We Do Not Understand The Software We Have Written

Simon Parent

My Ninth CGL Tech Talk
Thursday 2013 September 05

This version of the slides is accompanied by notes which appear here in the bottom half of each page. This is an experiment to see if a tech talk can also be useful after it is presented.

Introduction

- we write software without understanding what we have written
- completed software as a black box: the abstractions cannot be reconstructed
- gap between the software we write and the software we thought we wrote
- more than just the prevalence of bugs

Even a moderately large piece of software is a man-made artifact of incredible complexity. Our ability to write the software exceeds our ability to understand what is written, which is why debugging is so difficult: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." –Brian Kernighan

Consider studying the behaviour of a particular program, treating it as a black box. This resembles our attempts to understand the universe we live in, but understanding software is easier since we can measure anything with perfect accuracy.

The programmers who wrote the software have certain "laws of physics" in mind, that is, they have a **model** of the software. However, these models simply do not apply to the actual software which was written.

Of course, this is just another way of saying that bugs will always exist However, this issue runs even deeper than our lack of perfection. In extreme cases we even appear inconsistent: we write software with the intention of it satisfying property X, but then write it in a way which guarantees that property X will fail to hold.

Introduction

- software implementing video games has many good examples of this disconnect
- video game universes are usually studied by the players as black boxes
- video games have the advantage that the creator chose the rules of the game universe deliberately
- or so we would think

In this talk I will discuss examples of this phenomenon coming exclusively from video games. Video games often involve a universe obeying fictional rules which the players try to understand. The players are usually confined to studying this world as an inhabitant of it. In particular, they do not know the details of how the software was implemented.

However, these details creep into the game mechanics, as we will see. So, even those who know nothing of programming or Computer Science naturally become aware of implementation details in the game software.

Example 1: Rock Band Vocals

- following the notes fills the meter
- the goal is to fill the meter by the end of each "phrase"
- doing so maintains a combo which affects a score multiplier
- after the song is finished, an accuracy statistic is also given

Rock Band scores the player's singing based on a pitch detection algorithm: the arrow on the left shows the detected pitch. There is a meter which fills faster the closer the player is to the correct pitch, which is indicated in green on a track which scrolls to the left during game play.

If the meter is full by the end of a "phrase", then the player receives the highest rating of "awesome". Perfection is not required to fill the meter completely: an awesome can still be achieved with some error (this naturally covers for some inaccuracy in the pitch detection).

An awesome corresponds to 100% accuracy and gives the maximum possible number of base points for a phrase. There is also a score multiplier, which starts at 1x and increases by one for each awesome phrase, to a maximum of 4x. It resets to 1x after a phrase which was not awesome.

Example 1: Rock Band Vocals

- a full combo should always imply 100% accuracy
- however, in Rock Band 3 it is possible to have a "full combo" without getting 100% accuracy, presumably due to a race condition
- some players claim that adjusting the game's internal sync can mitigate this bug
- this bug was also not present in the original game

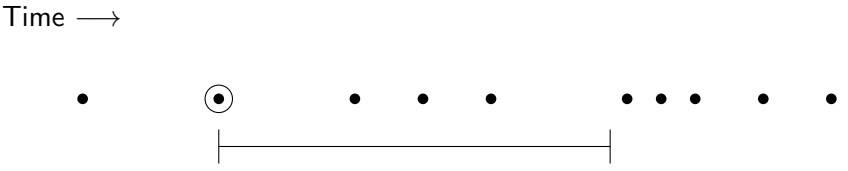
Based on this description, having a full combo of awesomes should imply that:

- The score multiplier rises to 4x within the first four phrases, and then remains at 4x.
- The user interface indicates "AWESOME" as each phrase is completed.
- The maximum base score of 1000 points is awarded for each phrase.
- The accuracy rating presented after the song is 100%.

However, it is possible to have the first two without the latter two. Presumably this is due to a race condition in the game. It is definitely not intentional, and was not present in the original version of the game, whereas it occurs regularly in Rock Band 3 for players who are trying to obtain a perfect score.

Rhythm games often have manual synchronization settings to compensate for latency in input devices, audio output, and displays. Some players claim that adjusting this latency affects this bug, which provides further evidence that it is a genuine race condition.

Example 2: Rock Band Drums



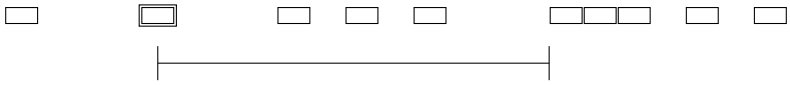
The above is a simplified representation of a drums chart in Rock Band. Some special notes can be used to trigger *Overdrive*, a mode which doubles the value of every note hit for a certain period of time. This is indicated on the diagram by the circled dot and interval below.

Computing the maximum score for a given drums chart is not completely straightforward, since the player has some choice as to where they spend their energy to trigger Overdrive. However, the chart can be modelled as a series of decisions, and dynamic programming can be used to compute the maximum score in time linear in the number of notes.

However, some players can legitimately attain scores higher than this theoretical maximum. Let's investigate how.

Example 2: Rock Band Drums

Time $\longrightarrow$



First of all, each note is better represented by a window of time, rather than a point in time. Any hit within the window – which is approximately 100ms wide – counts as a success. This is standard for rhythm games, although Rock Band is unusual in having only one level of timing. Other games have nested windows giving differing numbers of points.

Example 2: Rock Band Drums

Time $\longrightarrow$

Normally, Overdrive triggered on the boxed note would affect the three notes which have been shaded black. The note which triggers the Overdrive does not gain this bonus.

Example 2: Rock Band Drums

Time $\longrightarrow$

However, by hitting the special note a little later, the hit is still registered, but Overdrive starts later. Thus, it lasts long enough that if the red note is hit early, the Overdrive will also apply to that note. If the rest of the song is played perfectly, this gives a score higher than what the naive model predicts is possible.

Techniques which manipulate the beginning of Overdrive are called "squeezing", and this particular technique is known as "back end squeezing". Because these techniques allow players to obtain scores which look impossibly good, they quickly became famous in the community of players competing for high scores.

Example 3: Final Fantasy Tactics

In this game, the amount of damage done is deterministic, and is affected by several modifiers, including:

- The ability "Defense UP" reduces damage by 33%.
- The status effect "Protect" also reduces damage by 33%, and these stack multiplicatively.
- Hitting a character who is charging a spell increases damage by 50%.

Final Fantasy Tactics is a turn based Japanese-style Role Playing Game. It is somewhat unique in that there is very little randomness in the damage calculation formulae. This is even obvious to the players, since in many cases the amount of damage can be seen as the result of a single multiplication. For example, a character with an attack stat of 10 and a sword of power 7 will always do exactly 70 damage under normal circumstances.

However, there are conditions which modify this base amount; some examples are given here as a typical player would understand them.

Example 3: Final Fantasy Tactics

From the Final Fantasy Tactics Battle Mechanics Guide by Aerostar:

Many factors – including the attacker's abilities and status, the target's abilities and status, as well as Zodiac compatibility – can affect the damage done by weapon attacks. The procedure outlined below is a summary of how to apply all these different modifiers.

1. If target has Defense UP, then ($Sp1 = \lfloor Sp0 * 2/3 \rfloor$), else $Sp1 = Sp0$
2. If target has Protect, then ($Sp2 = \lfloor Sp1 * 2/3 \rfloor$), else $Sp2 = Sp1$
3. If target is Charging, then ($Sp3 = \lfloor Sp2 * 3/2 \rfloor$), else $Sp3 = Sp2$

...

Some of the fans set out to understand the game more throughly. As we can see in the given excerpt, because the game rounds internally after every step, a full understanding of the formula can only be attained by knowing the order in which the modifiers apply.

This is a failure of symmetry, and also a leak of implementation details. Because the number can get smaller first by rules 1 and 2, and then bigger by rule 3, the effect of rounding can actually be amplified. Even more so because the damage is eventually calculated as the above integer SP times some other quantity.

Players can thus naturally observe effects which allow them to deduce this order. In particular, even though $(2/3) \cdot (3/2) = 1$, Defense UP does not always cancel out the extra damage to a Charging character. However, I suspect that much of the information in the Battle Mechanics Guide was gleaned by manually disassembling the game code.

If the game code were written differently, then this lack of symmetry could be removed. For example, manipulating the modifiers as exact rationals, and then applying the compound modifier once, would remove the asymmetry while still allowing it to be easily computed using bounded integer arithmetic.

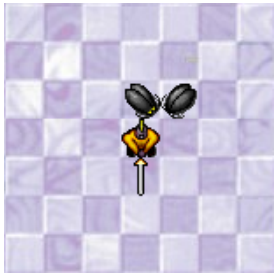
Example 4: Deadly Rooms of Death (DROD)

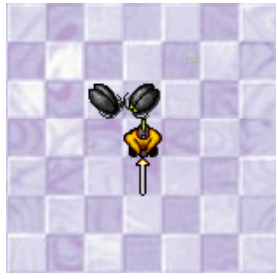
Deadly Rooms of Death is a turn based puzzle game where you are trying to exterminate all of the monsters in a dungeon. Turns alternate between the player and the monsters. This game could be Tech Talk all by itself, so we will focus on just one aspect of how the monsters move.

Every turn, roaches move in the direction of the player. In one turn, they can move horizontally, vertically, or diagonally. So, in this case, both roaches want to move onto the marked square.

How is this resolved? Well, any solution which preserves the symmetry will be inadequate in some way. The game simply assigns each monster a secret number, which determines its priority of movement.

Example 4: Deadly Rooms of Death (DROD)





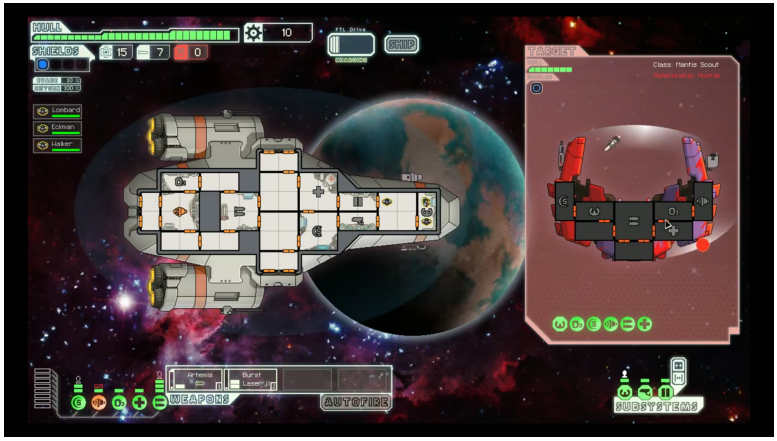
Here are the actual outcomes, given the two possible monster orderings. In the left image, the left roach moves first, and likewise for the right image.

The creator mentions in the bonus content that some puzzles were rejected from inclusion in the game because they were too sensitive to the monster ordering.

Example 4: Deadly Rooms of Death (DROD)

There is also directional asymmetry. Here we can see that roaches slide around obstacles when they move diagonally, but vertical movement takes priority over horizontal movement.

Example 5: FTL: Faster Than Light



A video which shows the basic game play:
`https://www.youtube.com/watch?v=Cxl_VnZgNW8`

There are many interesting things about this game, but we will focus on the combat. The enemy ship is shown in a window on the right side of the screen. When a shot is fired from the player's ship, it is shown going off the screen, and then it appears in the window coming from a random direction, and hits the enemy ship.

In this screen shot, we can see a missile which is about to hit the enemy ship. It was fired from the missile launcher seen sticking out of the upper side of the player's ship. This is probably easier to understand by watching the linked video which gives a brief demonstration of game play.

Stop and think about how you would model these combat mechanics. What are the entities in the model? What level of detail exists in the model?

Example 5: FTL: Faster Than Light

Video: The Miracle Shot

`https://www.youtube.com/watch?v=um1zPzbRw00`

This video speaks for itself I think, especially if you have some experience with the game.

The player in the video is surprised, as we can hear. I think pretty much everyone would be, since it is so natural to form a model for this game which is far too simplified.

I have encountered this miracle myself while playing the game, and apparently there are other strange things which happen because the game world is simulated on a more detailed level than the players expect.

Conclusion

The software we write easily gains complexity, which is irreducible in the sense that it cannot be abstracted away.

This is especially embarrassing when the abstraction we would like to have is part of the specification of the software!

References

- Rock Band
- Final Fantasy Tactics
  - Battle Mechanics Guide by Aerostar
- Deadly Rooms of Death
- FTL: Faster Than Light

Questions?

You should definitely consider playing each of the video games discussed in this talk. The Final Fantasy Tactics Battle Mechanics Guide is also impressive in its devotion, shown by the thoroughness with which it describes the game mechanics.