingo.com/`

[2] Facebook. (2004). Meta. Accessed: May 15, 2024. [Mobile Application/Website]. Available: `https://www.facebook.com/`

[3] Messenger. (2008). Meta. Accessed: May 15, 2024. [Mobile Application/Website]. Available: `https://www.duolingo.com/`

[4] Instagram. (2010). Meta. Accessed: May 15, 2024. [Mobile Application/Website]. Available: `https://www.duolingo.com/`

[5] The Binding of Isaac: Rebirth. (2014). Edmund McMillen. Accessed: Apr. 21, 2024. [Video Game]. Available: `https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/`

[6] Vampire Survivors. (2020). Poncle. Accessed: Apr. 21, 2024. [Video Game]. Available: `https://store.steampowered.com/app/1794680/Vampire_Survivors/`

[7] Hades. (2020). Supergiant Games. Accessed: Apr. 21, 2024. [Video Game]. Available: `https://store.steampowered.com/app/1145360/Hades/`

[8] The Binding of Isaac: Rebirth Wiki. (2014). Fandom. Accessed: May 15, 2024. [Website]. Available: https://bindingofisaacrebirth.fandom.com/wiki/Binding_of_Isaac:_Rebirth_Wiki

[9] The Binding of Isaac: Rebirth Cheatsheet. (2021). Tboi. Accessed: May 15, 2024. [Website]. Available: https://tboi.com/rebirth

[10] Mass Effect series. (2007). Electronic Arts. Accessed: Apr. 21, 2024. [Video Game]. Available: https://store.steampowered.com/app/1328670/Mass_Effect_Legendary_Edition/

[11] Superliminal. (2019). Pillow Castle Games. Accessed: Apr. 21, 2024. [Video Game]. Available: https://store.steampowered.com/app/1049410/Superliminal/

[12] Outer Wilds. (2019). Annapurna Interactive. Accessed: May 8, 2024. [Video Game]. Available: https://store.steampowered.com/app/753640/Outer_Wilds/

[13] Elden Ring. (2022). FromSoftware. Accessed: May 8, 2024. [Video Game]. Available: https://store.steampowered.com/app/1245620/ELDEN_RING/

[14] Dark Souls series. (2011). FromSoftware. Accessed: May 8, 2024. [Video Game]. Available: https://store.steampowered.com/app/374320/DARK_SOULS_III/

[15] Diablo IV. (2023). Blizzard Entertainment. Accessed: May 8, 2024. [Video Game]. Available: https://diablo4.blizzard.com/en-us/

[Sim] The Sims series. (2000). Electronic Arts. Accessed: May 8, 2024. [Video Game]. Available: https://www.ea.com/games/the-sims

[16] Warframe. (2013). Digital Extremes. Accessed: May 8, 2024. [Video Game]. Available: https://store.steampowered.com/app/230410/Warframe/

[17] No Man's Sky. (2016). Hello Games. Accessed: May 8, 2024. [Video Game]. Available: https://store.steampowered.com/app/275850/No_Mans_Sky/

[18] Starfield. (2023). Bethesda Softworks. Accessed: May 8, 2024. [Video Game]. Available: https://store.steampowered.com/app/1716740/Starfield/