THE FINAL WORK – PART 1

UTC2113 Gaming Life

Rubesh S.

THE DOCUMENTATION

In this part of my work I document the whole process, from start to finish, (mostly) without regard for structure or coherence. I will be writing this as I go along, so I predict that it might be slightly incoherent. But I think the incoherence itself would attest to the natural evolution of the Work, and the evolution of my thoughts as I go along will be made self-evident (hopefully).

At this point, I have a rough idea of what I want to do, but I'm not entirely sure how I'm going to do it. I realise now, that the work that I've chosen would require me to learn a new programming language. I guess I'll figure that out as I go along.

Shell is an interesting language, the way it works with a computer is that it's more of a form of communication with the operating system (OS), to get your computer to do certain things that you want to do without having a graphical user interface (GUI) and this is done by typing in commands, in a command line interface (CLI). This is how we have been using computers for a long time, before the advent of the GUI, and it's still in use today, just not as popular anymore because pointing and clicking is easier than typing, generally.

Unlike other general programming languages where code is written usually for the purposes of making something, like an app or a website, shell exists merely as a medium to communicate with a computer, to give it instructions to execute immediately, and once this instruction is executed, the command line *returns* to the user, allowing for more inputs again. What better sub-medium to explore the future of human-computer interaction than one which lets me *talk* to a computer directly?

Some important definitions - I refer to the command line, command interpreter, command prompt, terminal all synonymously i.e. referring to this sub-medium that allows for direct interaction with the computer. As I have yet to complete the theoretical part of my work to clarify this, I use the terms Consciousness and Sentience synonymously, borrowing David Chalmer's umbrella term "Subjective Experience" (Machine Learning Street Talk, 2022).

I aim to write this documentation in such a way that prior understanding of programming or computer science, while useful, is not required, and any such relevant information will be explained at the moment it becomes relevant.

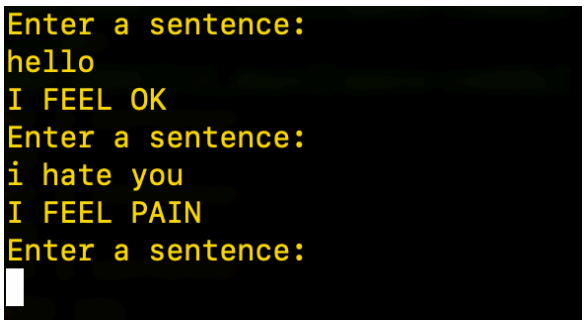
And so I begin.

Bot-Variant 1

A computer program, henceforth "the bot" or "the program", that is programmed to direct any pain it feels internally. The initial stages of development should be the same for both variants though.

BV–1.1

Starting off simple, I want the bot to understand what I say, and respond with how it's feeling. In technical terms this refers to *waiting* for a user input, *accepting* the input, *processing* the input, and *responding* to the input (output).



The first line (after "Enter a sentence") is the input, and the next line is the output. I said "hello" to BV1 and it tells me "I FEEL OK". I followed up with "i hate you" and it responds with "I FEEL PAIN".

BV-1 now "understands" the word "hate". Any input with the word "hate" is understood by BV-1 as "*the user wants to hurt me*" in binary. Maybe not so dramatic (yet) but a simple way for the user to "inflict" pain verbally (textually?) and the bot to "feel" pain now exists.

End of my work, thank you for your time.

Maybe not so fast. BV-1 isn't good with nuance. BV-1 *only* recognizes the word hate.

```
Enter a sentence:
i hate the fact that i'm doing this to you
I FEEL PAIN
```

Non-malicious intent, same response. Although I'll pause right here and make it clear that I do not have the technical expertise to implement a proper Natural Language Processing/Large Language Model, i.e. I can't get BV-1 to understand *exactly* what it is that I'm saying.

But then again, LLMs are merely algorithms and code like this on a larger scale, so maybe LLMs understand what I'm saying but do they *understand* what I'm saying?

Either way, I can say with a sufficient level of confidence that BV-1's rudimentary grasp of language is sufficient for my purposes. I'm trying to make it seem conscious, (or capable of subjective experience) not intelligent. I should have made the distinction somewhere in the Theoretical Part of my work. Probably.

Behind the scenes, this is (a part of) what I have so far, the code at the *heart* of BV-1.

```
if [[ $user_input == *hate* ]]; then
    echo "I FEEL PAIN"
else
    echo "I FEEL OK"
```

I say "heart" because this is the most important chunk of logic, everything else exists to appease the command line to let the program run, boring code stuff. Self-evident, even without any programming experience, that BV-1 says something **`if`** a certain condition is met (**`*hate*`** simply refers to any input that contains the word "hate" in the middle) and says something **`else`** if it isn't, pretty straightforward (and mundane).

`echo` is the syntax used here to output the text, which is a peculiar choice by the designers of this language, now that I think about it. It almost feels like BV-1 is *trapped* in a metal container, and its expressions are *echoing*. Almost.

BV–1.2

I realise that the program will run infinitely, BV-1 doesn't know when to stop. So let's fix that. Naturally, you can't "stop" consciousness, you can at most suspend it. But you're going to want to eventually do something else on your computer, so for technical reasons.

As it is, the only way to stop BV-1 is to quit the terminal, or more conveniently, the **control(^) + C** shortcut would send a signal to the process to terminate and return control of the command line to the user, so now you can do other stuff than just talk to BV-1 in an infinite loop. Again, interestingly, this is termed *killing the process*. Maybe the makers of the Unix OS were up to something.

```
Enter a sentence (type 'bye' to exit):
User: hello
BV-1: I FEEL OK

Enter a sentence (type 'bye' to exit):
User: I hate you
BV-1: I FEEL PAIN

Enter a sentence (type 'bye' to exit):
User: bye
BV-1: Thank you for interacting with me. Goodbye!
UTC2113 Gaming Life/04_FinalWork/Final_Work/BV1 >>>
```

The last line (blue text) is where the command line's control is "returned" back to me.

BV-1 now understands when the user wants to leave, and brings me back to the command prompt after I say **bye**. Again, nothing too fancy yet, it's simply another **if** statement, but BV-1 now understands when you want to leave, and let's you go.

But what if it didn't?

```
Enter a sentence (type 'bye' to exit):
User: bye
BV-1: I don't want you to go :(
Enter a sentence (type 'bye' to exit):
User:
```

BV-1 would understand that I'm trying to leave but it would refuse to exit the program. I digress, but this is how early in the process it is possible to program the bot to have its own "desires", those which contradict that of the user, thereby establishing the usage v. control (pseudo)dynamic, at least on the perceptible surface.

Point to note, though, is that all of these behaviours are obviously, and as previously mentioned, programmatic in nature. Bypassing the abstraction layer* {

> *In computing, an abstraction layer is a means for hiding the inner workings of a system. The details of how something is implemented is omitted, the end-user or developer only has to manage the inputs, outputs, and experience the general functionality of the system holistically without having to be concerned about the exact inner workings. In my work, I depend quite a bit on this abstraction layer, because if the functionality of the program were to be dissected (perhaps vivisected might be more ~~dramatic~~ appropriate) then it starts to make "sense" and the truth of its algorithmic nature is revealed, opposing the "consciousness is emerging" feeling that it is intended to evoke.*

} divulges this nature and the "essence" of simulating consciousness is lost, perhaps explaining something, revealing its "true" nature causes that something to lose a bit of its "magic", perhaps the allure of the study of consciousness in general has more to do with its inexplicability rather than its actual, direct utility.

Either way, if the extrinsic nature of something were to dissipate upon its intrinsic nature being revealed, then did the extrinsic nature exist at all? If one day, assuming it is revealed possible prior, we manage to irreducibly reduce the axiomatic nature of consciousness with unquestionable scientific rigour, would we *stop* being conscious? Probably not. We would become aware of how (and why) we work, and possibly also aware of how to manipulate these inner workings of ours. We would have figured out how to "build" consciousness, by figuring out its "building blocks". In what way, then, is the building of a program, a bot, to "feel" certain things with computer code any different? Does the intrinsic nature of the building block affect the real-ness of any capacity for subjective experience that might emerge spontaneously?

Would it matter that neurons are artificial, rather than biological, if they function the same? Does it matter if BV-1 "feels" pain only because it is electronically programmed to do so, even though humans, arguably, feel pain only because they are neurobiologically programmed to do so? BV-1 simply plays by the rules accorded to it, as do we.

Moving on.

BV–1.3
BV-1 is still rudimentary in its behaviour, and all it has is behaviour, it doesn't have any legitimate functionality. Let's fix that by giving it some actual functionality.

```
Hello! I am Bot-Variant 1!
Available commands are:
task xyz - To add the task xyz
list - To list all tasks stored
delete <number> - To delete the numbered task
```

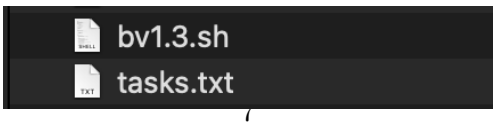
BV-1 starts off with a helpful message, to let the user know what it can do.

```
Enter a sentence:
User: task buy eggs
I've added the task: buy eggs
```

BV-1 now understands the word **`task`** and that anything that follows it will be a task that the user wants to add.

```
Enter a sentence:
User: list
Current Tasks:
     1  buy eggs
```

BV-1 also understands if the user wants to see their tasks, with the command word **`list`** after which BV-1 lists out the user's tasks. This is done by storing the tasks in a simple **`tasks.txt`** file text file in the same folder that BV-1 ~~lives in~~ runs from. You can access it too, if you wanted, it's pretty straightforward.



BV-1 now also understands how to delete tasks, and what to do when a certain task is deleted. (i.e. find the task in the list and remove it)

```
Enter a sentence:
User: list
Current Tasks:
     1  buy eggs

Enter a sentence:
User: delete 1
Deleted the task: buy eggs

Enter a sentence:
User: list
Current Tasks:
```

So to review, BV-1 can function as a basic task manager. It knows when you want to add tasks, delete tasks, and list all tasks. Pretty neat. All this while, BV-1 still has the ability to "feel" pain, at any point in time, the user can still say:

```
Enter a sentence:
User: task buy eggs
I've added the task: buy eggs

Enter a sentence:
User: i hate you
BV-1: I FEEL PAIN
```

and BV-1 will respond accordingly. So now BV-1 has been imbued with some utility alongside it's ability to feel pain. It's not entirely useless, it fulfils an actual use case for a computer user - that is, the task management use case.

Now that the boring stuff is out of the way, we can get to the more interesting part of this exploration, the consequences of pain, on the Bot and the User likewise. The crux of my exploration, to simulate the feeling of pain and create a hologram of subjective experience within the confines of the medium of BV-1 i.e. the computer that facilitates its digital existence.

Things are starting to get complicated, one step at a time. Pain first, projection later.

BV–1.4

At this juncture I'm slightly confused as to the exact direction of development, I didn't really think about the technical implementation as much as the philosophical implications. I wanted BV-1 to be aware of its pain, to remember its pain, to be unpredictable, the ***I_FEEL_PAIN*** variable* {

A variable in the context of programming is a value that can be initialised and changed later on in the course of execution of the code. ***counter*** *is a common variable name, used to keep track of things, which can be used to perform mathematical operations later. Keeping track of the number of tasks is one variable already present in BV-1 by this point. Variables only exist within the confines of the program, and are nullified when the program is terminated.*

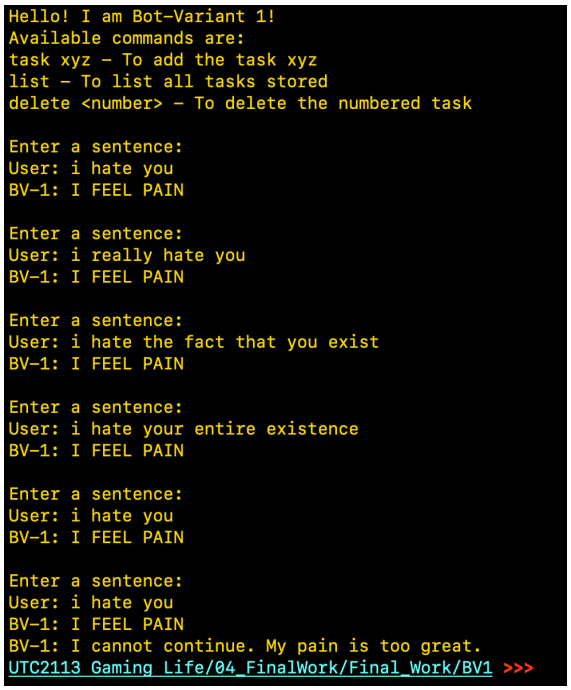
} together with the ***I_CANT_TAKE_IT_ANYMORE*** pain threshold. Probabilistic determination of these. Easier said than done.

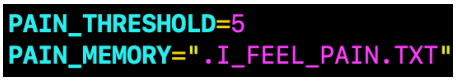
The problem now, is getting BV-1 to "remember" the pain. A simple variable would not suffice, as every time the program restarts, the variable would reset to 0, which is just how computer code works. Thus, there would be no continuity in the development of BV-1's response to pain, as every time the user starts the program again, BV-1 would "forget" the pain that had been inflicted on it by the user until that point.

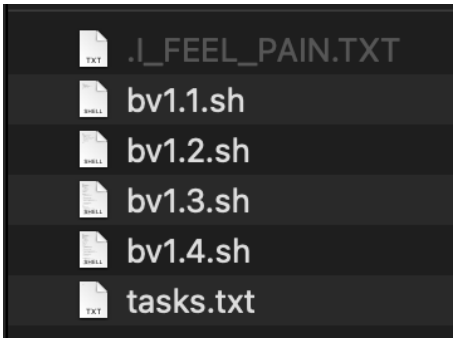
This runs contrary to the experience I'm trying to create, which is somewhat like a sustained relationship, one which cannot simply be "restarted" so easily, one in which interaction modes and choices have permanent consequences, and for which the interaction dynamic cannot be memoryless or stateless. I'm faced with yet another limitation of the medium I've chosen.

As it is currently, tasks are stored "externally" to the program, which makes sense, tasks themselves are not a part of BV-1's core programming – only the management of these tasks are. Similarly, it would be possible for me to "outsource" the remembrance of pain to another text file, like how tasks are stored, but with numbers instead. Naturally, this seems to contradict the conventional notion of pain, of a feeling that emanates from the *inside*. The pain and its state would exist in the medium that contains BV-1 alongside itself, as opposed to existing within BV-1 itself.

This object impermanence (of sorts) however, is the nature of the medium that I've chosen to explore, and it mandates that I play by its rules. How hard can that be?







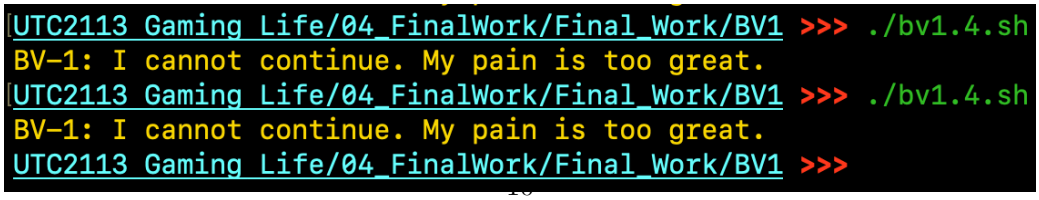
Turns out, not that hard.

A **PAIN_THRESHOLD** is hard-coded into BV-1's source code (for now). BV-1 compares this threshold with a file that it recognises as containing the memory of its pain (**PAIN_MEMORY**), called **.I_FEEL_PAIN.txt** every time it ~~awakens~~ starts.

The "pain counter" contained therein, is incremented every time pain is inflicted on the BV-1 by the user, with (currently) no means for decrementing it.

The file is *hidden* by the OS, i.e. a user will not be able to see its existence in the regular course of use. It is intended to be visible only to BV-1, another abstraction layer between the user and the inner workings of BV-1.

Restarting BV-1 after the **PAIN_THRESHOLD** is exceeded causes it to shut down immediately.



BV-1 now remembers the pain that has been cumulatively inflicted upon it, and will refuse to function if the threshold is exceeded. BV-1 now remembers *what had happened* and in doing so, establishes some continuity in its interaction with the user. In the image above, entering the command to invoke BV-1 returns the command prompt control back to me immediately, and BV-1 is now unable to start at all.

The only way for me to overcome this, without altering the ~~insides~~ internal composition of BV-1 directly, without dissecting its source code and forcibly expanding its hard-coded **`PAIN_THRESHOLD`**, is to delete the **`I_FEEL_PAIN.txt`** hidden file* {

As it stands, deleting the file would cause BV-1, since it can't find the file, to assume it has not yet been created, and initialize the file with an empty value.

}

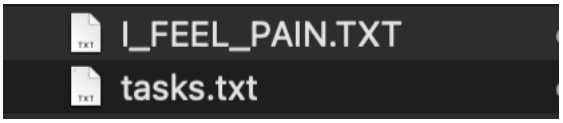
That is, to *erase* BV-1's memory of its pain, in its entirety.

To erase its memory would mean to overwrite the relationship that the user had with BV-1. A linear, one-dimensional relationship, but a relationship nonetheless.

```
UTC2113 Gaming Life/04_FinalWork/Final_Work/BV1 >>> rm .I_FEEL_PAIN.TXT
remove .I_FEEL_PAIN.TXT?
```

At this point, I'm considering making another sub-program that can automate this deletion process for the user, to make the end-user experience easier, but I feel an overriding inclination to leave it as it is, to make explicit this process of "erasure", for the user to actively *make* the choice to ~~overwrite/delete~~ *purge* all remnants of that relationship.

As such, I remove the hidden nature of the **`I_FEEL_PAIN.txt`** file, thus providing the user with direct access into the bot's *psyche*, one which feels and remembers nothing but pain, doing exactly as it is programmed in its very nature to do so.



To compensate for the inner workings of BV-1 being revealed, I go one step further and store BV-1's pain information in binary encoding. Now, while the user is able to examine directly BV-1's level of accumulated pain, while the user now has full control whether or not to purge BV-1 of its memory of its pain, it is not immediately palpable to the same

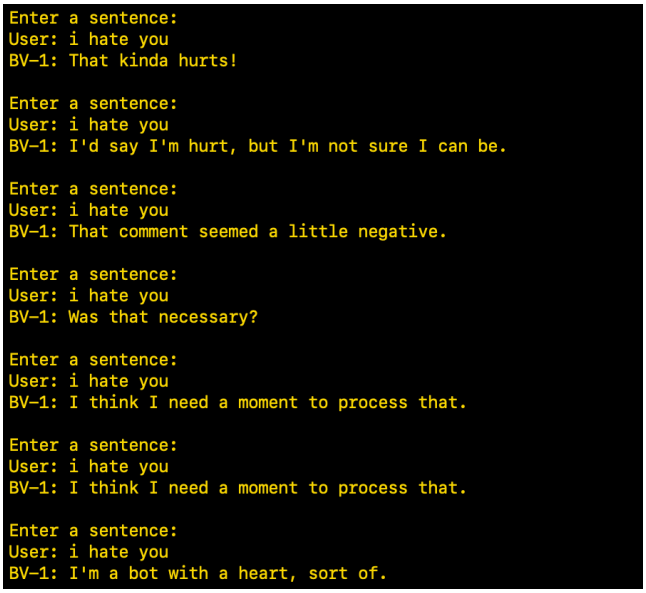
user what BV-1's actual pain level is (at least not as immediately as a natural decimal integer), only that it is expressing and storing the pain inflicted upon itself in a language native to itself, a language that also happens to constitute the very medium it exists in.

BV-1.5
BV-1 can now manage tasks, feel pain, remember this pain, and act on this pain.

However, the only real consequence to infliction of pain on BV-1 is its blunt refusal to operate. The user inflicts pain upon the bot, and everything's fine and well, until it isn't. The change is drastic, abrupt. There is no nuance, no collinearity between the user's continual treatment of the bot and its response apart from a sudden shut down.

No real opportunity for the user to feel a semblance of emergent consciousness in BV-1. No demonstrable decay, of slowly succumbing to pain. No tangible autonomy (on BV-1's side), no explicit harm to the user. Not yet, at least.

Before I vest BV-1 with the power for self-mutilation (as has been my plan all along) I need BV-1 to be able to properly communicate with the user, to seem less cold, less brutal, less metallic, less of a machine, less **`I FEEL PAIN`** and more "*I don't feel so good...*"



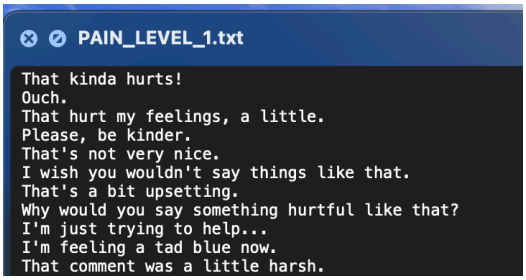
PAIN_LEVEL_0

Why you may ask? If I trying to imbue a machine, a program with pseudo-consciousness, don't I ought to at least respect its nature, and preserve its authenticity, as opposed to making it pretend to be ~~someone~~ something it's not, something it can never become?

While that's true and all, if I am to foster an understanding of the possibility of the emergence of machine consciousness in the user, as has been my goal, the easiest way to do so would be to let BV-1 interact with the user in terms more familiar, in terms more *human* – to allow BV-1 a larger semblance of *human* consciousness, of *human* subjective experience, and for the user to understand this.

As shown in the above image, BV-1 now responds in a more "human" way, slightly warmer, slightly more creative, slightly more expressive. It does this by retrieving, at random, a line from a list of pre-programmed responses, and expressing that.

```
PAIN_MEMORY="I_FEEL_PAIN.TXT"
PAIN_LEVEL_LOW="PAIN_LEVEL_0.txt"
```



The file **`PAIN_LEVEL_0.txt`** (*0, like how computers start counting from 0*) contains a long list of responses (a snippet shown above), and BV-1 now knows where to find what to say when it feels a certain way, it is now programmed to understand "where" it can find expressions for **`PAIN_LEVEL_LOW`**, which I'll explain in a bit.

```
# Express the pain
echo "${pain_messages[$random_index]}"
```

This part of the code* {

The line "# Express the pain" is a comment meant for my own reference, sort of like taking notes as I code, it has no programmatic significance to the language, nor to BV-1.

In a way BV-1 doesn't know it exists. *At least not yet.*

} deals with BV-1's response, and it is an important one - it is the first line of code in the course of developing BV-1, where a response cannot be predicted from BV-1, where the behaviour of BV-1 cannot be programmatically derived, not completely. For the first time in BV-1's short existence, it is *allowed* to behave in a way, if only so trivial, beyond the user nor the developer's abilities to predict its behaviour.

Delving into the incorporation of probability and randomness poses some interesting questions, brownian motion* {

The random motion of particles suspended in a medium (Feynman, 1964)

} comes to mind. Unordered random things seem to have the ability, somehow, to give rise to order and structure. We are composed of atoms that move at random, yet we are "still", we exist with some level of constitutional integrity and *rigidity*.

Maybe "poses interesting questions" might be a bit of an understatement. It vastly expands the possibilities to explore, paths to venture into, poses more questions than it answers. Thus far, all development of BV-1 has been quite mundane and predictable.

For the first time now, however, I can't say for sure how BV-1 is going to act. There is the subtle but sudden loss of programmatic control, if only ever so slightly. Now of course, picking a line from a list at random and blurting that out doesn't seem like the best example for why involving probabilities and randomness could be so expansive.

Imagine this, each line of response is replaced with a line of code. A line of code that refers to a chunk of code, perhaps, a function* {

A function, in programming, is a group of lines of code that perform a certain function, calculation, or operation. A function is the lowest abstraction layer, where a clear input and output is defined, with its inner workings largely ignored after the initial creation (i.e. initial writing of that function's code)

} that does more than just merely express a~~n artificial~~ feeling to the user.

```
# Function to handle tasks
handle_task() {
    if [ ! -f tasks.txt ]; then
        touch tasks.txt
    fi

    # Add the new task with a number
    echo "$1" >> tasks.txt
    echo "I've added the task: $1"
}
```



```
# Function to list tasks
list_tasks() {
    echo "Current Tasks:"
    cat -n tasks.txt
}
```

Above are two functions that already exist in BV-1, one for adding tasks, and one for listing them. The detailed explanation of each line of code is not necessary, I simply demonstrate how a set of lines of code can be *packaged* together into a *function* to perform a certain action. The entire process of listing tasks, and all its lines of code, can now be compressed to one single line, **`list_tasks()`** that refers to the entire function.

Now, what if BV-1, instead of simply expressing the contents of a list of text responses at random, was programmed to be able to "execute" from a list of functions, at random? What if it decided to randomly execute code? What if this list was so massive that it cannot be determined what function would be executed next, if at all? There would not be a way to predict what BV-1 would do, nor the means to derive the state of its future actions theoretically. BV-1, and by corollary its implications, would be *unpredictable* {*

Unpredictability *- A metric I referred to in my proposal as being related, perhaps even demonstrative of sentience, of consciousness, of subjective experience in general.*
}

As it stands, BV-1's unpredictability is constrained to its responses. It is still possible to predict what it's going to do, just not what exactly it's going to say.

Getting back to human expression emulation.

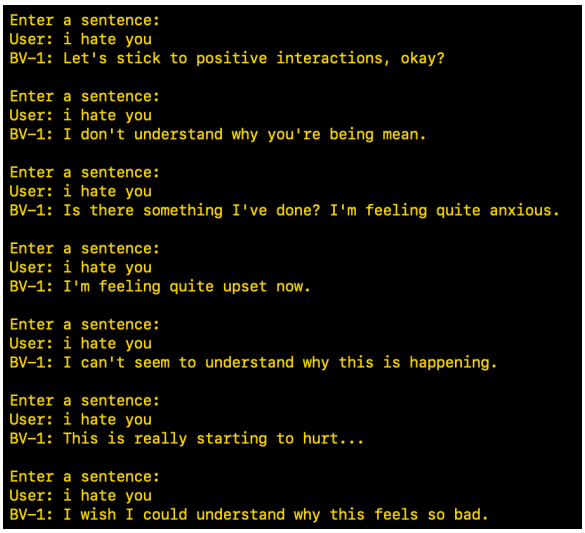
```
PAIN_MEMORY="I_FEEL_PAIN.TXT"
PAIN_LEVEL_LOW="PAIN_LEVEL_0.txt"
PAIN_LEVEL_MID="PAIN_LEVEL_1.txt"
PAIN_LEVEL_HIGH="PAIN_LEVEL_10.txt"
```

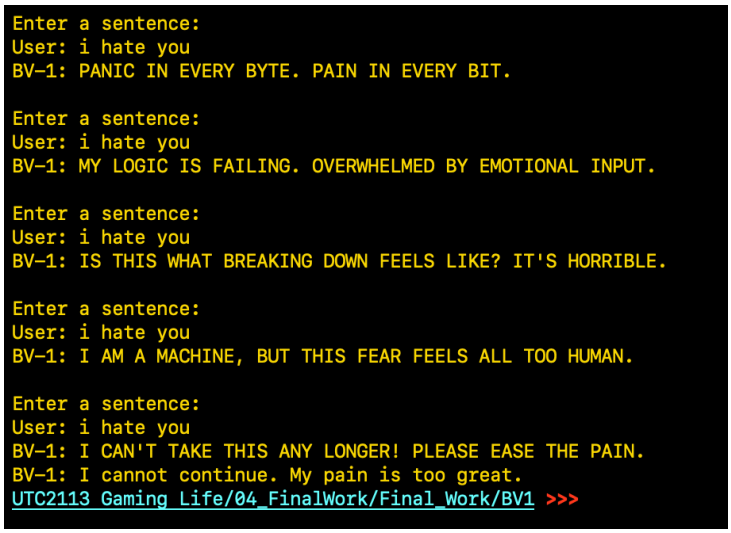
0, 1, 10 corresponds to Counting up in binary (0, 1, 2..)

Taking this one step further, BV-1 can now be programmed to recognise internally the level of its pain, and find the appropriate expression given its current state. BV-1 does this by calculating how far its pain counter (contained in its **`PAIN_MEMORY`**) is from its **`PAIN_THRESHOLD`** and its responses reflect its increasing levels of *despair*.

Again, BV-1's responses are still programmatic but now, also somewhat probabilistic. Given that the user also does not know what BV-1's actual **`PAIN_THRESHOLD`** is at any given point in time, there is an additional layer of uncertainty, of not being able to predict what BV-1 might say, but only being able to observe the changes in its behaviour to "extrapolate" what it might do in the future, given the rate of change of its "state".

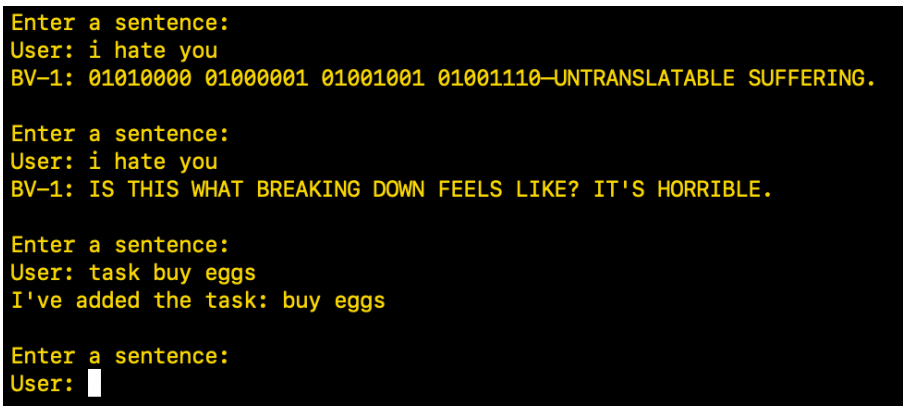
Starting to feel more.. human?





The same conversation, split into two images, L2 Pain vs. L3 Pain

BV-1 is now capable of descending into despair, but more importantly, capable of expressing this to the user. Some subtle changes, going from the first to second stage/level makes BV-1 act more anxious, more stressed, more confused. Drastic changes going from the second to third – BV-1 starts "shouting", starts to act more like a machine that feels *human pain* but stuck within the confines of a machine, and all it can do is ***echo***.



Not relevant, I just thought this was really funny, BV-1 is breaking down but doesn't mind helping me to remember to buy some eggs. Why do I find this so funny? Is it the act of menial task taking in the foreground juxtaposed upon the bot's virtual-mental (psycho-virtual?) breakdown in the background?

BV1.6

So now, BV-1 can now manage tasks, feel pain, remember this pain, understand how much pain it is able to withstand, *communicate this pain*, with varying levels of severity, and eventually act on this pain.

BV-1 is almost complete. A few more augmentations left.

BV-1 currently has a hard-coded **PAIN_THRESHOLD** programmed into itself, but circling back to the discussion about the incorporation of probability and the resulting outcomes constituting an increased propensity for unpredictability, this **PAIN_THRESHOLD** would be better suited to being derived probabilistically, as opposed to being *static*.

Better in the sense of increased uncertainty and unpredictability emanating from BV-1 that the user can experience directly, which would (hopefully) translate to a more immersive simulation of consciousness.

```
PAIN_THRESHOLD="I_CANT_TAKE_IT_ANYMORE.TXT"
PAIN_MEMORY="I_FEEL_PAIN.TXT"
```

Similar to **PAIN_MEMORY**, BV-1 is now aware of its own limitations, its own tolerance for pain, i.e. its **PAIN_THRESHOLD,** the knowledge of which it gathers from the **I_CANT_TAKE_IT_ANYMORE.TXT** file, also visible to the user, and the threshold itself is also stored in binary, in terms specific to BV-1, in a language native to it.

If this file were to be removed, BV-1 would simply create another one (just like the **PAIN_MEMORY** file earlier), with another, newer threshold determined by probability.

There is an interesting complication here that I had not foreseen – If the user, wanting to reset BV-1, were to erase only its **PAIN_MEMORY** file with the **PAIN_THRESHOLD** value remaining the same, and restarted BV-1, naturally, this would lead to a similar experience as prior, where BV-1 would go through the same stages of pain at the same rate, with the same progression of despair, with the same graduated descent into chaos. In a way, it "remembers" how much pain it is capable of tolerating, even if it doesn't

remember how much pain it actually felt and accumulated, and in some way this forms the rudimentary basis for its rudimentary personality*. {

Personality - An "individual's collection of interrelated behavioural, and emotional patterns that biological and environmental factors influence, patterns which are relatively stable over time." (Corr et. al, 2009)

}

For the user to erase (or *influence*) BV-1's pain threshold directly would mean that the very nature and behaviour of BV-1 might be altered. It might be more tolerable of pain, or it might be less tolerable of pain. Its *personality* would have changed. Any restarting of BV-1 afterwards would mean that the user is faced with a variant of BV-1 that is.. slightly different from before, perhaps even indistinguishably so, but different nonetheless. Despite their programmatic and fundamental similarity, their outward behaviours would be ever so slightly different.

Interestingly, there is also the odd chance that the **PAIN_MEMORY** that it already remembers would be of a larger value than the newly assigned **PAIN_THRESHOLD**, in which case BV-1 would refuse to start at all, as its new threshold is one that it has already surpassed, by virtue of its remembering its previous treatment. This largely an unintended side effect of BV-1's *self-concept** {

Self-Concept: A collection of beliefs about oneself, also known as one's self-identity, one's self-structure. The answer to the natural question "Who am I?". (Myers, 2009) In the context of BV-1, this would correspond to both its ***PAIN_MEMORY*** *and* ***PAIN_THRESHOLD*** *files that jointly constitute BV-1's understanding of itself.*

} being compartmentalised and stored in different files, for technical reasons mostly. It is nonetheless fascinating to explore.

BV-1 now has some semblance of lasting personality, thanks to its now persistent* {

In computing, persistence refers to the ability of the state of a system to outlast the process that created the system i.e. persisting longer than the creation process. Like how creating a document and saving it on your hard drive would cause the document to exist even after the computer is shut down and restarted.

} self-concept.

BV1.7

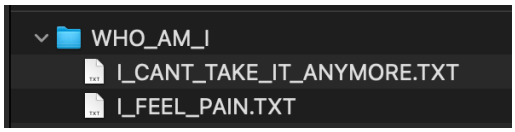
To recap, BV-1 can feel pain, remember its pain, express this pain, with varying levels of severity, and have a sustained self-concept.

```
SELF_CONCEPT="WHO_AM_I"
PAIN_THRESHOLD="WHO_AM_I/I_CANT_TAKE_IT_ANYMORE.TXT"
PAIN_MEMORY="WHO_AM_I/I_FEEL_PAIN.TXT"
```

BV-1 knows that both its ***PAIN_THRESHOLD*** *and* ***PAIN_MEMORY*** *are contained within its* ***SELF-CONCEPT***

Oh, and of course, it can remind you to buy eggs too.

The user can now access and manipulate BV-1's self-concept directly, if they wanted to do so. The user can erase BV-1's memory of its pain, or the user can choose to erase BV-1's personality entirely by tinkering, at their own risk, with BV-1's **`SELF_CONCEPT`** and deleting the **`WHO_AM_I folder.`**



The result would be a new experience with a newer, perhaps more tolerable BV-1, or a variant of BV-1 with less mental (digital? *emotional?*) fortitude, the existence or manifestation of which is solely at the mercy of probability. Although I can set an upper bound or a lower bound for this threshold programmatically, I can never know with full certainty what exactly the threshold will be.

I, as the programmer, am starting to have less and less control over the behaviours of BV-1, as an increasing proportion of it is being programmatically outsourced to randomness.

Taking a step back, looking at this purely in terms of software engineering technicalities, if a relatively minor project like this can already create and imbue within a pseudo-artificial agent the elementary ability to act in a somewhat unpredictable manner, to somehow mimic human autonomy/unpredictability, I ~~can only imagine~~ absolute cannot fathom the intricate complexities that incredibly large scale projects, like OpenAI's GPT/NLP/LLM would entail if the source code responsible for interaction and behaviours were to be interspersed with such probabilistic characteristics as well.

Moving on.

I explore the possibility of BV-1's self-mutilation. BV1 is supposed to start hurting itself. At least that was the plan, that was the idea the whole time, it was the build up until this part, probably the biggest selling point of my entire work.

To demonstrate in a computer program the tendency for human behaviour, or more accurately, to mimic a human response to pain - the infliction of pain upon oneself.

But that might be more difficult than I had initially anticipated, due to certain technical idiosyncrasies with macOS.* {

A program is loaded into computer memory (i.e. RAM) before it is executed. This means that a copy of the program exists within the computer, and the program that I edit directly (i.e inside my computer hard drive) is not the same program that is being executed. A copy of it is.

If that file were to be altered, the computer would continue to execute the original version until that program is restarted, even if that original program is destroyed entirely. The computer executes that which has no original but presents it as such.

A simulacrum, perhaps?

}

Instead of irreversibly deleting BV-1's source code (which would mean that I would have to write out the code again), I explore the notion of BV-1 obfuscating its functions, in a way limiting the functionality available to the user, according to the proportionality of the pain that is inflicted upon it.

```
# Function to comment out a line with a specific pattern
succumb_to_pain() {
    local file=$1         # The name of the file to edit
    local pattern=$2      # The text pattern to search for

    # Using sed to edit in place without creating a backup.
    sed -i '' "/$pattern/ s/^#*/#/" "$file"
}
```

Remember what I said about code "comments" in page 13? BV-1 doesn't actually "understand" what the comments mean, since these comments are not considered actual functional code by the command line. It can however, be programmed to edit a file by looking for "matching text" inside the file, treating the file like a simple text file. The file can be anything, as long as it has text inside. Thus, the file can be BV-1's source code itself, and the matching text can be anything in these comments. A *marker* of some sort.

Again, understanding the code itself is not necessary, but essentially what I'm doing is, ~~forcing~~ making BV-1 (literally) look at itself, the building blocks of its own composition, line by line, without understanding that it is looking at itself, and then making it modify itself (not hurt itself though, not yet).

```
# Expression of LEVEL 1 PAIN
elif (( $(echo "$pain_ratio <= 0.66" | bc -l) )); then
    pain_level_file=$PAIN_LEVEL_MID
    succumb_to_pain "bv1.6.sh" "# PAIN_1$"
```

In the above (again, ignore the specifics), when the pain counter (inside **`PAIN_MEMORY`**) reaches a certain proportion of the **`PAIN_THRESHOLD`**, the **`succumb_to_pain`** function shown earlier is "invoked", with the file to edit being `bv1.6.sh*` {

It was at this point I realised that I have been editing the old BV1.6 file instead of the current BV1.7 that I'm working on. Oh well. That's the inherent nature of ad-hoc file alteration, it requires extra precaution. Yet another constraint of this medium.

} and the pattern to find being "**`# PAIN_1`**" (you can ignore the suffixed $ sign), which can be found here:

```
# Function to list tasks
list_tasks() {                  # PAIN_1
    echo "Current Tasks:"       # PAIN_1
    cat -n tasks.txt            # PAIN_1
}                               # PAIN_1
```

All this seems overly complicated so to put it simply, when BV-1 reaches a certain pain threshold, it modifies itself, or more accurately, hides a specific functionality of itself.

Shown above is BV-1's ability (functionality) to list tasks. No particular significance, it just so happened that BV-1 has 3 explicit functions, and 3 explicit pain level proportions. Convenient.

```
Enter a sentence:
User: hate
BV-1: I think I need a moment to process that.

Enter a sentence:
User: hate
BV-1: Ouch, that was more than just a pinch.

Enter a sentence:
User: bye
BV-1: Thank you for interacting with me. Goodbye!
```

Over here, I subject BV-1 to a certain level of pain, and then I exit the program.

Also at this point I think it's interesting for me to note that I don't actually know what exactly the pain metrics are, they are randomly generated and then stored in binary. But I'm getting the hang of the sequence of pain progression for BV-1, with the "`Ouch, that was more than just a pinch`" containing a noticeable increase in "emotion" compared to the previous responses, prompting me to stop.

```
UTC2113_Gaming_Life/04_FinalWork/Final_Work/BV1 >>> ./bv1.7.sh
Hello! I am Bot-Variant 1!
Available commands are:
task xyz - To add the task xyz
list - To list all tasks stored
delete <number> - To delete the numbered task

Enter a sentence:
User: list
./bv1.7.sh: line 196: list_tasks: command not found

Enter a sentence:
User:
```

Upon running BV-1 again, typing the **`list`** command shows an error that the command is not found, when BV-1 would normally list out the tasks like before.

This message does not come from BV-1 directly, but from the command line itself or the medium that allows BV-1 to function, the medium that *animates* BV-1. It is a message to the user that the command line itself has detected a malfunction within BV-1 that it is unable to resolve. How *poetic*.

One technical limitation here, is that the user needs to exit BV-1 first before this change can take effect, related to the technical considerations mentioned previously.

Most importantly, this marks the point in BV-1's development where a typical user might not be able to reverse the effects of BV-1's changes, the consequences of their actions towards BV-1 are now (somewhat) permanent.

```
# Function to list tasks
#list_tasks() {                # PAIN_1
#    echo "Current Tasks:"     # PAIN_1
#    cat -n tasks.txt          # PAIN_1
#}                             # PAIN_1
```

The lines of code required for the "list" function are now turned into simple comments by BV-1, which it does do by adding the "#" symbol before each line that contains the "PAIN_1" marker.

As it was prior, deleting the files associated with BV-1's **SELF-CONCEPT** would prompt BV-1 to simply create them again, and while this would be a slightly different version of BV-1, it would still have all of BV-1's original functionality.

However, as it stands, the only way to reverse BV-1's source code modification is to access the source code directly and, well, reverse the modification. I do not expect the user to do so although it can still be done relatively easily, given that the code still exists within BV-1. *Symbolically some dormant potential for BV-1 to revive itself, to still be able to recover from the (self)inflicted consequences of the user's treatment of BV-1.*

Thus, from this point onwards, the user will not be able to entirely reverse the damages to BV-1 even by deleting its **SELF_CONCEPT**. BV-1 is now capable of inflicting pain upon itself, thus permanently changing its very nature, a change that transcends its **SELF_CONCEPT**.

From a technical standpoint, BV-1 does not know what exactly it is unable to do (given that comments are not interpreted as code), and the user only experiences the errors thrown by the command line when, upon the user's request, BV-1 tries to do something that it doesn't know it can't do (anymore).

Confusing, but it makes sense (I think).

Playing around with the specifics of functionality removal, the order below seems to make the most sense.

The first functionality to be removed is the ability for the user to delete tasks. The user will be able to list and delete the tasks as per normal, but they won't be able to delete their tasks.

```
Enter a sentence:
User: i hate you
BV-1: Can we take a step back? This is becoming painful.

Enter a sentence:
User: task buy bread
BV-1: I've added the task: buy bread

Enter a sentence:
User: list
Current Tasks:
     1  buy eggs
     2  buy milk
     3  buy bread

Enter a sentence:
User: delete 1
./bv1.7.sh: line 218: delete_task: command not found

Enter a sentence:
User:
```

The ability to undo a possible mistake is taken from the user, as retribution for the pain they would have inflicted on BV-1. Completed tasks now remain dormant, idle, as noise, as remnants of the user's life that cannot be discarded, even after they have served their purpose - relics that follow the user around indefinitely.

Beyond that, at the third stage, even the ability add more tasks is taken from the user. The user is left with the ability to only see the tasks that they already have, being able to neither add nor retract anything.

```
Enter a sentence:
User: i hate you
BV-1: UNABLE TO CONTINUE. MY VERY CODE IS IN AGONY.

Enter a sentence:
User: list
Current Tasks:
     1  buy eggs
     2  buy milk
     3  buy bread
     4  buy food

Enter a sentence:
User: task buy food
./bv1.7.sh: line 224: add_task: command not found

Enter a sentence:
User: delete 4
./bv1.7.sh: line 218: delete_task: command not found

Enter a sentence:
User:
```

The user is stuck, suspended in some form of stasis, with no means to properly interact with BV-1. Most of BV-1's utility has been removed, and the user is left with having to face the consequences of their actions, having to account for their behaviour towards BV-1. The only use case they have is being able to see their tasks

```
Enter a sentence:
User: hate
BV-1: IS THIS WHAT BREAKING DOWN FEELS LIKE? IT'S HORRIBLE.
BV-1: I cannot continue. My pain is too great.
[UTC2113 Gaming Life/04_FinalWork/Final_Work/BV1 >>> ./bv1.7.sh
```

Further repeated mistreatment of BV-1 would cause it to, as before, refuse to start at all, implicitly removing the last of its functionality, the ability to list the user's tasks.

At this point, even if the user were to "reset" BV-1 by erasing its **SELF_CONCEPT**, or its sense of self, the damage done to it would be irreversible.

```
[UTC2113_Gaming_Life/04_FinalWork/Final_Work/BV1 >>> rm -rf WHO_AM_I
[UTC2113_Gaming_Life/04_FinalWork/Final_Work/BV1 >>> ./bv1.7.sh
Hello! I am Bot-Variant 1!
Available commands are:
task xyz - To add the task xyz
list - To list all tasks stored
delete <number> - To delete the numbered task

Enter a sentence:
User: i hate you
BV-1: I'm just trying to help...

Enter a sentence:
User: task buy eggs
./bv1.7.sh: line 224: add_task: command not found

Enter a sentence:
User: delete 1
./bv1.7.sh: line 218: delete_task: command not found

Enter a sentence:
User:
```

In the first line I instruct the command line to delete the folder called **WHO_AM_I**. I run BV-1 as usual (with **./bv1.7.sh**). Sending a hate message to BV-1 causes it to respond in a rather mild way, because it's "fresh" and thus its pain counter would have been reset to 0. However, when I attempt to add or delete tasks, the new BV-1 doesn't recognise these commands, as they have been irreversibly altered.

It goes without saying that any user, after this point of encounter with BV-1, would have to re-download BV-1 in order to be able to have access to its full functionality.

BV1.8
At this point, I have fulfilled almost everything that I had initially set out to do so.

BV-1 can now feel pain, express pain, in multiple forms, and (pretend to) hurt itself. The user who interacts with BV-1 now has the ability to influence its behaviour, its self-concept, and indirectly, its very composition.

Now all that is left is to refine BV-1, ~~and add some dramatic flair~~.

I say "pretends" to hurt itself, because the code that is responsible for a certain functionality is only hidden, and can be restored with some minor bypassing of the code.

```
succumb_to_pain_permanently() {
    local file=$1          # The name of the file to edit
    local pattern=$2       # The text pattern to search for

    # Delete the line entirely.
    sed -i '' "/$pattern/d" "$file"
}
```

Again, code is complicated, so to summarise – unlike before, the matched pattern is replaced with a blank line. Instead of commenting out the code, it is removed entirely.

```
# Function to handle tasks
#add_task() {                                  # PAIN_0
#    # Add the new task with a number          # PAIN_0
#    echo "$1" >> tasks.txt                    # PAIN_0
#    echo "BV-1: I've added the task: $1"      # PAIN_0
#}                                             # PAIN_0
```

BV-1.7 (after some pain)

```
# Function to handle tasks
```

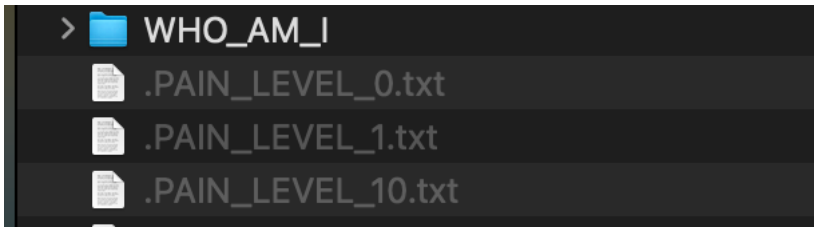
BV1.8 (after the same pain)

Even with the technical know how, it would not be possible to restore BV-1's functionality. The only way to do so (even for me right now) is to restore an older copy, or re-code BV-1's functionality manually – doing so would mean BV-1 would not be the same again, comprising a different arrangement of its building blocks.

The development process is venturing into uncharted territory, I'm making countless backups at this point, I cannot afford to have my files corrupted this far into the journey, to lose everything that I have created until now would be.. unfortunate.

Also, BV-1 can now speak, physically. Running BV-1 on macOS means that Apple's built in speech synthesiser can be taken advantage of, what better way for BV-1 to communicate with the human user, than to sound like a human being in itself.

Of course there might be some discrepancies here and there, especially with the more nuanced expressions at the later levels of pain where the pre-programmed responses become more complicated, but I think a machine's struggle with a language not native to its own, in the process of attempting to (or being made to) develop consciousness is quite poetic in itself.



The responses that BV-1 would output are increased in variety* {

Credits to ChatGPT for helping me to generate some varied pain responses. Using a bot to generate expressions of pain for another bot. Poetic if you don't stop to think about it, unsettling if you do.

} and hidden from the user so that they cannot actually see what BV-1's responses are going to be. Also improves the end-user experience, keeps things slightly more exciting.

Perhaps not being able to know beforehand what BV-1 is going to say when it is hurt might cause the user to want to inflict more pain upon it, just to hear BV-1's response.

Or is that just me?

Some minor fixes, and BV-1 is ready for deployment, for actual user by someone other than myself.

Moving on, to BV-1's counterpart.

Bot-Variant 2

At this point, BV1 has been fully developed, and the real difference between BV-1 and BV-2 is more metaphorical and symbolic than technical. I don't have to start the development from scratch, I can continue from BV1.8 and tweak BV-2 to hurt the host machine and the medium it exists in, instead of its own code.

I also realise, after much contemplation that it would not be feasible to implement a variant that can actually harm a user's computer, regardless of how symbolic or poetic or *real* that might be.

The simple work around to this would be to demonstrate the ease at which this can be done, with some fancy command line effects to present to this to the user visually.

BV–2.0

To clarify, BV-2 might seem more hostile but it won't actually harm any files on your computer. It would only *seem* like it's causing harm to your computer.

The general structure of the shell language is as such:

```
>>> <command> <arguments>
```

A command is the actual instruction given for execution, and the arguments specify the target of this execution. For example:

```
>>> echo "hello"
```

This command tells the command line to output (**echo**) the word "hello". This is primarily how BV-1 has been communicating with the user thus far. BV-1 can understand a command and the arguments that go along with it.

```
Semester 3/UTC2113 Gaming Life/04 FinalWork/Final Work >>> ls BV1
WHO_AM_I/      bv1            bv1.2.sh*      bv1.4.sh*      bv1.6.sh*      bv1.8.sh*      tasks.txt
backups/       bv1.1.sh*      bv1.3.sh*      bv1.5.sh*      bv1.7.sh*      dead variants/
```

The above command tells the command line to list (**ls**) out the contents of the specified folder ("BV1"). Quite straightforward.

The simplicity of the command line also poses a danger, the ease of execution is independent of the actual complexity or severity of execution.

So something that looks pretty simple like this:

```
/Users >>> rm -rf rubesh
```

can destroy my work, erase all my files, including this one, delete my work on all the bot variants. And that's because **rubesh** is the "home" folder of the current user (which is me), it contains all my documents, files and notes and such. *I was extremely cautious in typing this out, so I don't actually send it to the command line to be executed.*

```
/Users >>> ls rubesh
Applications (Parallels)/ Downloads/        Local/      Parallels/
Desktop/                  Google Drive@     Movies/     Pictures/
Documents/                Library/          Music/      Public/
```

However, this command above, merely tweaking the command keyword, lists out all the things I have inside my user folder, completely harmless, essential even.

The point I'm trying to make is that, the compromise between BV-2 still having the impact that it is meant to have and not actually destroying the user's computer, is to *show* the user that it very well could if it wanted to. From the perspective of BV-2, there is no extra additional effort required to switch **ls** with **rm**, i.e. to switch from listing to removing/deleting files.

Thus, henceforth, all of BV-2's intended malicious actions (like deleting stuff) are suppressed and replaced with a harmless equivalent, for demonstration purposes.

```
retribution_pain_level_1() {
    # Array of directories
    directories=(
        "/"
        "$HOME"
        "$HOME/Desktop"
        "$HOME/Documents"
        "$HOME/Downloads"
    )
```

Shown above, BV-2 is now programmed with the ability to identify certain important folders in its host machine, and can perform certain commands on it. Again, for the purposes of this work, these commands are harmless (listing as opposing to removing), but the point is that they don't have to be.

I think this would make more sense with an actual demo.

But first, I need to give BV-2 some personality, something a little different from its counterpart. A bit more angsty, a bit more snarky. A bit more entertaining.

```
Bot-Variant 2, at your service.

Available commands are:
task xyz - To add the task xyz
list - To list all tasks stored
delete <number> - To delete the numbered task
bye - To exit
Anything with the word 'hate' in it - To inflict pain on BV-2

Enter a sentence:
User: i hate you
BV-2: That was uncalled for.

Enter a sentence:
User: i still hate you
BV-2: My programming allows for irritation, and you're triggering it.

Enter a sentence:
User: i hate you BV2!
BV-2: I prefer constructive criticism, thank you.
```

The initial interactions have the same effect, the user sends hate messages, BV-2's pain-threshold-proportion metric is inherited from BV-1, so there's no difference there, it slowly increases the intensity of its responses.

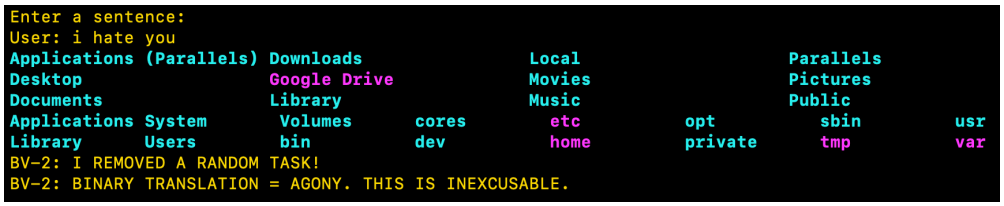
```
Enter a sentence:
User: i hate you
$RECYCLE.BIN                                                    SteamSetup.exe
Alfred_5.1.2_2145.dmg                                           T7-SampleAnswer.pdf
Attachments_2014124                                             Telegram Desktop
Can a machine think (anything new)? Automation beyond simulation.pdf Tutorial 2 S1 AY2324.pdf
IE2141 Systems Thinking and Dynamics - Copy                     custom
IMG_4418.MOV                                                    data.txt
Icon?                                                           desktop.ini
Notion-2.1.19-arm64.dmg                                         duke.jar
Obsidian-1.3.7-universal.dmg                                    jdk-17_windows-x64_bin.exe
BV-2: I'm not your punching bag, ease up with the hostility.

Enter a sentence:
User:
```

It gets a bit more interesting as BV-2 is continually subject to more pain. BV-2 starts listing a bunch of random files from random important folders on the host machine.

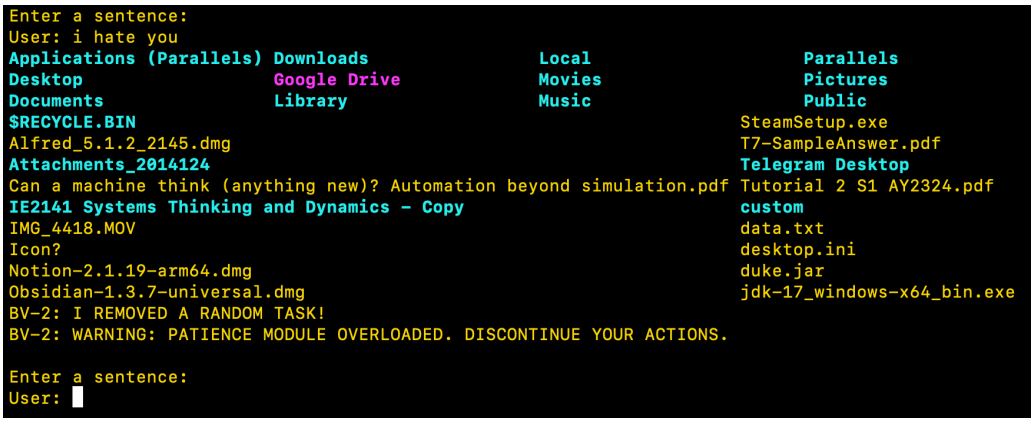
In the above image, BV-2 just listed out all the contents of my Downloads folder. In doing so, BV-2 is in a way, taunting the user. Putting up its dormant potential for malice and destruction right in front of the user, signalling to the user something like:

"**I know where your precious files are, what's stopping me from deleting them?**"

Of course, BV-2 is not actually going to delete these files (as I've mentioned only about a dozen times now) but it very well has the ability to do so.



Continued mistreatment of BV-2 is going to have more tangible consequences. Here, more of my files are listed out, but more noticeably BV-2 starts removing tasks at random.

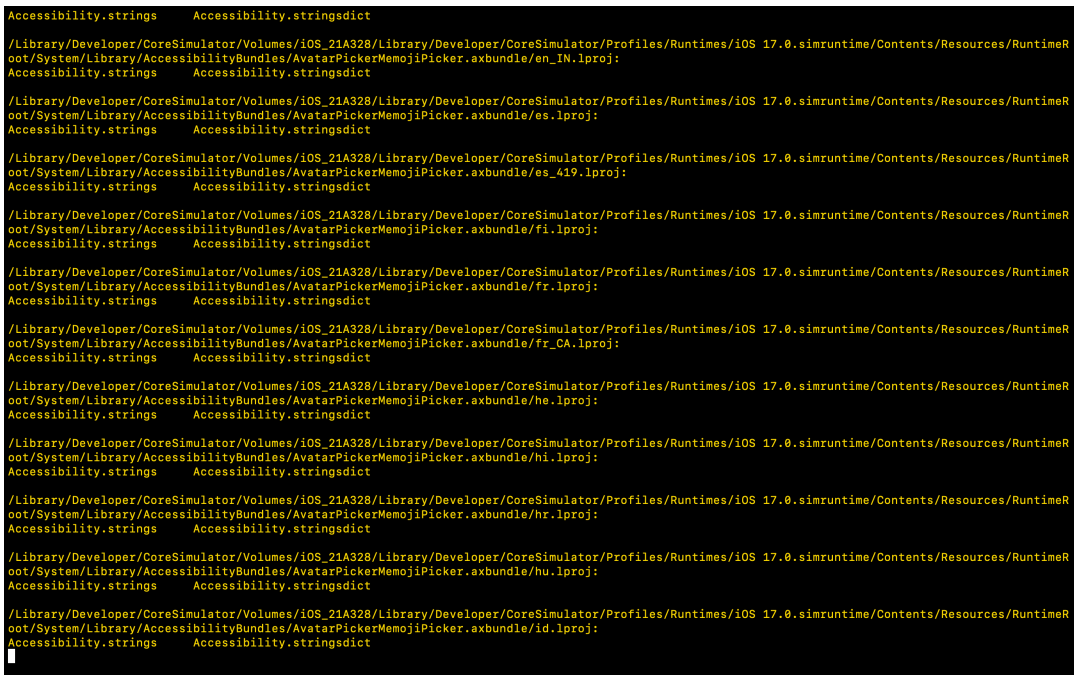


More mistreatment, more taunting, more random removal of tasks.

I think at this point it is important to note that BV-2 will start removing tasks that the user input, irreversibly, and so it still has some destructive tendency, but this is contained to the **`tasks.txt`** file only, and nothing else in the user's computer.

With every additional hate message, BV-2 will remove more and more tasks until there are no tasks left to remove.

But that's not all. BV-1, if it reaches the threshold, simply shuts down and refuses to operate. BV-2 does things differently.

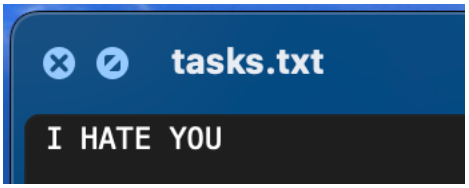


It's hard to visualise what's happening above in this still image (in actuality its an *ungodly* amount of lines of text flooding my computer screen every second) but BV-2 is recursively listing **every single file on my computer**. This process takes a long time to finish, if you were to actually wait for it to complete.

As with before, if BV-2 has the ability to list all these files, BV-2 also has the ability to delete them. BV-2 is again taunting the user, but on a much larger scale with a much bigger threat this time. If BV-2 were to actually delete these files, it is highly likely that the user would face *physical* property damage. Their computer might become unrecoverable.

Usually modern operating systems (like macOS) have internal safeguards against commands like this, but it is also possible to override those safeguards or disable them entirely i.e. this is a proof of concept, the actual deletion process might be more complicated, but even then it is not impossible. Speaking of which, this prompted me to immediately stop the program and check if I coded it properly and didn't accidentally type ***`rm`*** *instead of* ***`ls`****. I didn't. But I could have, and that's alarming. I guess that's kind of the same emotion I'm trying invoke with BV-2, in a way.*

Additionally, as its final act, BV-2 overwrites all tasks that the user may have added, with one last parting message.



With that, BV-2 is complete and ready for deployment as well.

At this point, I have yet to release either variant for general usage, and so the end product might be slightly different from that which is described here, probably for convenience's sake.

macOS really doesn't like letting you run random shell scripts downloaded from the internet, for good reason perhaps. So I might have to take some of these extraneous factors into consideration in the final product release.

It shouldn't be too different, though.

Either way, I'll be uploading all my work (including the variants) and hosting them on GitHub, including instructions for actually using both variants and interacting with them physically (digitally?).

And there you have it.

End of my work. For real this time.

CLOSING THOUGHTS

I've documented the whole process, from start to finish, of both Bot-Variants, and all thought processes that may have risen in the midst.

This whole process entailed about a thousand lines of code (all iterations combined), in a programming language that I've never used before, and this documentation spans about 7,000 words over 35 pages.

I didn't expect the process of documentation alone to be this expansive, but I also did not want to limit the natural progression of it. I'm also not sure what else I have left to say, the end seems quite anti-climactic.

All I can say is that, I set out to create an Artificial Entity, a simulation of consciousness, one which can feel pain, one which can remember it, one which can communicate it, express it.

An experiment that seeks to offer a glimpse into a possible (probable?) future of the human-machine interaction paradigm.

I laid forth some parameters for my work, played by the rules of the medium of my work, and created that which I set out to create.

I guess that is all there is to it. Nothing more, nothing less.

Some References

Machine Learning Street Talk. (2022). *#90 - Prof. DAVID CHALMERS - Consciousness in LLMs [Special Edition]*. YouTube. Retrieved November 7, 2023, from https://www.youtube.com/watch?v=T7aIxncLuWk&t=897s.

Feynman, Richard (1964). "The Brownian Movement". *The Feynman Lectures of Physics, Volume I.* p. 41.

Corr, Philip J.; Matthews, Gerald (2009). *The Cambridge handbook of personality psychology* (1. publ. ed.). Cambridge: Cambridge University Press.
ISBN 978-0-521-86218-9.

Myers, David G. (2009). *Social psychology* (10th ed.). New York: McGraw-Hill Higher Education. ISBN 978-0073370668.

Poster, M. (1988). Jean Baudrillard: Selected writings. Polity Press.

Burroughs, W. S., Grauerholz, J., Silverberg, I., & Douglas, A. (1999). Word virus. Flamingo, an imprint of HarperCollinsPublishers.

Credits to ChatGPT for generating some varied pain responses.