

Designing and Evaluating Player Learning in Human Computation Games

Alex Cho Snyder

a senior thesis submitted in partial fulfillment of
the requirements for the degree of

**Bachelor of Science
With College Honors**

Computer Science and Engineering
University of Washington

March 31, 2009

Abstract

We present a new approach to the design of effective training systems for participants in human computation projects. When the computation involved is not only too complex for computers to solve, but too specialized even for an untrained human to handle, there must be an effective and enjoyable way for would-be participants to learn the skills necessary to contribute meaningful results to the project. Our work on the protein structure prediction game Foldit indicates that player learning is most strongly affected by a small set of key design components, whose effect on player understanding and ability can be evaluated through both formal and informal data collection.

Contents

1 Introduction	3
2 Related Work	3
3 Problem	4
4 Conceptual Understanding	5
4.1 Understanding Goals	5
4.1.1 Designing for Feedback	6
4.1.2 Evaluating with Playtests	6
4.2 Conceptualizing Strategies	7
4.2.1 Designing for Progression.	7
4.2.2 Evaluating with Statistics	8
5 Technical Ability	8
5.1 Recognizing Abilities	8
5.1.1 Designing for Usability	8
5.1.2 Evaluating with Playtests	9
5.2 Applying Techniques	9
5.2.1 Designing for Interaction	10
5.2.2 Evaluating with Statistics	10
6 Results	11
7 Conclusion	17
8 References	18

1 Introduction

One new and barely explored area of computer science is the development of games that harness human brainpower to solve problems that are too complex for computers to handle. While some of these games, such as the Google Image Labeler, can be played without any special knowledge, the biochemistry game Foldit requires players to master a highly unique and specialized skill set - protein folding - before their playtime can be of any use to scientific research. Any game that expects to make use of unpaid human problem-solving ability within such a specialized domain must find a way to reliably turn new players into capable experts, and let them have fun while learning, or else no one will want to play.

Though previous human computation projects have succeeded in using game design to motivate the participation of thousands of volunteers and have labeled millions of images, none except Foldit have had the challenge of teaching an entirely new domain as complex as that of protein structure prediction. As a result, there has been no systematic attempt to identify the design principles that could make such teaching both effective and enjoyable for the participants who would like to contribute to the computational task.

Until now.

In this paper we identify four game design components that have the greatest effect on a player's conceptual understanding and technical ability: feedback, progression, usability, and interaction. We also describe several methods for evaluating the effectiveness of these components in order to guide and inform the design process. The utility of this approach is demonstrated by both the mistakes and successes of our work in improving the introductory levels of Foldit.

2 Related Work

The ESP Game, created by Luis von Ahn in 2003, was the first human computation project to successfully disguise itself as a fun game and thereby attract a large and stable population of participants [13]. Now known as the Google Image Labeler, this game has players assign useful labels to images on the web as a side effect of a simple two-player guessing game. While assigning labels to images is a trivial task for most humans, it is nearly impossible for computers. The ESP Game's pioneering accomplishment is in making this trivial task enjoyable and automatically harvesting the results, which has led to the labeling of over 50 million images [12].

Luis von Ahn has proposed a general process for turning a given computation problem into a game, which he has applied to the creation of several similar projects such as Peekaboom, Phetch, and Verbosity [12]. However, for more complex domains, such as protein structure prediction, creating the game is only half the story. If the task is nontrivial even for humans, as is the case with Foldit, then the other side of the story is in teaching people how to play at all.

In von Ahn's approach, game design is used to improve the enjoyability of these basic computation tasks. But there is another area of game design that deals with learning to accomplish these arbitrarily complex tasks in the first place. Many videogame theorists, including Raph Koster in his book *A Theory of Fun for Game Design*, have argued that the fun of games is fundamentally about the brain's addiction to solving patterns [8]. People like figuring things out. This applies to learning new patterns and domains just as well as it does to solving simple puzzles like those presented by the ESP Game.

Daniel Cook, game designer and author of one of the most widely read game design blogs on the web, has developed a conceptual framework for analyzing the learning in games, which forms the theoretical basis of our approach. In his article "The Chemistry of Game Design", he uses the concept of a "skill atom" to describe the process of learning a new skill, and describes the process of mastering a game as a progression through a network of skill atoms, a "skill chain" [4]. Enjoyment is defined as the result of mastering a given skill atom. "When you learn something new, when you understand it so fully you can use that knowledge to manipulate your environment for the better, you experience joy." [4]

While the skill chain model is obviously simplistic, it provides a useful framework for evaluating the learning in games. Any game that is found to fail by the skill chain model is unlikely to provide an effective learning experience in practice. With this in mind, we describe the four design components that are most crucial in facilitating game-based learning according to the skill chain model. To help put this theory into practice, we suggest a number of ways to collect data on each component's effectiveness and how to use this to inform the design process.

3 Problem

Foldit is a human computation game where players are presented with a virtual protein molecule and are asked to fold it into the shape with the lowest energy cost [6]. This is a very difficult computational task that has previously been approached through distributed computing projects such as Rosetta@home [11] and Folding@home [5]. But even with the combined power of hundreds of thousands of home computers, there still exist problems that such automated approaches fail to solve [7]. The idea with Foldit is that human ingenuity and intuition could succeed where raw computational power has failed.

Like the ESP Game, Foldit is presented as a game to motivate people to participate. Players compete to get the highest score on each protein, where score reflects the energy cost of the folded protein such that the more accurate solutions score higher. Players can also collaborate with each other by forming teams and sharing solutions and hints in the hopes of earning a higher score than competing teams.

The problem is that the rules of protein folding and the tools available to Foldit players are both complicated enough that without prior training, anyone given a typical protein to solve would have no idea what to do, or how to do it. Even a biochemist who studies proteins for a living would hardly be any better prepared to play Foldit [7].

When first introduced, Foldit had no way of teaching beginners other than a short tutorial video prepared by the game's creators. Then a sequence of introductory puzzles was added to the game. These puzzles were simpler than the big proteins given out for competition, usually requiring just one or two moves to solve. Instead of competing to get the highest score on each of these introductory puzzles, a player would only have to get past a certain score to progress to the next puzzle. This provided some assurance that the player had mastered the skills required by the previous puzzles before moving on to the next challenge.

While this sounds good in theory, the approach has its shortcomings. Even if a puzzle is designed to require the use of some new skill or concept, there is no guarantee that the player will ever learn the required skill in the first place. It may be possible to guarantee that a player who has already completed all the introductory puzzles is prepared to begin playing the real game, but there is no assurance that a significant percentage of players will actually make it through the entire sequence. And the more hints given to players to 'help' them complete each puzzle, the less assurance there is that they have actually learned the skills being tested, rather than simply having followed instructions without understanding.

The purpose of such an introductory sequence is to ensure that each player gains a certain conceptual understanding of the problem, along with the technical ability to carry out a solution within the game. In order to design an introductory sequence that successfully accomplishes this goal, we'd like to pinpoint the design considerations that are most relevant to producing conceptual understanding and technical ability, and define methods for evaluating their effect.

4 Conceptual Understanding

What we want to evaluate is how well a player has achieved mastery of a game's problem domain. To make this easier we distinguish between conceptual understanding and technical ability and consider each separately. In reality, the two are inextricably linked because, generally speaking, learning in games is learning by doing. A player learns a concept through exercising abilities in the game and observing the results. But as a basis for thinking and communicating about a game's design, or measuring its effectiveness, the distinction remains useful.

Conceptual understanding, in the context of a human computation game, refers to how well a player understands what characterizes a good solution, and how well that player can invent specific solutions that optimize such characteristics.

4.1 Understanding Goals

The first part of conceptual understanding is understanding the goals of the game. Every game has a goal. Most games have many goals. In the context of a human computation game, this means knowing what characterizes a good solution. When players understand their goals, they can choose to perform actions that will bring them closer to their goals, and avoid actions that will hinder their progress. Without a clear understanding of a game's goals, players are liable to become bored or frustrated.

4.1.1 Designing for Feedback

Feedback is the design component that has the most impact on a player's understanding of a game's goals. Simply put, feedback is how a game responds to player action, how a game speaks to the player. Whenever a player performs an action, the game immediately communicates the result of that action back to the player. That's feedback. And the most important information that can be communicated to a player through feedback is whether the action moved the player closer to a goal, or further away.

This is essential because games teach through exploratory learning [2]. The player is given a goal and a set of actions. Instead of being told how to reach the goal, the player is expected to experiment, to try out different actions in different ways. Then for each action, the game gives feedback to the player, if the goal is now closer or further, if the action was good or bad. Through sampling a variety of actions in a variety of contexts and observing the feedback that results, the player builds a mental model of how the game works, how the goal may be reached. And the player experiences satisfaction and delight when this mental model is confirmed by the game's feedback, especially when that feedback announces that the goal has been reached. Without appropriate and informative feedback, players are denied the opportunity to figure things out on their own, the learning is much less effective and the fun disappears.

In Foldit, there are several independent goals that often come into conflict. A single move might be beneficial from the point of view of one goal, but harmful from the point of view of another. Successful play requires balancing between these goals for the best outcome overall. Feedback in Foldit originally consisted of a single score to show progress toward all goals simultaneously. This prevented players from seeing which goals were affected by any given action, instead seeing only the aggregate effect on all goals. In order to make it easier for players to distinguish between competing goals and balance between them, we added a bit of text that would appear after each score change, indicating which goals were affected and whether they were improved or worsened by the player's action.

While this certainly represents some improvement, the information content of the feedback remains much lower than that of the action it attempts to guide. Foldit players must manipulate a protein in three dimensions, and to guide their action appropriately we would like to provide feedback in three dimensions as well. For each major goal in Foldit there exists a way to pinpoint the location of problem areas and display these to the player. However, few of these visualizations were actually utilized in the original introductory puzzles. What we found is that players would tend to give much higher priority to the few goals that were represented by these local problem indicators, and would often become confused when expected to optimize their solution toward a goal not associated with such spatial feedback. In order to encourage a more balanced understanding of the goals involved, we modified one of the unused visualizations such that it could be inserted into the existing puzzles easily.

4.1.2 Evaluating with Playtests

We use playtests to evaluate the effectiveness of feedback on player understanding. For each set of changes we'd like to test, we find two or three people who have never played Foldit

before and observe each of them playing the game. Evaluating feedback requires looking very closely at specific interactions and reactions between a player and the game, so we find that such informal methods end up being the most useful. The purpose is not to get statistically valid data, but to see where feedback works and where it doesn't, and to get a feel for why and how players get stuck or succeed. After a couple playtest sessions, we invariably end up with plenty of problems to fix, and more often than not a number of good ideas for how to fix them.

In accordance with "Ron's Rules for Playtesting" [1], we note that it is essential to use only new players to evaluate player learning, as anyone who has learned to play once will not be able to learn it again. We also avoid making comments or giving hints to the player, to ensure that we test the game's effectiveness rather than our own teaching ability. The only exception is that when the player is hopelessly stuck, we may try to test whether a certain change in the feedback would help lead the player back onto the right track, and describe the hypothetical new feedback to see if the player can get going again based on that one change. We also find it helpful to encourage the player to "think aloud" and verbalize thoughts, questions, and in particular, reactions to the feedback encountered in the game.

4.2 Conceptualizing Solutions

The next part of conceptual understanding is the ability to conceptualize a specific solution that satisfies the goals of the game. This solution could be a particular configuration for a protein molecule, or a battle plan in a war game. This is problem solving. This is the ability that makes humans so useful for difficult computational tasks. While a computer program may be able to 'understand' in some sense the goals of a particular task, or the criteria for an optimal solution, for complex problems there is often no efficient algorithmic way to generate solutions that satisfy these criteria. Human participants are useful to the extent that they develop this ability.

4.2.1 Designing for Progression

Progression is the most important design consideration when it comes to developing a player's problem-solving ability. Progression deals with the way new skills and concepts are introduced and how the player learns based on previous knowledge. A player must become competent with a large number of skills and concepts before achieving mastery. Every new player starts out with few basic concepts and through the course of playing the game, develops a nuanced understanding of the problem domain. Progression is what facilitates this process.

Games that facilitate effective learning experiences for their players share several important characteristics [3]. First of all, mastery of such games involves a few complex skills that are decomposable into many simple skills and concepts that are more easily acquired. These simple skills are not separate and unrelated but rather combine to make up more complex skills, which themselves can be combined into yet more complex skills, until mastery is attained. There also exists, for effective games, a smooth sequence for learning all the relevant skills, starting with the most simple and building up to the most complex. Simple skills are reused in the service of more complex skills, and fun comes from the process of finding ways to use existing skills and knowledge for new purposes or problems. Progression ensures that this process remains smooth and enjoyable.

In Foldit, as with most games, progression is made manageable by dividing up the learning experience into a sequence of introductory puzzles. Each puzzle is a scenario in which the player may exercise some new skill, and when the game determines that the player has demonstrated sufficient competence with this new skill or understanding of this new concept, the next puzzle is made available. These gated scenarios are often referred to as "levels" and even when this pattern is not explicitly used in a game, there is almost always some purposeful engineering of the game environment to encourage practice of particular skills or familiarizing with a certain concept.

4.2.2 Evaluating with Statistics

We record how many new players complete each introductory puzzle and use these statistics to evaluate progression in Foldit. What we end up with is a graph for each day showing the gradual attrition of players, starting strong and dwindling toward the end of the introductory sequence as players become bored or frustrated and stop playing. What we hope to see is a shallow and smooth decline, with the majority of new players making it to the end of the sequence. A large number of factors must contribute to this basic decline, including how long the puzzles take to play, and how engaging the game is overall. What we are concerned with in terms of evaluating progression, however, are the steep slopes between puzzles where many players quit. Such decreases often indicate puzzles for which players are insufficiently prepared, whether because the challenge itself is too difficult or the preceding puzzles fail to teach the required concepts. Once problem areas are discovered, we turn to playtests to discover the specific ways that players get stuck and how the progression might be improved.

5 Technical Ability

Technical ability in a human computation game refers to how well a player can make use of the tools available to turn an imagined solution into an actual one in the game. This tends to be a simpler problem than that of conceptual understanding, but remains an important consideration in all games.

5.1 Recognizing Abilities

The first part of technical ability is recognizing what actions are available. For simple games this may be trivial, but for complex games it can be quite difficult, especially as new features are added to an evolving project. Any player who does not understand how to use the basic abilities of a game will end up frustrated when pushed toward a goal that requires the use of such abilities.

5.1.1 Designing for Usability

Usability is the design consideration with the most impact on how well a player will recognize the abilities that are available in any given situation. This generally translates into optimizing the spatial and temporal organization of a game's visual display for clarity, and making control schemes easy to use and easy to understand. The key concern for usability as it pertains to technical ability is to make sure players know what they can do, and how to do it.

Usability is a broad and well studied topic outside of games, even outside of software, and it is the subject of many books, such as the web usability classic *Don't Make Me Think!* by Steve Krug [9]. One important concept for usability is visual hierarchy, the use of graphic design techniques to imply the relative importance of interface elements as well as the grouping of related elements. Visual hierarchy is what directs the player's flow of attention across the screen, where some elements may draw attention more strongly, or hold attention for longer, or direct attention to other elements based on factors such as size, color, or position. This is important because a new player looking at a game's interface for the first time does not see it in its entirety. Instead, a player sees the most prominent visual element and begins scanning from there, and may fail to ever examine elements assumed to be unimportant based on their visual characteristics and the feedback given by the game. Difficulty in recognizing options or accurately interpreting feedback may result if the visual hierarchy of the interface does not reflect the game's rules.

In Foldit we made two major changes for the sake of usability. One change was to center the score display on the top of the screen to make it more prominent, and in general to distribute various interface elements around the screen where they could each be seen more clearly. The second change was to move the instructions for each puzzle from a single block of text into a sequence of text bubbles that could be attached to any interface element or part of the protein. This made it easy to point to a specific button when introducing a new tool, or in the case of tools not activated by buttons, to direct the player's attention to the part of the protein that must be manipulated. The text bubbles also provided a convenient way to draw attention to a particular interface element that might otherwise be hard to find on the screen.

5.1.2 Evaluating with Playtests

We evaluate usability through playtesting, similarly to the way we evaluate feedback. Usability is another component where observing details and specific interactions is important, so again we rely on informal methods. We are interested in finding problems and possible solutions, not statistically valid data. As mentioned before, we make sure to use new players for every test and say as little as possible during the session, so as not to taint the results with prior experience or expectations. In the case of usability tests, it is not particularly effective to suggest hypothetical changes to the game in order to see how the player responds, as usability tends to depend on very specific details of the interface that cannot be conveyed through verbal description or simulated in the imagination.

5.2 Applying Techniques

The next part of technical ability is actually applying techniques, using several actions in combination or in different situations. It's one thing for a player to know what actions are available in a given situation, but the real concern is how well the player can make use of those actions to accomplish a specific task. If players are able to see what's required and conceptualize a possible solution, all that remains for them is to put their plans into action, and in the process, produce harvestable results.

5.2.1 Designing for Interaction

Interaction is the most important design component for ensuring that a player will develop techniques for accomplishing a game's goals. Progression is also important, but good progression may still fail when paired with poor interaction. If gameplay can be thought of as a conversation, interaction describes how well a game listens to the player. Strong interaction occurs when players are provided with a rich set of skills to exercise and plentiful opportunities in which to use them. As the saying goes, practice makes perfect. When players are able to use their abilities many times in a wide variety of contexts, and are encouraged to do so by the game, they are much more likely to extract general strategies and techniques from their experience.

One way to think about this is in terms of the information density of player action, how much information a player communicates through the controls of the game. As game design blogger Krystian Majewski puts it, "20 years ago, in a NES game, players used a 4-way d-pad and two buttons. That's 6 bit of information 30 times a second. That's 648.000 bits of information per hour. THIS is the kind of interactivity and freedom players are used to." [10] Of course, if a game does not make use of all this information then the benefit is lost. What is in question here is not simply how much information the player can pump into the game, but how much information the game makes use of in a meaningful way. If a particular scenario rewards skilled use of a deep input channel, this represents strong interaction, but if the scenario is set up in such a way that only a small subset of the possible input has any meaningful effect and the rest is ignored or punished, then the interaction is weak.

In Foldit, there were originally several 'puzzles' in which all that was required was to press a single button. Furthermore, attempting to experiment with any other action would often make the puzzle more or less impossible to complete without resetting it back to its original state. This obviously represents an extreme example of poor interaction, as well as poor progression, as no prior skills were involved in the completion of such trivial puzzles. In order to improve the degree of interaction involved for these particular puzzles, we redesigned each one such that a player would have to apply a previous skill to bring the protein to such a state where a single button press would complete the puzzle.

5.2.2 Evaluating with Chat

As technical ability is closely tied to progression as well as interaction, looking at attrition statistics may help point out places where players lack the technique required to complete a puzzle. However, we find that testing with live players through chat provides a more direct way to determine what players are able to accomplish. Evaluating technical ability does not require so much attention to detail as does evaluating the effects of usability or feedback.

With an in-game textual chat system set up and an easy way to share screen shots through the chat, we easily ask players to perform certain tasks and judge their ability by looking at screen shots of their result. New players who get stuck will often ask for help in the chat, and such players provide the perfect opportunity to get a feel for how well techniques are being learned at various stages in the game. Upon noticing a request for help by a new player, we ask for screen shots to help determine the issue that has the player stuck, and proceed to give verbal

instructions through the chat and examine the results through screen shots. For a game with a fairly consistent population of players, these chat sessions are much faster and easier to set up than playtests are, and provide a similar measure of player ability.

6 Results

To evaluate the overall effect of our changes to Foldit, we started recording statistics on December 2, 2008 to keep track of how many new players complete each introductory puzzle. As we mentioned earlier, looking at these statistics can provide useful information about a game's progression. But we'd also like to find out whether all this redesigning and adjusting, both of our own process and the game, has resulted in any concrete benefits to the project. These statistics help us determine whether the many changes introduced have actually increased the number of new players who make it through the training sequence.

We released five major updates to Foldit since we started collecting this data. What we would hope to see after each update is an increase in the number of players who complete each puzzle. However, as the following graphs will show, this is often not the case.

Each graph shows the average player attrition before an update and after, such that the effect of the update on player attrition is apparent. The puzzles are arranged along the horizontal axis, in the order in which they are presented in the game. The vertical axis shows the percentage of new players who complete each puzzle, starting with one hundred percent on the far left, and dwindling toward the final puzzle on the far right. The red line shows the average percentage after the update, while black is the average before the update. Here the average is calculated by taking the percentage of players for each day that the update is in effect, adding these together, and dividing by the number of days. The transparent pink and gray areas show the standard deviation after and before the update, respectively. Unconnected points indicate that a puzzle was moved, added, or deleted in the update.

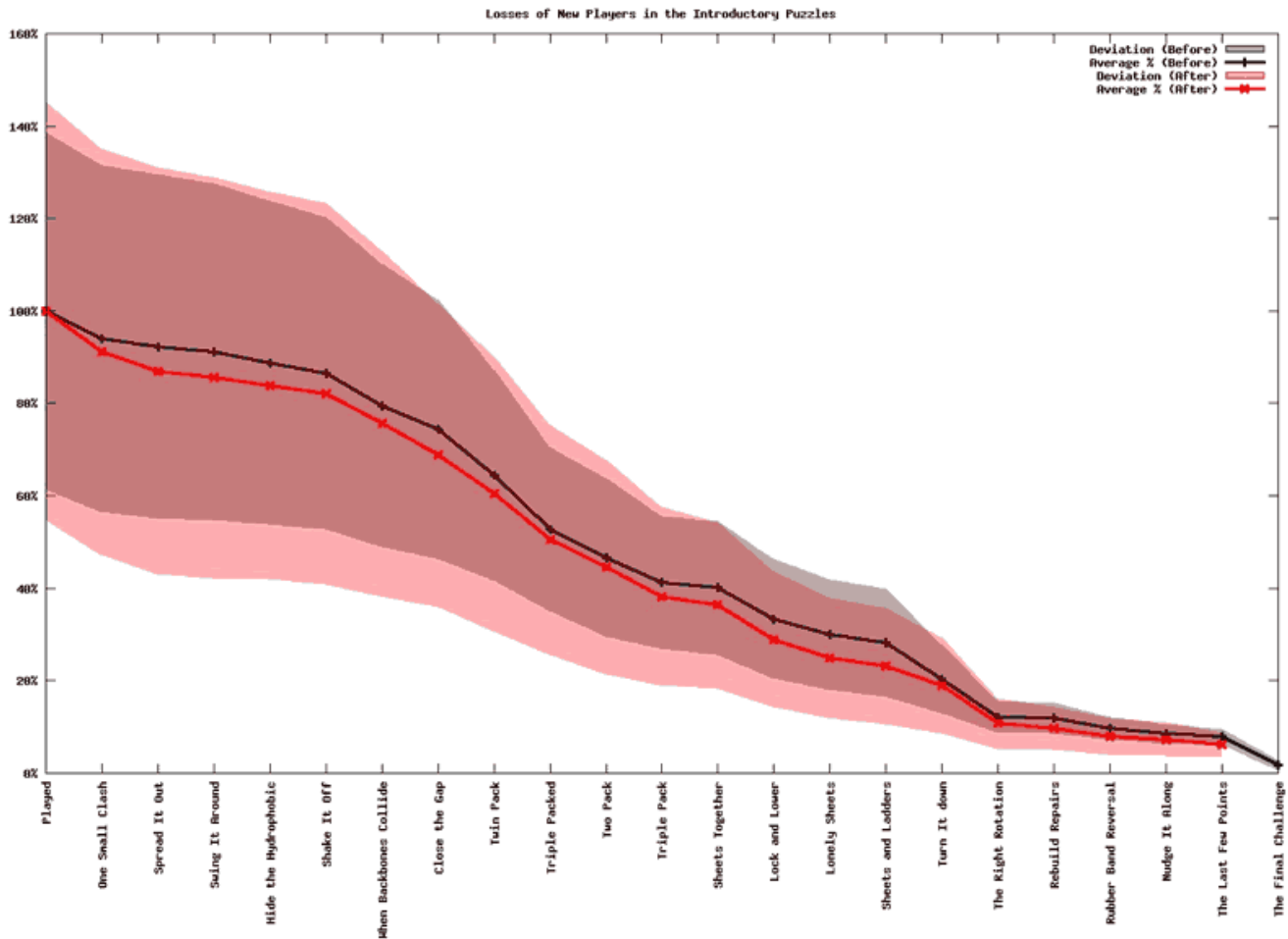


Figure 1: Player attrition before and after the December 08, 2008 update.

December 08, 2008 was the date of the first new release since the start of the statistics. The red line shows the player attrition after this update, while the black line shows the statistics collected from the previous update. Interestingly enough, the attrition seems to have gotten worse with this new update. However, the difficult last puzzle was also removed, meaning that the new update actually had a higher percentage of completion at the end. Despite this small triumph, we note that simply cutting out the difficult puzzles is hardly a sustainable approach if the real goal is effective learning rather than maximizing statistics.

Other than the removal of the last puzzle, which had been prompted by numerous complaints of its difficulty, the only other changes in this update were small adjustments to the text of the puzzles and perhaps some slight difficulty tweaks here and there. As we can see this had very little effect on player attrition, and what little effect it did have was apparently detrimental.

Perhaps one lesson that we might draw from this example is that small tweaks based on uninformed guesswork are a waste of time. At the time of this update, we did not recognize the importance of playtests in pointing out real rather than imagined problems. As a result, we failed to focus on real problems and wasted our time fixing problems that did not exist.

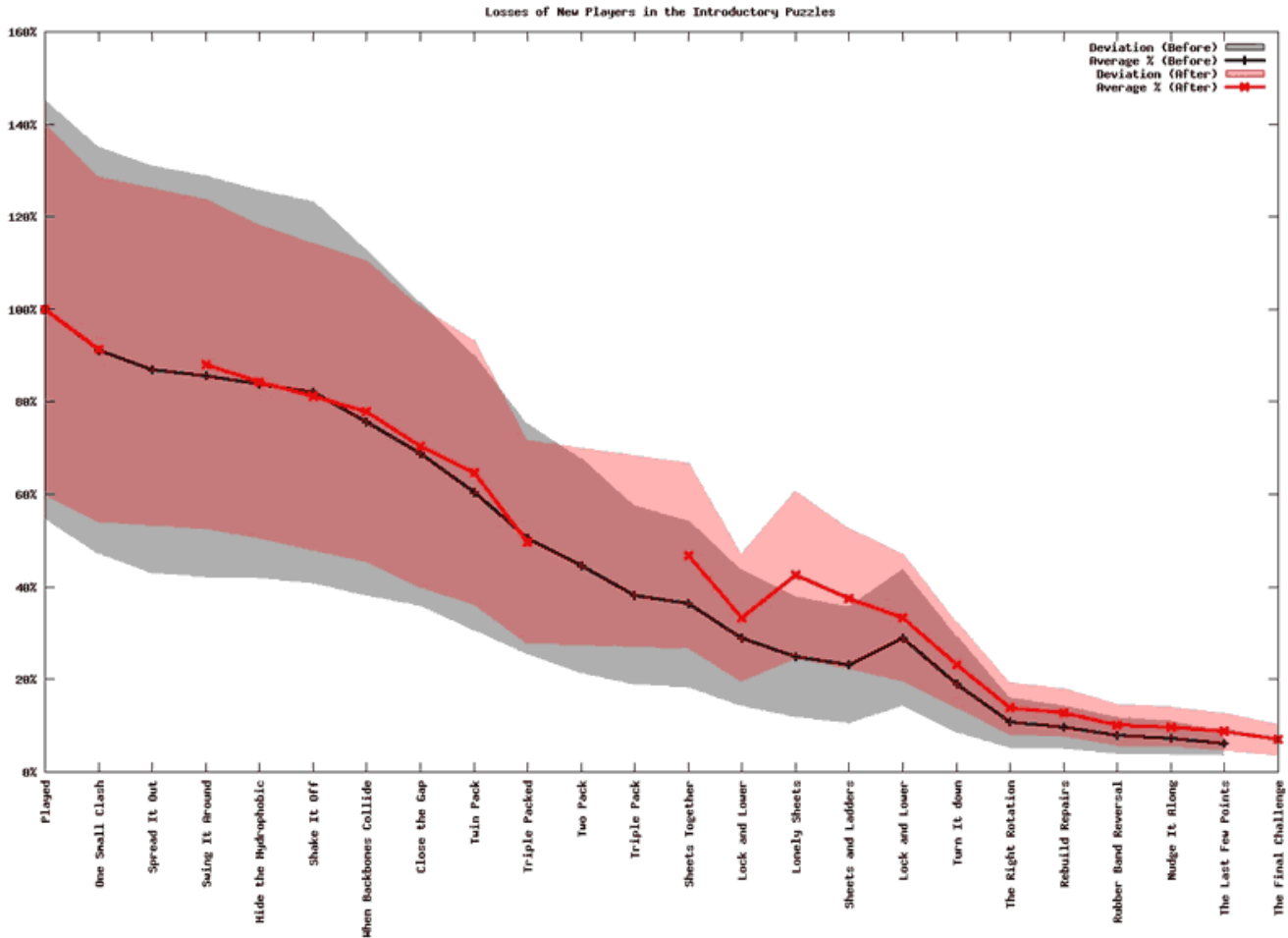


Figure 2: Player attrition before and after the December 18, 2008 update.

December 18, 2008 was the date of the second new release. The last puzzle reappeared again in this update, with its difficulty somewhat reduced. However, several other puzzles were removed on the grounds that they seemed redundant, and that perhaps by removing them players could progress to more interesting puzzles more quickly. This was another example of a guess uninformed by playtests or any other empirical method of evaluation.

Judging by these graphs, whatever immediate gain resulted all but disappeared as players encountered more difficult puzzles that might have been more manageable had they had more opportunity to practice in the 'redundant' levels. This could be a problem of interaction, where puzzles that may seem redundant are actually essential in providing sufficient opportunity for players to practice their new abilities.

Interaction may have been improved in another area however. One puzzle which previously had required a single button press was redesigned to require the use of another skill as well. But the effect of this change was hard to discern because so many more drastic changes were also introduced in the same update. A lesson we might extract from this experience is that updates with fewer changes are easier to evaluate, and should be preferred over large updates.

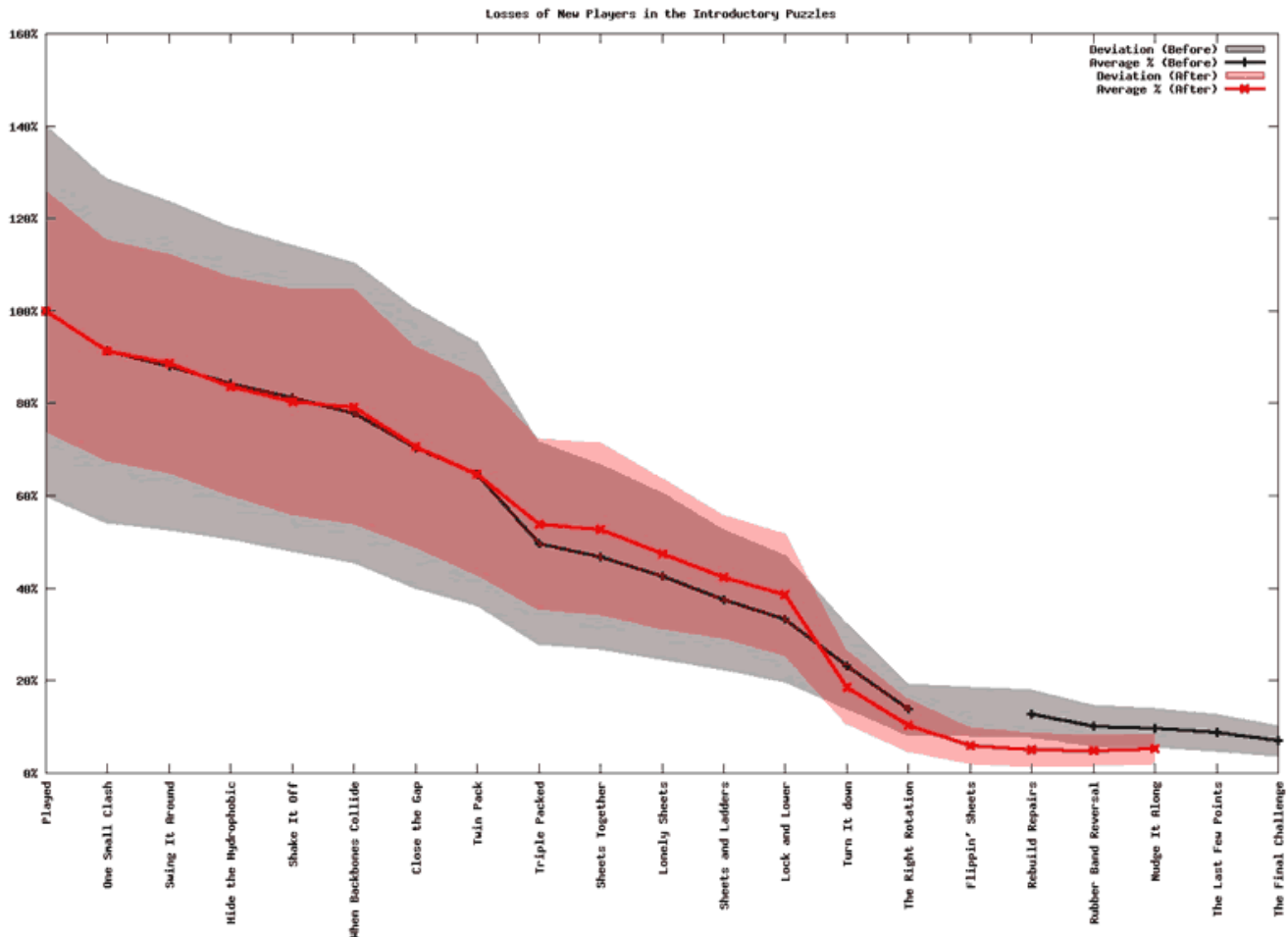


Figure 3: Player attrition before and after the February 10, 2009 update.

February 10, 2009 was the date of the third release. The most important change in this update was the addition of an entirely new puzzle toward the end of the sequence, intended to introduce a new tool that had been added to the game. This does bring up the question of whether a single puzzle can ever be sufficient to familiarize players with a new tool, though at the time of this update we gave little thought to the issue of interaction.

We also removed the last two puzzles in this update, again because they seemed redundant and didn't make effective use of skills from previous puzzles. Whether or not this was a good idea is difficult to tell, without also tracking statistics on player progress beyond the introductory sequence. This is something that has yet to be done in Foldit. Regardless, removing puzzles without following up by reanalyzing and redesigning the progression as necessary seems unlikely to be a particularly effective strategy, in the long run.

Like the others, this update had its share of small tweaks and fixes for supposed problems. One seems to have increased the percentage of players completing at least a few of the puzzles, but this success is short-lived. Upon reaching a more difficult puzzle, the line plunges down to a level below the previous update. Once again, these tweaks prove to be ineffective.

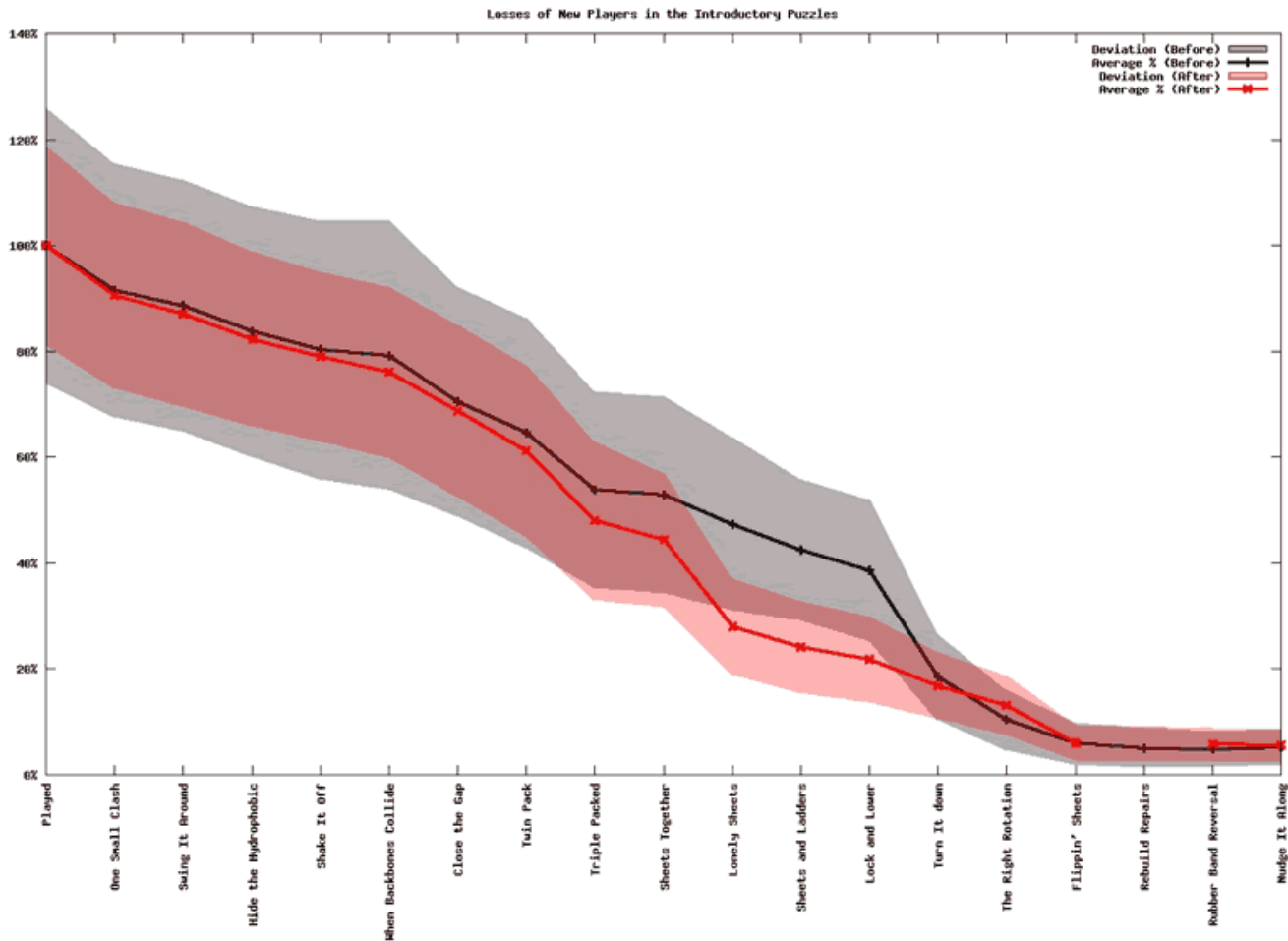


Figure 4: Player attrition before and after the February 19, 2009 update.

February 19, 2009 was the date of the fourth release. The biggest change for this update was in usability, where all the instructional text was moved into a sequence of text bubbles that would point to specific parts of the protein or the interface, instead of having all the text appear at once in one area of the screen. Though the effect on the player experience was significant, the change hardly registers in the statistics. Perhaps this lends support to the notion that these statistics are not particularly useful in evaluating usability.

Another usability-related change was showing a puzzle selection screen upon puzzle completion where players could choose which puzzle to play next instead of going directly to the next puzzle. This is another example where it would have been preferable to release a separate update for each modification. From the graph it is impossible to tell what effect, if any, the selection screen had on player attrition. We do not know whether players are more likely to quit when presented with a selection screen between puzzles, or less likely.

The one big drop in this graph has a known cause, however. After our first playtest, we decided to increase the difficulty of one puzzle such it would require mastery of a certain concept we wanted to teach. As it happened, this difficulty increase may have been premature.

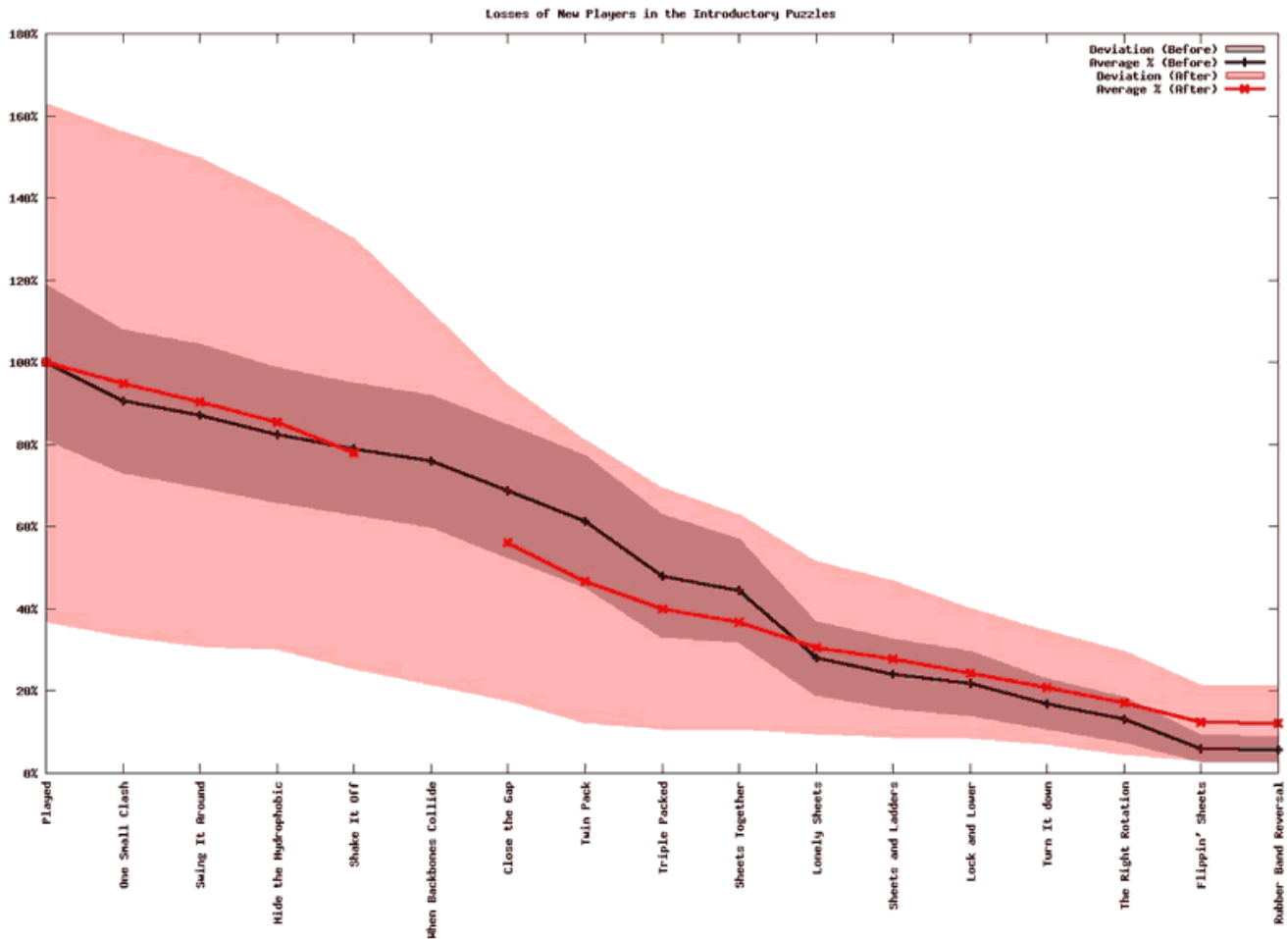


Figure 5: Player attrition before and after the March 21, 2009 update.

March 21, 2009 was the date of the fifth and latest release. This was a large update, featuring a new system of event triggers, where the puzzle could respond with text if the player got above or below a certain score, or performed a certain action, for example. We began doing playtests with this update, and the event system made it possible to respond to the tests with meaningful tweaks, to better handle the common cases where players would get stuck.

This was also the first update to introduce significant changes to feedback. Text explaining each change in score, as well as a new variety of local problem indicator was introduced. However, we did not have time in this update to redesign the puzzles based on these changes.

The one difficulty increase from the previous update was reduced here, but a new drop was inadvertently created when an early puzzle, thought to be unnecessary, was removed. Even though playtests had been done with this change, we did not find a case where this was a problem. Perhaps this serves as a useful reminder that playtests are not statistically valid, especially for evaluating progression, whose specific effect is highly variable from player to player. The best that can be done is to cautiously introduce progression-related changes, and be prepared to revert them quickly if the statistics reveal them to be mistakes.

7 Conclusion

Teaching complex skills is itself a complex skill. And the same design components that help a player master a game can help a game designer create an effective learning experience. Usability means know what changes are possible. Start with a clear picture of what needs to be taught and what software changes are feasible. Interaction means test often, in many contexts. Don't just try one option, try several variations and see which one works best. Feedback means acquire appropriate data. Collect data to evaluate and inform each important dimension of the design. Progression means build on a solid base. Get a minimal design working and gradually add pieces to it, testing and adjusting all the way.

Through our own experience in trying to improve Foldit's training sequence, we've been able to discover a few guidelines for using various evaluative techniques to effectively guide the design process. For the rough experimental stage of development, focus on playtests, as they are quick and will provide enough information to guide a rapidly evolving design. Then test these early assumptions with statistics once the game goes out to a steady pool of players. Statistics are more reliable than playtests but provide very little information. In order to maximize the guidance that can be extracted from statistics, release small updates with only a few changes at a time, such that resulting changes in statistics can be interpreted as the effects of specific changes in the design. And if the statistics indicate a new problem, run playtests to get an idea of what component of the design is failing and how to fix it. As a general rule, use informal evaluation like playtests to guide intuition about the design, and use formal statistics to guide interpretation of the informal evaluation.

The Foldit project is still in its early stages. We expect that much more will be learned along the way and many more improvements will be continue to be made to enhance the learning experience for players. With the techniques and concepts we have begun to develop, it is conceivable that any arbitrarily complex computational task may be made accessible to a wide pool of potential participants. We hope that this possibility may inspire the creation of many new and ambitious human computation projects, and that the work presented here will help to provide a solid starting point for the development of such projects.

8 References

- [1] Carmel, Ron. "Ron's Rules for Playtesting," 2D Boy. November 2007.
<http://2dboy.com/2007/11/12/rons-rules-for-playtesting/>
- [2] Cook, Daniel. "The Princess Rescuing Application: Slides," Lost Garden. October 2008.
<http://lostgarden.com/2008/10/princess-rescuing-application-slides.html>
- [3] Cook, Daniel. "What Activities Can Be Turned into Games?" Lost Garden. June 2008.
<http://lostgarden.com/2008/06/what-activities-that-can-be-turned-into.html>
- [4] Cook, Daniel. "The Chemistry of Game Design," Gamasutra. July 2007.
http://www.gamasutra.com/view/feature/1524/the_chemistry_of_game_design.php
- [5] Folding@home. <http://folding.stanford.edu/>
- [6] Foldit. <http://fold.it/>
- [7] Hickey, Hannah. "Computer game's high score could earn the Nobel Prize in medicine," University of Washington News. May 2008.
<http://uwnews.org/article.asp?articleid=41558>
- [8] Koster, Raph. *A Theory of Fun for Game Design*. Paraglyph Press, Scottsdale AZ, 2005.
- [9] Krug, Steve. *Don't Make Me Think! A Common Sense Approach to Web Usability, Second Edition*. New Riders Publishing, Berkeley CA, 2006.
- [10] Majewski, Krystian. "BORT Digest," Game Design Scrapbook. February 2009.
<http://gamedesignscrapbook.blogspot.com/2009/02/bort-digest.html>
- [11] Rosetta@home. <http://boinc.bakerlab.org/rosetta/>
- [12] Von Ahn, Luis, and Laura Dabbish. "General Techniques for Designing Games with a Purpose," *Communications of the ACM*, p.58-67, August 2008.
- [13] Von Ahn, Luis, and Laura Dabbish. "Labeling Images with a Computer Game," *Proceedings of the SIGCHI conference on Human Factors in computing systems*, p.319-326, April 24-29, 2004, Vienna, Austria.