Programming Proletarian Literature: Kobayashi Takiji’s "Kani Kôsen" and Gaming as Reading

Jacob Philip Fisher
Bard College

Programming Proletarian Literature: Kobayashi Takiji’s *Kani Kôsen* and Gaming as Reading

Senior Project Submitted to
The Divisions of Science and Languages and Literature
of Bard College

by
Jacob Fisher

Annandale-on-Hudson, New York
May 2019

Acknowledgements

Many thanks to Nathan Shockey, Keith O’Hara, Mika Endo, and Scott Mehl for your encouragement and much more. Thanks to my family and friends for your love and support. Pages of thanks could be written, but if I’m thankful for you, you know who you are. <3

Table of Contents

Abstract

This project translates a novel, Kobayashi Takiji's, *Kani Kôsen* (*The Crab Cannery Ship*, 1929) into a video game. As a joint project between Computer Science and Japanese, its focus is to develop a game for the original Game Boy (1989) narratively based on a work of Japanese proletarian literature. Specific tools used in development were the Game Boy emulator: bgb, the Game Boy Developers Kit (gbdk), the Game Boy CPU manual, as well as a foundation in the C programming language, and some lower level systems experience. Being based on a novel, the play style utilizes text heavily, and is a nod to earlier Role-Playing Games (RPG's), such as *Final Fantasy* and *Dragon Quest*. Through the changing of medium from text-based novel, to interactive role-playing game, or more distinctly, from book to gaming console, several acts of translation are undergone. Many of the theoretical questions with which this project revolves involve the ways in which the changing of medium effects a piece of media. In particular, the transition from a work of literature to a work of electronic media has been theorized as transforming a work's fundamental message.

Introduction

A quick riddle: It weighs about 400 grams and is about 15 centimeters in length. It is a displayer of text, a conveyor of images, and a teller of stories. What could it be? Is it a video tape? A pocket-sized book? A couple more hints may help: it is existing within a plastic shell, has an 8-bit processor running at 4.19 MHz with 8 kilobytes (kB) of internal RAM and 8 kB of dedicated video RAM. Its controls consist of an 8-direction joypad, A-button, B-button, as well as a start and select button. It displays visuals via an LCD screen with a 160 by 144-pixel resolution with a refresh rate of about 60 frames per second, and emits audio via a mono audio speaker or stereo headphone jack. Certainly not a video tape, or a book, it is the Game Boy. Although these physical descriptors may have been the hints necessary to derive the riddle's answer, they merely describe the Game Boy's physical representation in space. To describe what the Game Boy actually is would be to stay closer to the original definition: a displayer of text, a conveyor of images, and a teller of stories. The Game Boy is not just a collection of hardware, but is a medium by which stories can be told in the 20th and 21st centuries and beyond.

In 1929, the Japanese author of proletarian literature Kobayashi Takiji published his masterpiece *The Crab Cannery Ship* (*Kani Kôsen*, 1929) in the era's popular proletariat magazine, *The Battleflag* (*Senki*). The novel, based on actual events that had taken place in 1926, tells the collective story of the workers aboard a factory ship on the Sea of Okhotsk at the height of Russian-Japanese conflict in 1929. The cruel conditions endured by the workers are a product of the ambiguity of the setting. As neither a ship nor a factory, this place of work was exempt from maritime law. The novel outlines the steps taken towards organization, and successful revolution within the insular factory ship community. As literature, *The Crab Cannery Ship* represents the possibility and hope of the working classes that they too could be liberated from the oppression of the bourgeoisie. With the fall of this hope and the rise of right-wing fascism the story of *The Crab Cannery Ship* was nearly forgotten. It was not until 1991, with the bursting of the economic bubble and congruently worsening work conditions that unrest created an environment in which Kobayashi Takiji's *The Crab Cannery Ship* would appear relevant once again resulting in its republication. The resurgence in the novel's popularity reached its peak in 2008. Several manga adaptations, as well as two film adaptations have come from Kobayashi's *The Crab Cannery Ship*. It is through the recreation and translation of the novel to a new and more modern form of medium that one hopes to keep its original message alive.

Even with the many adaptations already made of *The Crab Cannery Ship* there remained the sense that more could be done to convey Kobayashi's original message. It is hypothesized by the media theorist Marshall McLuhan that through the changing of a medium the entire message is altered. His conjecture is not concerned with content, but rather with the differences between the messages of the medium itself. Six decades after *The Crab Cannery Ship*'s publicatoin Nintendo released the Game Boy. The project that is the focus of this essay synthesizes a work of Japanese Proletarian Literature with the world of Computer Science, or more precisely, programming for the Game Boy. On the surface, the task is to retell the original story of *The Crab Cannery Ship* through the medium of the videogame. From a general literary angle, this task is cause for inquiry as to what occurs in the translation of a story across mediums. Further, it concerns itself with the implications of translating a piece of literature into a videogame. Specifically, it asks what is gained and what is lost in the transference from the novel *The Crab Cannery Ship* to the videogame. From a more technical, or Computer Science, point of view the primary question is raised as to the practical use of revisiting an outdated technology from 1989.

To someone with little or no programming experience, it may seem that to single handedly design and build a functioning game is a large enough feat on its own, but what is learned from this experience is much less objective than the game's functionality. Rather, what is truly taken from the project as a whole is the cultivation of the skill to rethink and re approach solutions to questions that may very well be unanswerable. In full, through the making of a game based on Kobayashi Takiji's, *The Crab Cannery Ship*, a reworking of the material is undergoing, in which the quality of the final product is subjectively dependent on one's understanding of the original work: *The Crab Cannery Ship*; and understanding of the medium into which it is being adapted: a videogame for the Game Boy. This essay consists of the following sections:

"Historical Context of the Novel, *The Crab Cannery Ship* (*Kani Kôsen*)," which supplies the reader with the necessary backdrop with which to frame the novel itself. The novel is then explored in the following section titled "Literary Analysis – Reading *The Crab Cannery Ship*." It is at that point that a theoretical case is made underlying the videogame, and Game Boy as a medium. The following two sections are concerned with the more technical aspects of this work. Consisting of a "Historical Context of the GameBoy," and an "Literary Analysis of the Game – Code as Prose."

Historical Context of the Novel, *The Crab Cannery Ship* (*Kani Kôsen*)

The content of this novel exists within the context of Kobayashi Takiji's life, and it is from this angle that we can understand the origins of his novel *The Crab Cannery Ship* (*Kani Kôsen*, 1929). Takiji was born on October 13, 1903, to a small landowning family of independent farmers the year before the start of the Russo-Japanese War. Due to his uncle Keigi's failing investment in his baking business, at the age of four Takiji moved to Otaru, Hokkaido. It was there that his family earned their living working for Keigi. It was due to Takiji's income from working at his uncle's bakery that he was able to get his education at the Hokkaido Otaru Commercial High School, and later, at the Otaru Commercial School of Higher Learning. While in school, Takiji gained his interest in literature and the arts, and graduated fifth in his class. This placed him in the interesting position of living both within the worlds of the proletariat and the educated elite. Soon after his graduation he took a position at the Hokkaido Colonial Bank. His interests in literature and the arts maintained though, and he continued to publish.

Noteworthy is the influence of the earlier proletarian fiction author Hayama Yoshiki on Kobayashi Takiji. Many parallels can be drawn between the motifs, themes, and settings between the two authors' work. Takiji's appreciation of Hayama's work has been said to stem from when he read Hayama's collection of short stories, "The Prostitute" (Inbaifu, 1925). Regarding this text, Takiji wrote in his journal, "The meaning of 'The Prostitute' came to me suddenly with a bang. Literally with a bang." Not long after the publication of Hayama's text, Kobayashi Takiji published what came to be known as "The Takiko Stories." These were informed by his falling in love with a young sex worker, Taguchi Takiko. Komori Yôichi writes in his introduction to the 2013 translation of *The Crab Cannery Ship*:

> The experience of falling in love with a woman from the lowest stratum of capitalist society's class system – a woman who had been driven to commodifying her own sexuality – while working in a bank at the center of that system, imposed manifold disjunctions on Takiji. It also provided the germ of what would grow to become the core of his literature.

At this time (1927) there were many strikes held by tenancy farmers over low wages and rent with which Kobayashi Takiji was involved, helping to write publicity leaflets on his way home from work at the bank. The same year Takiji became executive secretary of the Otaru branch of the Worker and Farmer Artist Federation. The strikes that ensued were victorious, and resulted in an arbitration exceeding the tenant farmers' original demands for "reduction or exemption from tenancy rents." These experiences informed his novella, "The Absentee Landlord" (*Fuzai Jinushi*, 1930), published in the *Central Review* (*Chûô Kôron*), as Takiji became renowned as a writer of proletarian literature.

In the February General Election of 1928, the proletarian parties won 8 seats out of 466 in the National Diet. Mass arrests were carried out on March 15th of those affiliated with the Worker and Farmer Artist Federation in response. More than 1,600 arrests were made in Tokyo, Osaka, and Hokkaido. Kobayashi Takiji's experience of this event led to his publication of the short story "The March 15th Incident." Regardless of the attempted banning of this publication in the proletariat magazine, *The Battle Flag* (*Senki*), it managed to circulate, and brought Takiji acclaim and notoriety.

Kobayashi Takiji followed this novella with his masterpiece, *The Crab Cannery Ship*, in the May and June issues of *The Battle Flag*. It was 1929, and Takiji had just been elected as a member of the central committee of the Japanese Proletarian Writers League. *The Crab Cannery* ship was based on actual events that took place in 1926, but what stands out most prominently is the novel's relation to a work with a similar setting written three years before by Hayama Yoshiki titled *Life on the Sea* (*Umi ni Ikiru Hitobito*, 1926). It is not simply that Kobayashi Takiji was inspired by Hayama Yoshiki – in the two novels there are clear similarities in theme, and plot development.

Heather Bowen-Struyk's, "Rethinking Japanese Proletarian Literature" describes Japanese works taking place on ships as "the boat book genre." The importance of Kobayashi's use of this genre, and its thematic parallels to Hayama's work, *Life on the Sea*, is significant in its ability to "establish a lineage of great proletarian boat books in Japan." Takiji makes a dialogue between himself and the pre-existing proletarian literature in Japan like Hayama's. In particular, "It is significant that Kobayashi posits Hayama as his literary progenitor because this affects Kobayashi's ability to position himself in the history of proletarian literature."

The Crab Cannery Ship was quickly followed up by the publication of "The Absentee Landlord." In this story, Takiji criticizes the system of landlord-tenant farming in Hokkaido, and references his bank of employment by name, and as a result he was fired in 1929. At this point Kobayashi Takiji moved to Tokyo, joined the Communist Party, and became immersed in his work in the world of proletarian literature and activism. Only a few years later he was captured, and tortured to death on February 20, 1933. To this day, the name Kobayashi Takiji is perhaps the most recognizable name in a large list of great Japanese writers by most of the Japanese population. In Norma Field's research on Japanese proletarian literature, and Kobayashi Takiji, she states that though she "ha[s] stayed at length in Otaru, the port city in Hokkaido, [...] where this writer grew up. Even there, where most people had at least heard his name," upon mention of her research she "was greeted with surprise." This lack of interest in Takiji makes it all the more surprising that this novel, written in 1929, regained notoriety in Japan in 2008. It is, however, as relevant to the working classes in Japan and abroad today, as it was in the 1920's. In brief, the rise of the "precariat" (a word consisting combining the italian graffito, "precario," and "proletariat"), and the threat of repeal of Article 9, which renounces the right to war, of Japan's constitution, it is no wonder that the phrase "kani kôsen" became an expression that drew together terms like "working poor," "lost generation," and "income gap society." Kobayashi Takiji's work demonstrates that the fight against imperialism is a fight done in solidarity, as well as the notion that this fight is not against their immediate managers and bosses, but the imperialistic structure that imposes itself on them in the first place. Establishing the relevant content of Kobayashi Takiji's life provides context of his novel, *The Crab Cannery Ship*, both at the time of its writing, and at the present. Next, we turn to the parallels within the boat book genre, as well as the unique and powerful differences that Takiji utilizes that make *The Crab Cannery Ship* the masterpiece of Japanese proletarian literature.

Literary Analysis of the Novel – Reading *The Crab Cannery Ship*

Komori Yôichi reminds us about a letter to proletarian literary critic Kurahara Korehito, in which Takiji elucidates seven points concerning the novel's "intent." These can be paraphrased as follows: First, there is no singular protagonist, but rather the novel centers around a group. This group being the workers on the ship. Second, there is "no depiction of individual personality or psychology." Third, many efforts were made towards popularizing proletarian literature. It was the case at this time that proletarian literature prior to this point was "superficial[ly] intellectual." On the other hand, *The Crab Cannery Ship* was said to be "overwhelmingly worker like." Fourth, its setting upon a crab cannery ship is meant to be a "unique form of labor." Fifth, the workers with which the novel dealt with were "unorganized." Sixth, it portrayed the ways in which capitalism, while "seeking to keep the workers unorganized," was (ironically) simultaneously "causing the workers to organize." Seventh, despite the emphasis on "the proletariat [...] unconditionally oppos[ing] imperialist wars," numerous workers aboard the ship did not understand the basis for this.

What stands out are the implications of these themes for a novel. Given these themes, what is Kobayashi Takiji trying to say? These core seven notions are founded on Marxist, or Communist philosophy. Bowen-Struyk points out two main meanings of Communism in the 1920s and 1930s in Japan: first, Communism could be understood as "a political philosophy envisioning a utopian communistic organization of society," and second, it was thought of as "referr[ing] to the policies and directives of the Soviet Union's Communist International, or Comintern, and to the policies and directives of the individual national incarnations of the Communist Party." Marxist ideology shines through in the use of a group protagonist as opposed to an individual. This is then emphasized in the way that the role of “leader” is fulfilled by a group personality, rather than any singular psychology taking center stage.

To return to Takiji’s first point, the lack of a singular protagonist and use of a group as the central figure is shown in two ways: the complete lack of formal names, and the non-human descriptions of the fishermen and other members of the crew. It is clear at first that there is an immediate contradiction. There are only two main characters referred to by their proper names: Asakawa, the manager, and Miyaguchi, the weakest and most tormented by Asakawa. In this case, proper names are given only to the strongest and the weakest, in which the strongest commits brutal acts of violence on the weak. The other recurring characters are only titled by their actions, such as the “Stuttering Fisherman,” or the “‘Better not act so big,’ Fisherman.” The dehumanizing conditions, and treatment of the workers is highlighted early on in the novel when an S.O.S. is picked up by another ship. Not wanting to waste the resources of going off track and aiding them, Asakawa made the executive decision to let the ship sink. When the fishermen hear of this, one asks “‘What the hell does he think human lives are, anyhow?’” to which the reply is, “‘But Asakawa does not think of you fellows as human beings.” The fisherman is said to have “tried to respond but could only stutter.” Komori Yôichi pronounces that this moment of having been stripped of all dignity physically manifests itself in the fisherman as a disorder that strips him of his human ability to speak.

This point is then followed by Takiji’s attempt to popularize proletarian literature through *The Crab Cannery Ship*, which raises questions as to how the novel relates to the Communist ideologies behind it. Through the history of proletarian literature up to the publication of *The Crab Cannery Ship* the legitimacy of these works as pieces of literature rather than propaganda was a heated debate. It is evident through the rough, “worker-like,” language of the book that

Takiji is trying to make the novel more approachable to the community it was about. There were instances where works of proletarian literature were given the recognition they deserved, in Kurahara Korehito's commentary, "Kaisetsu," he points out in regards to Hayama's works that, "[e]ven the *bundan* (literary world), who had thus far rejected proletarian literature as inartistic, could not ignore these works." Of course, it is with Kobayashi Takiji's predecessor, Hayama Yoshiki, that these works began to be seen as artistic.

It follows then that one of the foundational pieces of the novel is its setting, a crab canning ship – this being the fourth point of Kobayashi Takiji's "intent." With this, Kobayashi points out the uniqueness of the labor as well. On the surface the author, Nakano Shigeharu points out the importance of utilizing workers on a boat as unique, in that ships feature "intellectual" workers – intellectual because as sailors, they know math well and their world is as large as the world, in contrast to industrial laborers whose world is the size of the lunchroom. Beyond the workers, Bowen-Struyk breaks the significance of this setting into three parts: the labor of the workers aboard the ship is essential to the growth of capital; the lives of the subject matter (the workers) are harsh, and akin to that of a prisoner's; and finally, the insular environment of a ship in which a strike takes place is recognizable as a small-scale version of a class revolution. In other words, the setting of a crab cannery ship acts as a simulation of a larger scale revolution by the working class. Kobayashi Takiji's use of the crab cannery ship is notable in its exploitative setting. These crab cannery ships are considered "factory ships," which as such are not governed by maritime law. Specifically, "no other site offered such an accommodating setting for management's freedom to act with total impunity." It is this fact that made the crab cannery ship the perfect setting to display the cruel and ruthless working environment in Japan at that time.

We then look to the fifth basis of "intent" in the novel: the ways in which the workers were "unorganized" aboard the ship. This is shown in several ways, but the most apparent is the way in which the manager, Asakawa, attempts to pin the workers against each other. The tactic that Asakawa deploys most is a kind of twisted gamification of the worker's labor. The first incident of this is when the fisherman, Miyaguchi (as mentioned earlier), goes missing. In response Asakawa leaves a note to the workers:

> To all hands: anyone who finds Miyaguchi wins two packs of *Golden Bat* smokes and a hand towel.
>
> – Manager Asakawa

This offering of prizes and rewards to the workers in exchange for turning against their coworkers escalates into a more blatant act of competition, where "[w]henever the fishermen and workers 'lost' to the seamen by killing fewer crabs, they were made to feel that they had failed – although the work brought them no benefits at all," and when the workers grew tired "[t]he manager [...] began to hand out 'prizes' to the winning side," and when the workers grew tired of that "the manager put up a poster announcing that whoever accomplished the least amount of work would be branded with a red hot iron rod." It was this exact treatment that Kobayashi Takiji is talking about that "caused the workers to organize."

This is to say that by "organizing" the workers against each other in order to increase product output, management concurrently organized the workers together. This leads us to Takiji's sixth point of intention: that the novel portrayed the ways in which capitalism, while "seeking to keep the workers unorganized," was (ironically) simultaneously, "causing the workers to organize." This is to say that in the incidents mentioned above, in which Asakawa attempts to motivate and turn the workers against each other through competition, the true result was not a separation, but rather a growing solidarity, or uprising, against their overseer, and eventually, against the structure that put their overseer into power. This arises for the first time as a side effect to the harsh labor conditions Asakawa had made, and out of exhaustion one morning a character titled, "the miner," exclaims "'I'm going to slow down [...] I just can't keep this up.'" After being questioned the miner makes perfectly clear that his slowing down was not out of laziness, but out of physical necessity. It is because of this rationality that soon "it wasn't just one or two workers who were working slowly – nearly all of them were – [Asakawa] could do nothing but storm about and fume. [...] [T]he manager's club was of no use at all!" Through the unified use of these "slowdown tactics," the workers were able to gain control over their environment. The evidence of the effectiveness of this kind of group disobedience being the driving force to incite and encourage more organization on their behalf.

The seventh point, and perhaps the idea most distinct to Kobayashi Takiji's *The Crab Cannery Ship* is that despite the emphasis on "the proletariat [...] unconditionally oppos[ing] imperialist wars," numerous workers aboard the ship did not understand the basis for said opposition. In the midst of the workers' strikes and disobedience towards their immediate manager, there was still a great pride amongst the workers towards Japan as a nation, and consequently, Japan's acts of imperialism in Manchuria at that time. It is no coincidence that the crab canning ship is located precariously between Japanese and Russian waters, all the while guarded by an Imperial naval ship. Kobayashi employs this setting as a tactic for opening a dialogue, and raising questions about the validity (and ethics) of government intentions at this time. This mental disconnect on the workers behalf is most heavily displayed at the time of the first strike. During which the imperial destroyer approaches the ship. In excitement the striking workers shout "'Imperial navy, hurrah!'" The question is raised "'How the hell can a warship that belongs to the empire not be on the side of the people who belong to the same empire!?'"

This question reaches to the heart of the issue in the way it backhandedly points out the analogous nature of capitalism, imperialism and nationalism. Kobayashi Takiji is displaying the state of the world, in which the rich are able to fund the military, which is dependent on the nationalism of the working classes to fuel their agenda. This raises the question as to why Kobayashi Takiji found it important to display the workers' lack of understanding towards imperialist opposition.

As both Komori Yôichi and Heather Bowen-Struyk point out, what separates Kobayashi Takiji's, *The Crab Cannery Ship* from other works of Japanese proletarian literature is "its assessment of the interrelated nature of the systems of capitalism, imperialism and nationalism." Given the setting as a factory ship on the Sea of Okhotsk, crossing over into Russian waters at the peak of the Russo-Japanese war, while being protected (or monitored) by a naval warship, Takiji develops a clear backdrop to have a dialogue about "international, military, and economic relations." The significance of this dialogue is most clearly rooted in its conversation with Marx's, *Wage Labour and Capital*, in which Marx states that "[a]s long as the wage-worker is a wage-worker his lot depends upon capital," this being "the much-vaunted community of interest between worker and capitalist." This references a core ideology of the essay, which is that "the existence of a class which possesses nothing but its capacity to labor is a necessary prerequisite of capital." In essence, Marx points out the use of nationalism in maintaining the lower class as subservient to the bourgeoisie. In the scene previously mentioned in which many of the workers misunderstand the intention of the Imperial naval ship, Takiji illustrates the alignment of the bourgeoisie and imperial navy's interests. In this instance, nationalism is used in such a way that it maintains the worker's obedience to their masters and suppliers of capital.

At the time of *The Crab Cannery Ship*'s publication, there was a great risk in publishing this kind of content. Heather Bowen-Struyk spells out that "*The Factory Ship* [...] implicates the emperor and imperialism as well as nationalism in its critique of capitalism. To a Japan already gearing up for potential war in Manchuria, such accusations were undeniably dangerous." The level of censorship the novel faced at this time communicates the dangerous nature of its ideas. Once again, the novel is most clearly investigated in the ways in which it is comparable to its predecessor, Hayama Yoshiki's *Life on the Sea*, in which relatively minimal lines are obfuscated by *fuseji* (censorship marks) – X's, O's or ellipses in place of objectionable material. *The Crab Cannery Ship*, on the other hand, was heavily censored. The time of its publication 1929, a mere year after the mass arrests of March 15, 1928, the Japanese government was making clear its willingness to suppress radical opposition. What could be seen as somewhat surprising is that it was exactly this censorship that aided in the sales of *The Crab Cannery Ship*, and more so *The Battle Flag*. In Nathan Shockey's, *The Typographic Imagination: Reading and Writing in Japan's Age of Modern Print Media*, a "balance between censorship and commercialism" within publication of the magazine in the late twenties to the early thirties is pointed out. This can be seen in the ability of the "editors [...] to successfully commodify the very notion of suppression in order to increase media attention and circulation." In fact, it could be debated that the attention gained through the banning of eight issues resulted in "the tripling print runs to over 22,000 copies per month." In effect, the timing of *The Crab Cannery Ship* was historically with the rise of a proletarian popular culture. *The Crab Cannery Ship* lost its popularity with the de-popularization of proletarian culture and the rise of right-wing fascism, but miraculously, the novel rose back into the public gaze in the year 2008.

The years following the bursting of the economic bubble in Japan (1991) marked a time of uncertainty in economic growth. This uncertainty resulted in the rise of the concept of the "precariat," a term deriving from "precarious" and "proletariat." Amamiya Karin, a writer and well-known spokesperson for the precariat movement, talks about her personal experience as a "freeter" (*furītā*), and describes the era as a "warped picture: poorly paid freeter or regular full-time employee fated to die from overwork." Her younger brother, she says, "got a job at Yamada Denki, where he was forced to work eighteen hours a day." The parallels between the proletariat work conditions of 2008 and 1929 are overt. These parallels are cause for the rerelease and re-publish of *The Crab Cannery Ship*, for which Amamiya wrote the introduction. With the rerelease of the novel came a number of adaptations as well. At least five manga based on the novel have been published in the interest of attracting a younger audience, in addition to a documentary film, "Strike the Hour, Takiji," as well as a remake of the 1953 film. These works were necessary pieces of raising awareness and building new communities and circles for conversation over matters involving the modern oppressive work environment. Of all these adaptations there lacked a medium that, perhaps, could be fruitful: a videogame. As a medium, the videogame has its own unique immersive qualities that are worth being explored. So, for this project, one was made.

Making the Game – Adaptation as Reading

While developing a personal interpretation of *The Crab Cannery Ship*, reading and watching the manga and movies based on the novel was just as valuable as reading the novel itself. In particular, the distinct differences in the way that the three manga read is especially informative. It is apparent that each manga clearly has a unique audience in mind. This first manga, published in 2007 by East Press, was most directed towards a younger audience. This was most evident in its use of a central character, Morimoto, and his “sidekick,” Akiyuki. This clearly breaks Kobayashi’s motif of lacking a singular protagonist. The choice to follow a single character rather than putting the emphasis on a group was a creative liberty, and having done so, the manga stays more concretely in the genre of manga known as *shônen manga*. Shônen manga being targeted towards a young audience (specifically male), generally between the ages of 12 and 18. What is most skillful about this manga is its ability to maintain the expected tone of a piece of shônen manga, while still delivering the hard hits of the novel, yet sparing the details that may be considered inappropriate to a younger audience.

The version published by the Kobayashi Takiji Literature Museum and illustrated by Fujio Gô packed the most punch. Directly more towards an adult audience, this manga would be likely considered to be within the genre of *seinen manga*. The opening and closing scenes of this work depict a dead Kobayashi Takiji surrounded by those appearing to be allies and supporters of his work. They commemorate his memory with the telling of *The Crab Cannery Ship*, which then closes with the iconic lines, “Takiji! Won’t you rise up one more time!? For all of us…” (Takiji! Mô ichi do tatte misene ka! Minna no tame ni–). Though this manga also lacked some specific detail from the novel, it stays very close to Kobayashi Takiji’s original ideals behind the novel laid out above. The general goal of this work was unquestionably targeted towards educating a more adult audience of the history of *The Crab Cannery Ship* as well as the substance of the novel.

The 2008 manga by published by Shinchôsha and illustrated by Hara Kēichirô was the last manga adaptation that I read. Having took very few liberties, and staying true to almost all details of the original, it was surely the version that most followed the original story set forth by the novel. Interestingly enough, this does not necessarily mean that it best delivered the message best. When reading these adaptations, it becomes apparent how in many ways changing the medium of a work becomes an act of translation. Though these share them same fundamental narrative, the medium through which *The Crab Cannery Ship*'s content is delivered changes the intake of the material, and often the experience and emotions it is capable of evoking.

This brings into question if the medium has an impact on the message of the media? The media theorist Marshall McLuhan conjectures that the medium does not simply affect our understanding of the message, but rather, that "the medium is the message." This is the approach that has been considered most heavily throughout the making of the videogame adaptation of *The Crab Cannery Ship*, and it is through this inquiry regarding what it means to change mediums that has derived the most insightful progress in the midst of the project's development. To perhaps over simplify, when Marshall McLuhan states that "the medium is the message," he is not pointing to the aspects of the work being delivered via the medium, but the message being told by the actual medium itself. For example, the message given by print is different from the message given by a medium such as film. Through text one cannot simply make an entire landscape or setting appear instantaneously to the reader. In print, setting must always be developed meticulously in order to make the reader feel that they are truly immersed in the author's world. On the other hand, film can put its viewer into the setting in the blink of an eye. The message is less about the content in these examples, but about how the content is received. It is because of the unnecessary nature of content in the deliverance of this message that Marshall McLuhan goes so far as to say that light itself is a medium. He states:

> "[E]lectric light and power are separate from their uses, yet they eliminate time and space factors in human association exactly as do radio, telegraph, telephone, and TV, creating involvement in depth."

It is in this passage that McLuhan states that the change from a linear building of our involvement in a novel to the instantaneous involvement of electronic media creates an entirely new meaning for all works with which this distinction is made. Here, the juxtaposition of linearity and instantaneousness is made in relation to time. This is to say that the message delivered by media is altered depending on the time required for it to be consumed.

What then can be said of the videogame console as a medium? It too is electronic, and therefore capable of creating involvement, or immersion, in depth. It follows then that these separate mediums have separate ways of fostering such immersion. It can be seen that a film is to a novel what a videogame is to a “choose your own adventure” book. Much like in Wolfgang Petersen’s film, *The NeverEnding Story*, the content (or media) with which the consumer is interacting becomes personal by way of the strength of the interaction. This of course being the case most often in the genre of game known as “Role Playing Games” (RPGs). Most commonly associated with the Live Action Role Playing Game, *Dungeons and Dragons*, and in videogames, *Final Fantasy*. Much like in *The NeverEnding Story*, these games rely on the players’ personal investment in them. In *Final Fantasy Tactics A2: Grimoire of the Rift*, the opening tells the story of a boy staying after school on the last day before summer vacation. He notices a book on his teacher’s desk, and upon opening it, enters a new universe. This analogy is somewhat self-referential quality, in that the player in the real world is transported into a new world within the console where they are a character being transported into a new world within the game.

What are the distinct immersive elements of a game that separate it from other forms of media, and how do these differences change our consumption of the material? As a game, there is a play-element in the absorption of the media in that we play the game. It follows that what is inherently embedded in the act of play, will also be embedded in using, or playing on, the Game Boy as an electronic medium. To look deeper into this element of play, one can look to Johan Huizinga's work *Homo Ludens: A Study of the Play Element in Culture* (1938), which centers itself around studying play's many facets. Most commonly, play is understood as a learning tool. Johan Huizinga points out two aspects of play that are most applicable to the transformation of *The Crab Cannery Ship* into a game. These two aspects being the voluntary nature of play, as well as its aspect of being representative of something. The basis of play as voluntary action gives play a quality of freedom. This is striking in that this freedom can be understood in the way that it "leaves untouched the philosophical problem of determinism." In other words, the player is not being forced into play. One plays because they want to, not because they must.

This hardly seems important until play's representative quality is addressed. That is, in representation, we as players attempt to represent, or display ourselves as new characters in new roles under new contexts. These two points in conjunction are recognizably persuasive in that it puts the player in a position in which a role or situation is being "'staked out,' and moreover [...] in mirth and freedom." What this means is that through the act of play, one allows oneself to voluntarily take on the role of a character, and since it is play, it is done with free will. When partaking in an act of play these two aspects outlined above suggest that through the enactment of a character there is a certain investment in the game that takes place in which the player truly is, for a moment, that character or part. It is through this uniquely immersive aspect of play, which is embedded in games, that perhaps the user of said medium can voluntarily escape their own ideologies and beliefs in order to enter another's with enough open mindedness to empathize with the characters with which they are personating. *The Crab Cannery Ship* is a novel that calls upon the empathy of its readers, and it is because of videogame's potential to build empathy that the medium lends itself perfectly to the narrative.

Historical Context of the Game Boy

Developing a relatively simple game for the Game Boy turns out to be extraordinarily not simple. After a quick glance at the Game Boy CPU Manual it can be seen that for the speediest progress one would be best off writing the code in the C programming language through the use of the "Game Boy Developers Kit" (gbdk). Although the abstraction caused through the use of gbdk may enable a lot of progress, this progress can result in poor code that is hardly understandable as to how it is functioning. In order to gain a deeper understanding of the inner workings of gbdk, it was imperative to take a slight detour to try and rewrite some programs in the Game Boy's CPU assembly language, "z80." Though the final product is not written in z80 assembly it is worth noting that oftentimes in both the practice of programming and generally in the nature of programming languages layers of abstraction are made for the ease of understanding the function of the code. In essence, gbdk comes with nicely packaged pre-built functions like `joypad` which have a name that nicely describes their purpose, or what they are doing. Unfortunately, in doing so often times the how they are doing it can be lost. In this portion of the essay I'm going to attempt to create a layer of abstraction that delivers the message of what I've done in an approachable and relatively non-technical manner, while still maintaining a sense of how I've done it. Before delving right into the analysis of the mechanics of the game itself, it will be best to contextualize the Game Boy and its hardware historically, just as has already been established with the novel.

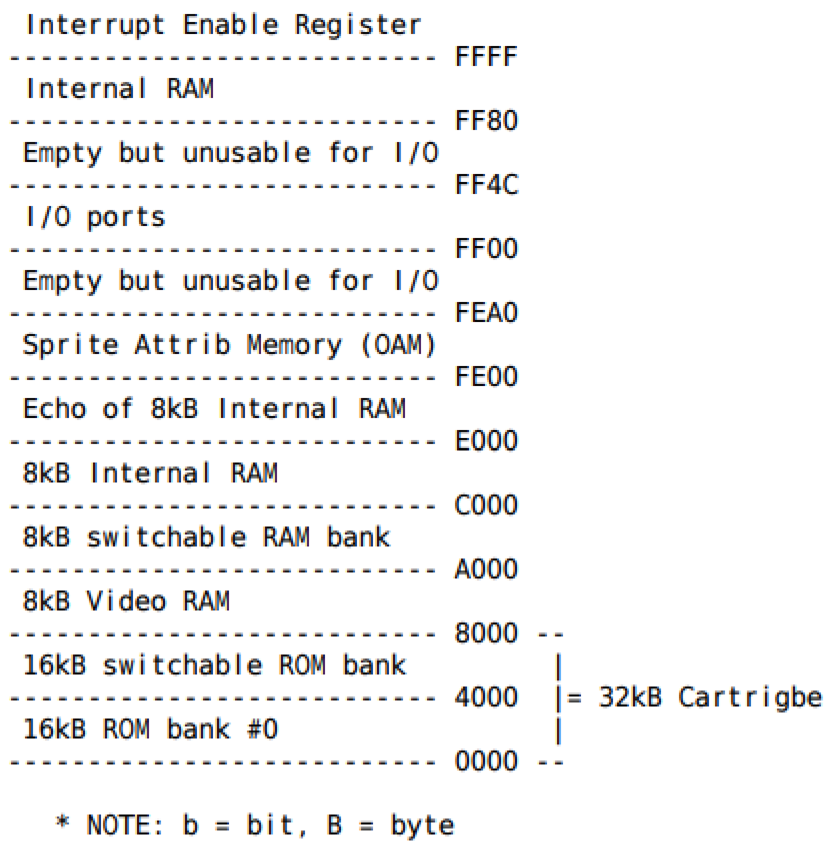
Following its predecessor, the Game & Watch, the Game Boy was released by Nintendo in 1989. The Game Boy differs in its ability to take cartridges of read only memory (ROM), which would allow the user to play a variety of games with a single console, much like Nintendo's earlier Nintendo Entertainment System (NES), except on the go. The Game Boy was not the first handheld console ever manufactured, but it was one of the most (if not the most) influential. This is seen clearly through its high sales: "[s]ince its introduction in 1989, Game Boy has sold well over 150 million systems worldwide." At the time of its release its competitors were Sega's Game Gear, Atari's Lynx, and NEC's TurboExpress. What distinguished the Game Boy was its durability and battery life, not to mention its choice of games (often sold with *Tetris* or *Super Mario Land* to name a few). The Game Boy line had many offspring, consisting of the Game Boy Color, Game Boy Pocket/Game Boy Light, and the Super Game Boy. The line was eventually discontinued with the advent of the Game Boy Advance in 2001.

When talking about the foundational ideology behind the Game Boy's technical design, Daniel Reynold's quotes Yokoi Gunpei, its lead developer, who describes it as "using 'withered technology' in new and novel articulations." In Japanese these is written, "*kareta gijutsu no suiheishikô*," which can be translated as "lateral thinking of withered technology." This design philosophy is apparent in many aspects of not only the Game Boy's aesthetic, but its actual hardware as well. It is evident that at the time of the Game Boy's release higher quality LCD screens were available as well as faster and generally higher quality central processing units (CPUs). These were not the things that Nintendo was going to use to gain people's interests though. Rather, Yokoi's approach was much more focused on the affordability of the console, and the fun games that it could run despite the hardware's limitations. Yokoi's mentality when it came to the limitations of the hardware was to face them head on, and to find creative solutions around them or to rework them advantageously.

Literary Analysis of the Game – Code as Prose

It is through this context that the element of programming for the Game Boy can be more clearly observed. At the very least, the temperament and mentality necessary for doing so has been established. The specific limitations of working with the Game Boy CPU is of the most relevance. The CPU is described as an Intel 8080 and Zilog z80 hybrid. Its instruction set is almost identical to the 8080 despite the exclusion of exchange instructions, and when compared to z80 there are some instructions that have been added and some have been removed. The specific differences on these limitations in relation to the z80 are outside the scope of this essay, and can be found in more depth online in the Game Boy CPU manual. This manual is not proprietary to Nintendo. Rather it is a compilation of research done by a number of programmers interested in writing games for the Game Boy. The Game Boy itself is most easily understood by looking at the memory map as shown below in **Figure 1**. Memory begins at address 0000 (which is actually in the cartridge) and ends at FFFF. For those unfamiliar with the hexadecimal system of counting, that’s from 0 to 65,535. Each one of these four character or integer strings is representative of two bytes. This means that two hexadecimal characters or integers are representative of a byte, and one hexadecimal characters or integers represent a nibble (four bits). In the case of the Game Boy’s memory going up to 65,535, this is translatable to approximately 65 kilobytes (henceforth denoted kB).

Figure 1:



```
 Interrupt Enable Register
--------------------------- FFFF
 Internal RAM
--------------------------- FF80
 Empty but unusable for I/O
--------------------------- FF4C
 I/O ports
--------------------------- FF00
 Empty but unusable for I/O
--------------------------- FEA0
 Sprite Attrib Memory (OAM)
--------------------------- FE00
 Echo of 8kB Internal RAM
--------------------------- E000
 8kB Internal RAM
--------------------------- C000
 8kB switchable RAM bank
--------------------------- A000
 8kB Video RAM
--------------------------- 8000 --
 16kB switchable ROM bank         |
--------------------------- 4000  |= 32kB Cartrigbe
 16kB ROM bank #0                 |
--------------------------- 0000 --

   * NOTE: b = bit, B = byte
```

It follows then that the first 32 kB, from 0000 to 8000, actually consist of the cartridge's memory. When programming for the Game Boy this is the memory that the developer is limited to. There are in fact variant cartridges that have been developed for the Game Boy. All cartridges have 32 kB of ROM (read only memory), but more advanced cartridges were later designed to have RAM (random access memory) as well as several switchable banks of extra memory in both ROM and RAM. That is why there is a section in the Game Boy's physical memory for 8kB of switchable RAM from A000 to C000. *The Crab Cannery Ship* game is limited to a 32 kB cartridge such as the one mapped out above. Specifically, 32 kB of ROM, nothing more and nothing less. *Tetris* is an example of a game that fits into one of these cartridges. If one was to download this paragraph as a pdf, it would take up more room than 32 kB. This is very little memory, and this is one of the many things that Yokoi referred to when he spoke of the Game Boy's technology as "withered."

Like an author, the programmer tells stories, the primary difference being a programmer's audience is a computer. Two authors can tell the same story, and though these two stories will share the same meaning, the journey of getting there will be different, and therefore the human experience of reading them will be different as well. For a human these differences generally manifest themselves as changes in emotional response. Two programmers can write the same program, and though these two programs will do the same things, the journey of getting there can also be different, and therefore the computer experience of reading them will be different as well. To the computer these differences can manifest themselves as changes in program speed or space used in memory. To the human this can also be a matter of aesthetics. If Faulkner was a programmer, he would write long drawn out and recursive obfuscated lines of JavaScript. Whereas Hemingway would write in the plainest of assembly. Taking each step towards a final product, one line at a time. Both of these styles are capable of being beautiful and creative and efficient or concise in their own ways, but the Game Boy tends to prefer Hemingway. That is, concise and efficient, or minimalist code. When dealing with only 32 kB of memory it's imperative that the code is as resourceful as possible. This is where the "lateral solutions" that Yokoi proposed come into play.

Depending on the genre of the game being developed, the problem sets confronted while programming can differ, and therefore the solutions end up differing as well. Given that *The Crab Cannery Ship* game is an RPG, or role-playing game, it requires many more functions for printing and displaying text. Whereas a less text heavy game such as *Mario*, which is more centered on the character's (or sprite's) interaction within its world, would require more functions to handle physics and movement within the game. There are also parallel issues of memory between all games, in that they all must deal with the limitations on Visual Memory. Visual Memory in the Game Boy is split between 8 kB of Video RAM (VRAM), and 160 B of Sprite attribute memory (OAM). The VRAM primarily consists of what is called "tiles" in Game Boy language. Each tile typically consists of 8 by 8 pixels (px). These tiles are the basis of all display on the LCD, which is 160 px wide by 144 px tall. Within memory, the Game Boy is limited to a total of 256 total tiles. The Game Boy splits VRAM into two halves, one half for static background images, and the other half for dynamic sprites. This can be perfectly split into 128 tiles for sprites, and 128 tiles for background memory, or however else one pleases. This can be visualized in memory as shown in **Figure 2.**

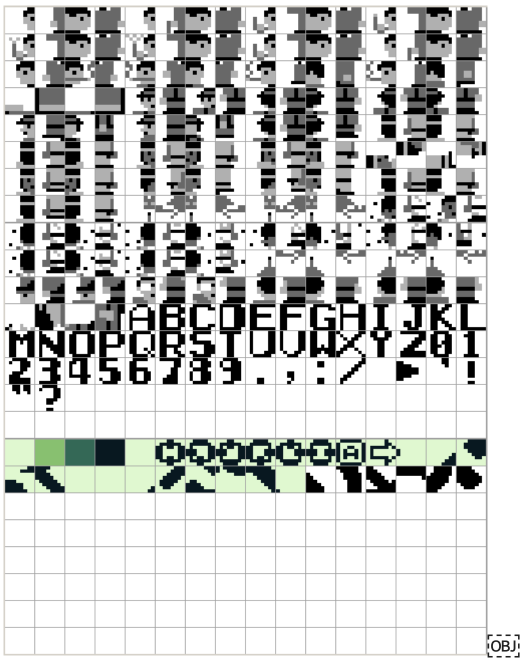


Figure 2: All the way down through to the question mark consist of sprite tiles. Then there is a row of blank tiles (leftover sprite tile space), after which are the background tiles. *The Crab Cannery Ship* game uses exactly 256 tiles.

Fortunately, the issue of tile number is scalable. There are many solutions, but the one which *The Crab Cannery Ship* game employs is reliance on the 160 B of OAM mentioned above. In plain

English, these 160 bytes consist of “sprite attributes.” Without describing the exact mechanics, this partition of memory delineated as OAM is responsible for telling the Game Boy the direction a sprite is facing, as well as its color palette. Through manipulations of these attributes a game developer for the Game Boy can easily half the amount of sprite tiles previously thought to be necessary.

The primary example of where the palette comes in handy is when dealing with text on different colored backgrounds. Without the use of the palette one would need two sets of tiles utilizing two separate palettes. The tile set that handles alphanumeric characters as well as a small selection of punctuation marks consists of 46 total tiles. Doubling this would end up taking almost half of tile memory alone. The directional sprite attributes are also large assets. Given any symmetrical sprite the number of tiles required for said sprite can at least be halved. In this project the sprites could be made up of four tiles, taking up an entirety of 16 px by 16 px tiles. Given that a forward-facing character’s left side is identical to its write, a simple method of conserving space would be to only store the left or right side in memory. The Game Boy allows for up to 40 sprites on the screen at a time. Each sprite is a copy of a given tile. In order to display an entire sprite with only half its tiles in memory one would make a copy of the half unaltered, and another copy flipped over the x-axis. By placing these two copies next to each other a complete sprite is formed. Not only do sprites have static symmetry, but they can also have dynamic symmetry. That is, sprites can have symmetry in their animations. This is great news for the Game Boy programmer, given that animations can be much costlier than a static sprite. Without the use of sprite attributes, animating a 16 px by 16 px sprite walking to the bottom of the screen would take at least eight tiles. This can again be halved through the use of attributes. Given one half of a sprite depicted with its foot up, and another half with its foot down (as depicted in **Figure 3.a**) these two sprites can be flipped over the x-axis and swap positions iteratively. Typically, in modern programming the use of the modulo operator is a viable means of doing some actions on odd iterations and others on even iterations. The Game Boy's hardware can be much less forgiving, and because of this, it is much more efficient to use bitwise operations to check for the even or oddness of a number. **Figure 3.b** displays an example snippet of some code for animating a sprite in this manner.

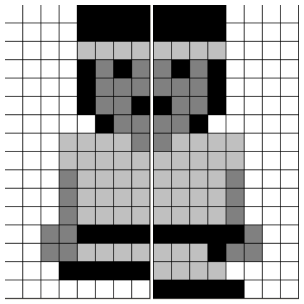


Figure 3.a: Here is a depiction of a walking sprite split in two eight by sixteen pixel sets. They can be animated by flipping and swapping their positions.

```
if((hero_posx+hero_posy)&0x1)
{
    set_sprite_prop(0, S_FLIPX);
    set_sprite_prop(1, S_FLIPX);
    move_sprite(0, hero_posx+SPRITE_WIDTH, hero_posy);
    move_sprite(1, hero_posx, hero_posy);
}
else
{
    set_sprite_prop(0, S_FLIPX&0x0);
    set_sprite_prop(1, S_FLIPX&0x0);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+SPRITE_WIDTH, hero_posy);
}
```

Figure 3.b: This code is simply saying that if the sprite (called "hero" in this case) is at an odd numbered pixel location on the screen then flip the left half of the sprite (sprite 0 and 1), and swap their positions, otherwise do the reverse.

As mentioned earlier, the method most integral to *The Crab Cannery Ship* game would inevitably end up being the manner in which it printed text to the LCD screen. An understanding of this projects print method only requires knowledge of the American Standard Code for Information Interchange (ASCII). ASCII is simply the equivalence between characters, and integers. In essence, it is the notion that to a computer the decimal value 65 is equivalent to the uppercase 'A' character. In C the following program: `printf("%d", 'A'-23);` will print the decimal value of 'A'-23, which is 42. Whenever the sprite tiles for characters, integers, and punctuation are loaded into memory, they essentially exist in tile memory from OAM address 0 to 2D (or 45 in decimal). Other than punctuation and number values, the tiles exist in the same order they appear in the ASCII chart. This means via the constant subtraction of 65 from the value of a uppercase letter will always derive the tile number in OAM which depicts that letter. The way this looks in C is displayed in **Figure 5.b**. Due to the difference in offset for numbers and punctuation tiles, a change in input can result in the desired output. This is to say that although the character '1' does not correspond to the tile with a '1' on it in OAM, a '\' does. Therefore, whenever we wish to print a '1' we type '\'. This logic is repeated for all those tile numbers that do not match up with their ASCII value minus 65. An example print method call is displayed in **Figure 5.a**.

a.
```
print("HELLO\nWORLD\0", 24, 32);
```
b.
```
void print(char *arr, UINT8 x, UINT8 y)
{
    SPRITES_8x8;
    cushion = letter_count;
    set_sprite_data(0, 46, font);
    start_x = x;
    for(i = 0; arr[i] != '\0'; ++i)
    {
        if(arr[i] == '\n')
        {
            y+=16;
            x = start_x;
            continue;
        }
        set_sprite_tile((i+cushion), (arr[i]-65));
        move_sprite((i+cushion), x, y);
        ++letter_count;
        x+=8;
    }
}
```

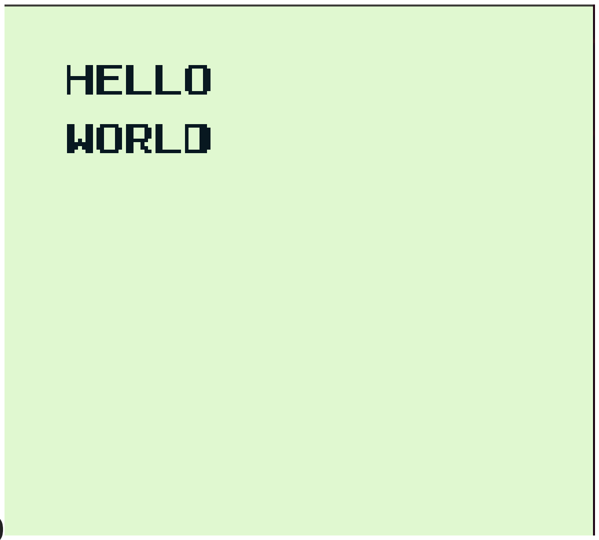
(c)

```
HELLO
WORLD
```

Figure 5: (a) A single calling of the print function. **(b)** The contents of the function can be recognized to be a single passing over the input string. **(c)** The resulting display on the Game Boy.

As can be seen in **Figure 5.a** there is also the addition of special characters such as '\n'. This character is used to automate the formatting. The technique of automating the format of text is integral in saving space in ROM. The automation allows for a major decrease in the amount of times the print function needs to be called, in that without this automation whenever a new line is to be printed the print function would have to be called again. In this way entire pages of text can be printed with a single function call. Saving memory in this way allows for more thematic and narrative expansion, or perhaps more items or enemies during battle.

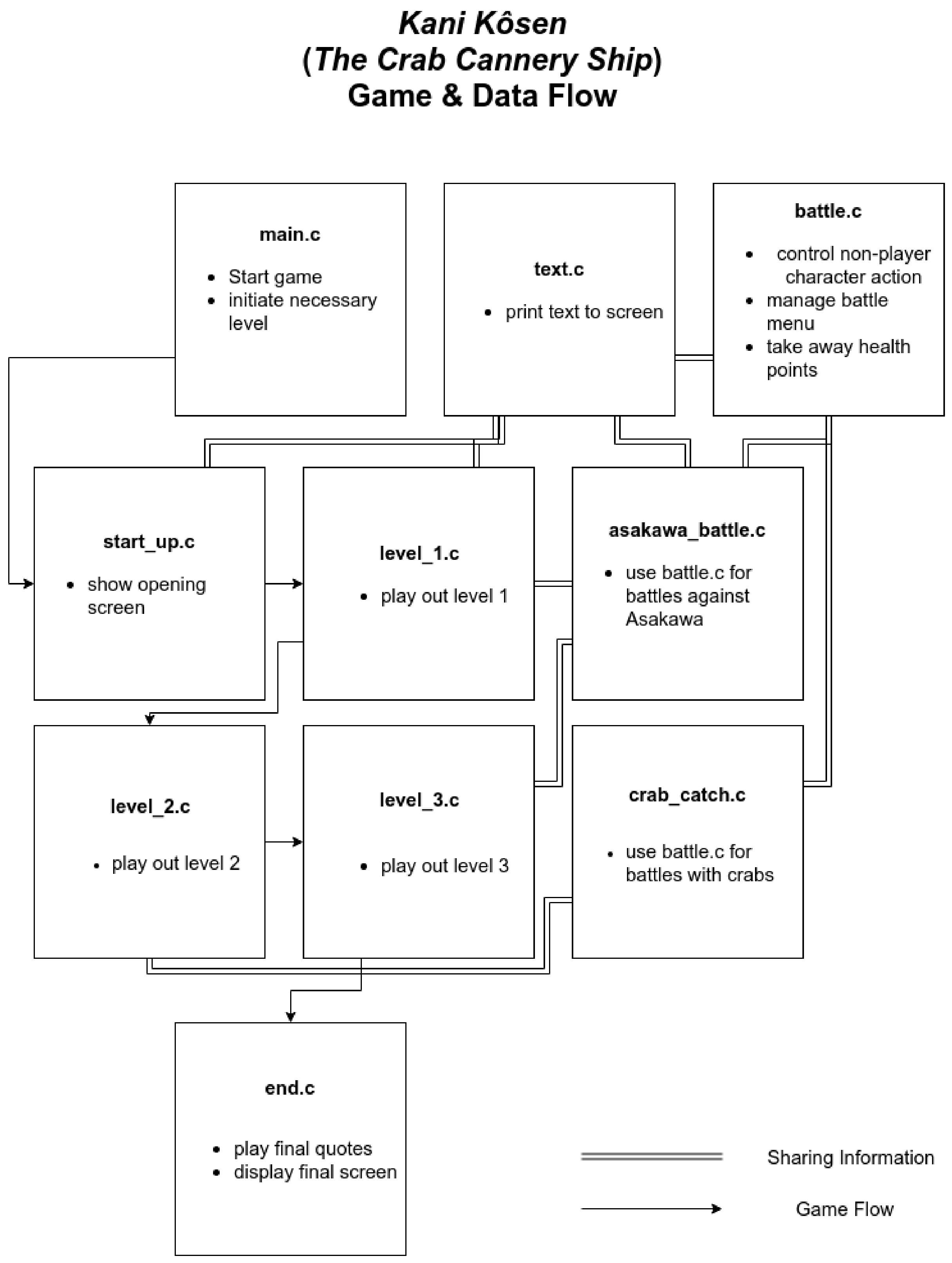
Another element particular to the RPG genre is the manner in which one battles enemies. Unlike a first-person shooter, the player is presented with a selection of moves or items to perform in the coming round, and after the choice is made these actions are executed with the added chance of missing or hitting. The computational element of these functions is less inherently interesting or clever, but as a core element of the game they deserve a brief going into. What is important about the formation of this collection of commands is that they be dynamic enough to apply to multiple battles with different enemies. The central control function of all battles, called “fight,” relies on a number of outside procedures to keep this part of the game running smoothly. The code, in plain English, begins by generating four random numbers. Three of these random numbers are chosen between 0 and 5, and are respectively responsible for determining the non-player character’s (npc’s) choice in action, accuracy, and the character to go first (either the player or the computer). The last random number generated is between 0 and 10, and is responsible for the player’s accuracy. After these numbers are configured, the battle can be played out. The game is skewed towards letting the npc get the first hit, that is, if the random number chosen between 0 and 5 is greater than 0, the npc will go first. After the npc makes their chosen move (based on the random numbers generated) damage will be dealt to the player. In the event that the player is not dead, their chosen action will then be executed and landed depending on the accuracy determined by its chosen random number. The exact reverse operation is then performed in the event that the player gets to act first. The generalized process is displayed in a snippet of code from the `fight` method below:

```
void fight(UINT8 *fighter_hp, UINT8 *enemy_hp)
{
    fight_config();
    if(first_fighter >= 1)
    {
        npc_fight();
        damage(fighter_hp);
        if(state != DEAD)
        {
            hero_fight();
            enemy_damage(enemy_hp);
        }
    }
    else
    {
        hero_fight();
        enemy_damage(enemy_hp);
        if(state != BATTLE_WIN)
        {
            npc_fight();
            damage(fighter_hp);
        }
    }
}
```

The most interesting aspect of programming the battles is determining the best way to check for a character's death. Again, in modern computation this would be trivial. One could simply check if the character's health points are less than or equal to zero. However, an issue arises when deciding between using signed or unsigned integers. The relevant difference between these two integers is the representation of their bits. For a signed integer, the final bit on either the left or right of the bit string (depending on the system) stands in as the negative sign if it is on (is set to one). Unsigned integers allow the programmer to count to higher numbers with the same number of bits because rather than utilizing that final bit as a negative sign, it is used for its decimal value. This means that the unsigned integer is capable of counting higher than the signed integer, but it is incapable of counting negatively. This can be problematic when looking for events that result in a character’s health points dropping to or below zero. In the event that a character is dealt damage greater than its health points the code will be incapable of recognizing its death due to the subtraction being below zero, and therefore being positive. The only time at which this code would recognize a character’s death is when it gets to zero exactly. Therefore, a reliable means of monitoring a character’s health points is to deal damage incrementally. Specifically, whenever a character has damage dealt to their health, rather than simply subtracting the amount of damage from their health points all in one go, it is best to incrementally subtract one from their health in a loop until the number of increments is equal to the amount of damage dealt. During this incrementation there can be a check to see if the character’s health is at zero, in the event that it is they will be deemed dead, and the battle will end.

After having established the elements that enabled the game to even exist in memory, it is important to recognize the logic, or flow of the game. **Figure 6** should give a general idea of the interrelation of the different levels and primary functions of the game.

Figure 6:


Kani Kôsen
(*The Crab Cannery Ship*)
Game & Data Flow
main.c
Start game
initiate necessary level
text.c
print text to screen
battle.c
control non-player character action
manage battle menu
take away health points
start_up.c
show opening screen
level_1.c
play out level 1
asakawa_battle.c
use battle.c for battles against Asakawa
level_2.c
play out level 2
level_3.c
play out level 3
crab_catch.c
use battle.c for battles with crabs
end.c
play final quotes
display final screen
Sharing Information
Game Flow

To be clear, the file titled “main” can be thought of as the main control, or heart of the game. It does the job of calling the other primary control methods of the starting screen, and the rest of the levels. The remaining two files at the top of **Figure 6** consist of functions generalized enough to be useful throughout the entirety of the game. It follows that the one titled “battle” is used for handling important events during battle scenes, and the file titled “text” is used for displaying text on the screen in different ways. As mentioned earlier, “text” is the most heavily used file throughout the entirety of the game, whereas the “battle” file acts as a group of helper functions for “crab catch,” and “asakawa battle.” Each of which rely almost solely on “battle,” to deal with the loss of character health, and sprite animations, and general event handling during battle. What is unique to both “crab catch” and “asakawa battle” are there respective sprites and therefore changes in a character’s attack accuracy, health points, and ability to deal damage. As for the rest of the files, the layout is intuitive: “main” prompts the player to play, choosing to play (the only choice), begins level 1, finishing level 1 triggers the beginning of level 2, which when finished starts level 3, and then it all comes to a close in “end.”

Retrospective Comments

Despite the current functional state of the game, as is often the case with any body of work, improvements are worth being made. *The Crab Cannery Ship* game would benefit most from further optimization of the “print” function throughout the game. Ideally, an entire file would be allotted towards holding all the text within the game. Each particular dialogue would be given a name, which when entered into print could be automatically displayed, and anticipate the pressing of buttons to trigger moving onto the next passage. Not only would this improve gameplay in its ability to enhance physical interactivity, it would also lower the amount of memory. Further, if the game was to be rewritten in assembly, through the added use of memory banking extra tiles could be added to enhance the currently opaque background. As mentioned above, the addition of more detailed scenes, characters, and enemies would be preferable, but as it is, the use of C as a programming language for hardware with already such limited capacity would be unlikely to allow these kinds of expansions. In full, when programming larger scale games with deadlines far in the future, one is best off doing it in the Game Boy's z80 assembly language. For the purposes of this project C has served very well though, and is a pragmatic and approachable tool for relatively quick game development for the Game Boy console.

Conclusion

With the closure of this project, only one question remains: "Why?" Why use the Game Boy? Why use it as a technology? Why use it as a medium? Or generally, why use it at all? Finding some form of answer to any single one of these questions is often a validating exercise. Fortunately, there can be several well-founded responses. Starting with the first, "why use the Game Boy as a technology?" is truly a question of why one would readdress an outdated piece of hardware. At first glance it seems that one would be hard pressed to learn or do anything with the Game Boy that has not already been done. On the contrary, despite the Game Boy's "withered technology" at the time of its creation, Yokoi Gunpei persisted in "lateral thinking." This is a mentality that connects the act of programming to the arts. The striving to break the mold and try one's hand at creation in a new way. Just as the individual who produces a work of art is distilled in that work, so too is a programmer distilled in their code and their program. So, when asked, "why use the Game Boy as a technology?" the answer lies within the fact that to readdress an old and withered technology is to exercise the core practice of being a programmer and artist. That is, to work with the Game Boy, is to try at recreating, and coming up with new ideas and solutions to old questions or problems.

The answers do not end there. This withered technology is a stand in for all handheld gaming platforms and what is unique to them. As the progenitor to the handheld consoles produced by Nintendo today, what better means of investigating the handheld console as a modern medium? It was previously established in "Making the Game – Adaptation as Reading" that videogames, as a medium, have their own unique means of developing immersion, and empathy towards the characters being played. In brief, as established in John Huizinga's *Homo Ludens*, through the immersive qualities of play that are inherent in the action of "gaming" with videogames one allows themselves to voluntarily take on the role of a character, and since it is play, it is done with free will, and open mindedness. Though this clearly states the unique capabilities of videogames, it says nothing for the Game Boy as a platform through which the videogame is received. In Daniel Reynold's, "The Vitruvian Thumb: Embodied Branding and Lateral Thinking with the Nintendo Game Boy," he looks to the materiality of the Game Boy to affirm his assertion that it is "the intimate relationship between bodies and technologies that characterize videogame play." In other words, it is by the haptic qualities of play that are particular to the handheld console which enhance, and arguably, enable one's personal investment in a game and its narrative in the first place. What separates the Game Boy from other mediums is not the visuals it shares with film, or the text it shares with literature, but the physical interaction with narrative. It is because of the "transmedia properties" of the Game Boy that not only can a player enter mentally into the narrative proposed within a game, but they can enter physically through the enablement of the joypad and buttons supplied via the console, and because it is handheld, they can do it anywhere.

Despite the haptic qualities of all consoles, unless they are handheld, the experience of the narrative within the game is limited to the space in which the console is played. That is, before the portability of the handheld console, consumption of the media was defined by the minute amount of spaces in which it could be consumed. The territories in which our intake of media takes place is relevant, but how? Before mass reproduction of media was made possible via the printing press or tapes of film those that could intake the media were very few, and the experience of that intake was circumscribed and defined by the rare spaces in which said intake could take place. Through mass production this experience is altered, less rarified, and in exchange, personalized. The chronological movement of the videogame from the arcade, to the television, to the pocket is significant in its ability to gradually personalize the meaning of the narrative by allowing the consumer to choose when and where they partake in its experience.

Through the portability of the Game Boy it achieves a level of personalization unique to other portable media such as the novel. It was in 1929 that Kobayashi Takiji published *The Crab Cannery Ship*, and brought hope of liberation from the bourgeoisie to the proletariat. Through the passage of time this hope has remained alive, and has persisted within the evolution of mediums. With the evolution of mediums this message will always be heard: “The proletarians have nothing to lose but their chains. They have a world to win.”

```c
/**
 * text.c
 *
 * All print functions and methods
 * concerned with text formatting.
 */

#include "text.h"
#include <stdio.h>
#include <stdlib.h>

UINT8 i;
UINT8 j;
UINT8 cushion;
UINT8 LETTER_COUNT = 0;
/* MAX number of sprites that can be on screen at once */
UINT8 MAX_SPRITES = 40;

UINT8 START_X = 0;

/* hide arrow off screen */
UINT8 arrow_x = 200;
UINT8 arrow_y = 200;

/**
 * 0 = [     . = e
 * 1 = \     , = f
 * 2 = ]     : = g
 * 3 = ^     / = h
 * 4 = _    ' '= i
 * 5 = `     > = j
 * 6 = a     ' = k
 * 7 = b     ! = l
 * 8 = c     " = m
 * 9 = d     ? = n
 *
 * {A...Z} = {A...Z}
 */
void print(char *arr, UINT8 x, UINT8 y)
{
    SPRITES_8x8;
    cushion = LETTER_COUNT;
    set_sprite_data(0, 46, font);
    START_X = x;
    for(i = 0; arr[i] != '\0'; i++)
    {
        if(arr[i] == '\n')
        {
            y+=16;
            x=START_X;
            continue;
        }
        else
        {
            set_sprite_tile((i+cushion), (arr[i]-65));
            move_sprite((i+cushion), x, y);
            ++LETTER_COUNT;
            x+=8;
        }
    }
}
```

```

void bkg_clean(void)
{
    set_bkg_data(0, 10, chain_border_tiles);
    set_bkg_tiles(0,0,20,18,chain_border);
}

void sprite_clean(UINT8 start)
{
    /**
     * could also try making this letter
     * count instead of MAX SPRITES
     */
    for(i = start; i < MAX_SPRITES; ++i)
    {
        set_sprite_tile(i, 40);
        set_sprite_prop(i, S_FLIPX&0x0);
    }
}

void hide_sprites(void)
{
    for(i = 0; i < MAX_SPRITES; ++i)
    {
        move_sprite(i, 250, 250);
    }
}

/* special text settings for battles */
void battle_print(char *arr, UINT8 x, UINT8 y)
{
    SPRITES_8x8;

    set_sprite_data(0, 46, font);
    set_bkg_data(0, 10, chain_border_tiles);
    set_bkg_tiles(0,0,20,18, small_chain_border);

    cushion = LETTER_COUNT;

    for(i = 0; arr[i] != '\0'; ++i)
    {
        set_sprite_tile((i+cushion), (arr[i]-65));
        move_sprite((i+cushion), x, y);
        ++LETTER_COUNT;
        x+=8;
    }
}

void damn_that_asakawa(void)
{
    sprite_clean(8);
    LETTER_COUNT = 8;
    print("DAMNiTHAT\nASAKAWA\0", 24, 48);
}

void you_hear(void)
{
    print("YOUiHEARln\0", 24, 48);
}

/**
```

```
 * labels for the different
 * characters when they talk
 */
void asakawa(void)
{
    print("ASAKAWAg\0", 24, 32);
}

void fisherman(void)
{
    print("FISHERMANg\0", 24, 32);
}

void student(void)
{
    print("STUDENTg\0", 24, 32);
}

void miner(void)
{
    print("MINERg\0", 24, 32);
}

void workers(void)
{
    print("WORKERSg\0", 24, 32);
}

void battle_bkg_clean(void)
{
    set_bkg_data(0, 10, chain_border_tiles);
    set_bkg_tiles(0,0,20,18,small_chain_border);
}
```

```
/**
 * battle.c
 *
 * All functions necessary for
 * controlling battles.
 */

#include <rand.h>
#include <gb/drawing.h>
#include <stdlib.h>
#include <stdio.h>
#include <rand.h>
#include "../level_1/asakawa_battle.h"
#include "battle.h"
#include "../text/text.h"
#include "../start_up/start_up.h"
#include "../level_2/crab_catch.h"
#include "../assets/level_assets/level_assets.h"
#include "../assets/sprites/soldier.h"
#include "../assets/sprites/asakawa_front_idle.h"
#include "../assets/sprites/hero_back_idle.h"
#include "../assets/sprites/fisherman_idle_back.h"
#include "../assets/sprites/student_idle_back.h"
#include "../assets/sprites/asakawa_air_shot.h"

/**
 * 0 = asakawa;
 * 1 = crab
 * 2 = king crab
 * 3 = beriberi
 * 4 = lice
 */

UINT8 PUNCH = 1;
UINT8 DEFEND = 2;
UINT8 ITEM = 3;

UINT8 NET = 2;
UINT8 CLUB = 5;

UINT8 SHOOT = 10;
UINT8 CLAW = 2;
UINT8 KING_CLAW = 3;

UINT8 PUNCH_LOC = 32;

/** change above to define **/
UINT8 BATTLE_CHOICE = 0;
UINT8 FIGHT_CHOICE = 1;
UINT8 RUN_CHOICE = 2;
UINT8 ITEM_CHOICE = 3;
UINT8 FIGHTING = 4;
UINT8 DEAD = 5;
UINT8 BATTLE_WIN = 6;

UINT8 enemy = 0;
UINT8 battle_num = 0;

UBYTE items = 0;
UINT8 chosen_item = 0;
UINT8 choosing = 0;

UINT8 enemy_choice;

UWORD seed;

UBYTE npc_act, npc_acc, hero_acc, first_fighter = 0; //UBYTE

/* a stepping variable for character animations */
UINT8 y = 72;
UINT8 enemy_y = 40;
/* another animation counter */
UINT8 a = 0;

UWORD h_hp[4];
UWORD e_hp[4];

void battle_nav(void)
{
    if(arrow_y == 32)
    {
        state = FIGHT_CHOICE;
        fight_opt();
    }
    if(arrow_y == 48)
    {
        choice = DEFEND;
        state = FIGHTING;
    }
    if(arrow_y == 64)
    {
        state = RUN_CHOICE;
        run();
    }
    if(arrow_y == 80)
    {
        state = ITEM_CHOICE;
        item_opt();
    }
}

void run(void)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    battle_print("YOU\0", 18, 32);
    battle_print("CANkT\0", 18, 48);
    battle_print("ESCAPEl\0", 18, 64);
    show_fighter_stats();
    delay(1000);
    back();
}

void back(void)
{
    if(state < FIGHTING)
    {
        battle_menu();
        state = BATTLE_CHOICE;
        choice = 0;
    }
}

void fight_opt(void)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    battle_bkg_clean();
    battle_print("j\0", 18, 32);
    battle_print("iPUNCH\0", 18, 32);
    battle_print("AiSELECT\0", 16, 130);
    battle_print("BiBACK\0", 16, 142);
    show_fighter_stats();
}

void item_opt(void)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    battle_bkg_clean();
    if(items == 0)
    {
        battle_print("iEMPTY\0", 18, 32);
        battle_print("AiSELECT\0", 16, 130);
        battle_print("BiBACK\0", 16, 142);
    }
    if(items == 2)
    {
        choosing = 1;
        arrow_x = 24;
        arrow_y = 32;
        battle_print("j\0", arrow_x, arrow_y);
        battle_print("iCLUB\0", 24, 32);
        battle_print("iNET\0", 24, 48);
        battle_print("AiSELECT\0", 16, 130);
        battle_print("BiBACK\0", 16, 142);
        show_fighter_stats();
        arrow_y = 32;
        while(choosing)
            choose_item_toggle();
    }
}

void choose_item_toggle(void)
{
    if(joypad() & J_UP)
    {
        if(arrow_y <= 32)
            arrow_y = 64;
        delay(100);
        move_sprite(0, arrow_x, arrow_y-=16);
    }
    if(joypad() & J_DOWN)
    {
        if(arrow_y >= 48)
            arrow_y = 16;
        delay(100);
        move_sprite(0, arrow_x, arrow_y+=16);
    }
    if(joypad() & J_A)
    {
        if(arrow_y == 32)
        {
            chosen_item = CLUB;
            state = FIGHTING;
        }
        if(arrow_y == 48)
        {
            chosen_item = NET;
            state = FIGHTING;
        }
        choosing = 0;
    }
    if(joypad() & J_B)
    {
        choosing = 0;
        back();
    }
}

void battle_menu(void)
{

    sprite_clean(0);
    LETTER_COUNT = 0;

    arrow_x = 20;
    arrow_y = 32;
    /* setup the bkg */
    set_bkg_data(0,10, chain_border_tiles);
    set_bkg_tiles(0,0,20,18,small_chain_border);

    hide_sprites();
    /* selection menu */
    battle_print("j\0", arrow_x, arrow_y);
    battle_print("iFIGHT\0", 18, 32);
    battle_print("iDEFEND\0", 18, 48);
    battle_print("iRUN\0", 18, 64);
    battle_print("iITEM\0", 18, 80);
    show_fighter_stats();
}

void battle_toggle_up(void)
{
    /* if at top && up */
    if(arrow_y == 32)
    {
        arrow_y = 96;
        delay(100);
        move_sprite(0, arrow_x, arrow_y-=16);
        delay(100);
    }
    else
    {
        delay(100);
        move_sprite(0, arrow_x, arrow_y-=16);
        delay(100);
    }
}

void battle_toggle_down(void)
{
    /* if at bottom && down */
    if(arrow_y == 80)
    {
        arrow_y = 32;
        delay(100);
        move_sprite(0, arrow_x, arrow_y);
        delay(100);
    }
    else
    {
        delay(100);
        move_sprite(0, arrow_x, arrow_y+=16);
        delay(100);
    }
}

void battle_toggle_ctrl(void)
{
    SPRITES_8x8;

    if(joypad() & J_A)
    {
        if(state == FIGHT_CHOICE)
            state = FIGHTING;
        else
            battle_nav();
    }

    if(joypad() & J_B)
```

```
        back();

    /* toggle down options */
    if(joypad() & J_DOWN && state == BATTLE_CHOICE)
        battle_toggle_down();

    /* toggle up options */
    if(joypad() & J_UP && state == BATTLE_CHOICE)
        battle_toggle_up();
}

void choice_handler(UINT8 arrow_y)
{
    if(arrow_y == PUNCH_LOC && choice != ITEM)
        choice = PUNCH;
    if(chosen_item == CLUB)
        choice = ITEM;
    if(chosen_item == NET)
        choice = ITEM;
}

/**
 * need to refine this logic
 */
void hero_fight(void)
{

    if(choice == PUNCH || choice == ITEM)
    {
        /* figher chooses punch and hits */
        hero_fight_anim();
        for(a = 0; a < 6; ++a)
        {
            delay(100);
            DISPLAY_OFF;
            delay(100);
            DISPLAY_ON;
        }
        sprite_clean(0);
        LETTER_COUNT = 0;
        if((choice == ITEM) && (chosen_item == CLUB) &&
                (enemy > 0) && (!CRAB_CAUGHT))
            print("YOUiNEED\nTOiCATCH\nTHEiCRABl\0", 72, 64);
        else if((choice == ITEM) && (chosen_item == NET) &&
                (hero_acc > 0))
        {
            print("CAUGHTl\0", 72, 80);
            CRAB_CAUGHT = 1;
        }
        else if(((choice == PUNCH) && (hero_acc >= 3)) ||
                ((choice == ITEM) && hero_acc > 0))
        {
            print("YOUiHIT\0", 56, 72);
            if(enemy == 0)
                print("ASAKAWA\0", 56, 88);
            if(enemy == 1)
                print("AiCRAB\0", 56, 88);
            if(enemy == 2)
                print("KINGiCRAB\0", 56, 88);
            delay(500);
        }
        /* fighter chooses punch and misses */
                  else if(((choice == PUNCH) && (hero_acc < 3)) ||
                          ((choice == ITEM) && (hero_acc == 0)))
                  {
                      sprite_clean(0);
                      LETTER_COUNT = 0;
                      print("YOUiMISSl\0", 64, 80);
                      delay(500);
                      clear_screen();
                  }
    }
    /* fighter defends */
    if(choice == DEFEND)
        hero_defend_anim();
}

void hero_defend_anim(void)
{
    if(enemy == 0)
        sprite_setup(8, hero_back_idle, 8, asakawa_front_idle);
    if(enemy == 1)
        sprite_setup(8, hero_back_idle, 8, norm_crab);
    if(enemy == 2)
        sprite_setup(8, hero_back_idle, 8, king_crab);
    for(a = 0; a < 21; a+=3)
    {
        delay(100);
        set_sprite_tile(4, (8+(a&0x4)));
        move_sprite(4, 77, y);
        set_sprite_tile(5, (9+(a&0x4)));
        move_sprite(5, 77, y+8);
        set_sprite_tile(6, (10+(a&0x4)));
        move_sprite(6, 84, y);
        set_sprite_tile(7, (11+(a&0x4)));
        move_sprite(7, 84, y+8);
        delay(100);
    }
    y = 72;
}

void enemy_defend_anim(void)
{
    /* enemy */
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 1);
    set_sprite_tile(2, 2);
    set_sprite_tile(3, 3);

    move_sprite(0, 77, 40); // top left of the head
    move_sprite(1, 77, 48); // bottom left of body
    move_sprite(2, 84, 40); // top right of head
    move_sprite(3, 84, 48); // bottom right of body
    sprite_setup(8, hero_back_idle, 8, asakawa_front_idle);
    for(a = 0; a < 21; a+=3)
    {
        delay(100);
        set_sprite_tile(0, (0+(a&0x4)));
        move_sprite(0, 77, y);
        set_sprite_tile(1, (1+(a&0x4)));
        move_sprite(1, 77, y+8);
        set_sprite_tile(2, (2+(a&0x4)));
        move_sprite(2, 84, y);
        set_sprite_tile(3, (3+(a&0x4)));
        move_sprite(3, 84, y+8);
        delay(100);
    }
    enemy_y = 40;
}

void hero_fight_anim(void)
{
    if(enemy == 0)
        sprite_setup(8, hero_back_idle, 8, asakawa_front_idle);
    if(enemy == 1)
        sprite_setup(8, hero_back_idle, 8, norm_crab);
    if(enemy == 2)
        sprite_setup(8, hero_back_idle, 8, king_crab);
    delay(300);
    /* essentially hiding one sprite */
    for(i = 4; i < 8; ++i)
        move_sprite(i, 200, 200);

    delay(300);
    move_sprite(4, 77, 58);
    move_sprite(5, 77, 66);
    move_sprite(6, 84, 58);
    move_sprite(7, 84, 66);
    for(a = 0; a < 21; a+=3)
    {
        delay(100);
        set_sprite_tile(4, (8+(a&0x4)));
        set_sprite_tile(5, (9+(a&0x4)));
        set_sprite_tile(6, (10+(a&0x4)));
        set_sprite_tile(7, (11+(a&0x4)));
    }
}

void enemy_fight_anim(void)
{
    if(enemy == 0)
        sprite_setup(8, hero_back_idle, 8, asakawa_front_idle);
    if(enemy == 1)
        sprite_setup(8, hero_back_idle, 8, norm_crab);
    if(enemy == 2)
        sprite_setup(8, hero_back_idle, 8, king_crab);
    delay(300);
    /* essentially hiding one sprite */
    for(i = 0; i < 4; ++i)
        move_sprite(i, 200, 200);

    delay(300);
    move_sprite(0, 77, 52);
    move_sprite(1, 77, 60);
    move_sprite(2, 84, 52);
    move_sprite(3, 84, 60);
    for(a = 0; a < 21; a+=3)
    {
        delay(100);
        set_sprite_tile(0, (0+(a&0x4)));
        set_sprite_tile(1, (1+(a&0x4)));
        set_sprite_tile(2, (2+(a&0x4)));
        set_sprite_tile(3, (3+(a&0x4)));
    }
}

void clear_screen(void)
{
    /* clear the bkg */
    set_bkg_data(0,4, blank_screen_tiles);
    set_bkg_tiles(0,0,20,18,blank_screen);
}

/**
 * hnb    -- hero number of tiles
 * enb    -- enemy number of tiles
 * x_data -- sprite sheet
 */
void sprite_setup(UINT8 hnb, unsigned char *hero_data,
        UINT8 enb, unsigned char *enemy_data)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    clear_screen();

    /* setup sprites */
    set_sprite_data(0, 8, enemy_data);
    set_sprite_data(hnb, enb, hero_data);

    /* enemy */
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 1);
    set_sprite_tile(2, 2);
    set_sprite_tile(3, 3);

    move_sprite(0, 77, 40); // top left of the head
    move_sprite(1, 77, 48); // bottom left of body
    move_sprite(2, 84, 40); // top right of head
    move_sprite(3, 84, 48); // bottom right of body

    /* hero */
    set_sprite_tile(4, 8);
    set_sprite_tile(5, 9);
    set_sprite_tile(6, 10);
    set_sprite_tile(7, 11);

    move_sprite(4, 77, 72);
    move_sprite(5, 77, 80);
    move_sprite(6, 84, 72);
    move_sprite(7, 84, 80);

    if(striking)
    {
        if(!REVOLUTION_2)
        {
            set_sprite_data(16, 2, soldier);
            /* soldiers */
            set_sprite_tile(8, 16); // two on the left
            set_sprite_tile(9, 17);
            set_sprite_tile(10, 16); // two on the right
            set_sprite_tile(11, 17);
            set_sprite_tile(12, 16);
            set_sprite_tile(13, 17);
            set_sprite_tile(14, 16);
```

```
                set_sprite_tile(15, 17);
                set_sprite_prop(14, S_FLIPX);
                set_sprite_prop(15, S_FLIPX);
                set_sprite_prop(10, S_FLIPX);
                set_sprite_prop(11, S_FLIPX);
                move_sprite(8, 51, 40);
                move_sprite(9, 51, 48);
                move_sprite(10,51+sprite_width, 40);
                move_sprite(11,51+sprite_width, 48);
                move_sprite(12,99, 40);
                move_sprite(13,99, 48);
                move_sprite(14,99+sprite_width, 40);
                move_sprite(15,99+sprite_width, 48);
            }

            /* fisherman && student */
            set_sprite_data(18, 2, fisherman_idle_back);
            set_sprite_data(20, 2, student_idle_back);
            set_sprite_tile(16, 18);
            set_sprite_tile(17, 19);
            set_sprite_tile(18, 18);
            set_sprite_tile(19, 19);
            set_sprite_tile(20, 20);
            set_sprite_tile(21, 21);
            set_sprite_tile(22, 20);
            set_sprite_tile(23, 21);
            set_sprite_prop(18, S_FLIPX);
            set_sprite_prop(19, S_FLIPX);
            set_sprite_prop(22, S_FLIPX);
            set_sprite_prop(23, S_FLIPX);
            move_sprite(16, 51, 72);
            move_sprite(17, 51, 80);
            move_sprite(18, 51+sprite_width, 72);
            move_sprite(19, 51+sprite_width, 80);
            move_sprite(20, 99, 72);
            move_sprite(21, 99, 80);
            move_sprite(22, 99+sprite_width, 72);
            move_sprite(23, 99+sprite_width, 80);
        }
}

/**
 * Note to self: npc_fight does not ever defend...
 */
void npc_fight(void)
{
    /* enemy fights and hits */
    if(npc_act >= 1 && npc_acc >= 1)
    {
        enemy_fight_anim();
        for(a = 0; a < 6; ++a)
        {
            delay(100);
            DISPLAY_OFF;
            delay(100);
            DISPLAY_ON;
        }
        DISPLAY_OFF;
        sprite_clean(0);
        LETTER_COUNT = 0;
        print("YOUkREiHITl\0", 60, 80);
        clear_screen();
        DISPLAY_ON;
        delay(500);
    }
    /* enemy fights and misses */
    if(npc_act >= 1 && npc_acc < 1)
    {
        enemy_fight_anim();
        for(a = 0; a < 6; ++a)
        {
            delay(100);
            DISPLAY_OFF;
            delay(100);
            DISPLAY_ON;
        }
        DISPLAY_OFF;
        sprite_clean(0);
        LETTER_COUNT = 0;
        clear_screen();
        print("THEYiMISSl\0", 60, 80);
        DISPLAY_ON;
        delay(500);
    }

    /* enemy defends */
    if(npc_act == 0)
    {
        DISPLAY_OFF;
        clear_screen();
        sprite_clean(0);
        LETTER_COUNT = 0;
        if(enemy == 0)
            print("ASAKAWA\0", 56, 75);
        if(enemy == 1)
            print("THEiCRAB\0", 56, 75);
        if(enemy == 2)
            print("KINGiCRAB\0", 56, 75);

        print("DEFENDS\0", 56, 91);
        DISPLAY_ON;
        delay(800);
    }
}

/**
 * fight helper function.
 * makes number decision before appearance.
 */
void fight_config(void)
{
    /**
     * generate random number to find
     * out asakawa's action and accuracy
     */
    seed = DIV_REG;
    seed |= (UWORD)DIV_REG << 8;
    initarand(seed);
    /* enemies action */
    npc_act = rand()%5;
    /* enemies accuracy */
    npc_acc = rand()%5;
    /* hero's accuracy */
    hero_acc = rand()%10;
    /* enemies action */
    first_fighter = rand()%5;
}


/* checking to see if you're dead */
void damage(UINT8 *fighter_hp)
{
    if(start_hp == 50)
        SHOOT = 30; // going to need to set this back to 10 later.
    if(start_hp == 100)
        SHOOT = 10;
    if(enemy == 0)
    {
        if(choice == DEFEND && npc_acc >= 1 && npc_act >= 1)
        {
            for(i = (SHOOT-DEFEND); i > 0; --i)
            {
                --(*fighter_hp);
                if((*fighter_hp) == 0)
                {
                    option = GAME_OVER;
                    state = DEAD;
                }
            }
        }
        else if(npc_acc >= 1 && npc_act >= 1 && choice == PUNCH ||
                choice == ITEM)
        {
            for(i = SHOOT; i > 0; --i)
            {
                --(*fighter_hp);
                if((*fighter_hp) == 0)
                {
                    option = GAME_OVER;
                    state = DEAD;
                }
            }
        }
    }
    if(enemy == 1)
    {
        if(choice == DEFEND && npc_acc >= 1 && npc_act >= 1)
        {
            for(i = (CLAW-DEFEND); i > 0; --i)
            {
                --(*fighter_hp);
                if((*fighter_hp) == 0)
                {
                    option = GAME_OVER;
                    state = DEAD;
                }
            }
        }
        else if(npc_acc >= 1 && npc_act >= 1 && (choice == PUNCH ||
                    choice == ITEM))
        {
            for(i = CLAW; i > 0; --i)
            {
                --(*fighter_hp);
                if((*fighter_hp) == 0)
                {
                    option = GAME_OVER;
                    state = DEAD;
                }
            }
        }
    }
    if(enemy == 2)
    {
        if(choice == DEFEND && npc_acc >= 1 && npc_act >= 1)
        {
            for(i = (KING_CLAW-DEFEND); i > 0; --i)
            {
                --(*fighter_hp);
                if((*fighter_hp) == 0)
                {
                    option = GAME_OVER;
                    state = DEAD;
                }
            }
        }
        else if(npc_acc >= 1 && npc_act >= 1 && (choice == PUNCH ||
                    choice == ITEM))
        {
            for(i = KING_CLAW; i > 0; --i)
            {
                --(*fighter_hp);
                if((*fighter_hp) == 0)
                {
                    option = GAME_OVER;
                    state = DEAD;
                }
            }
        }
    }
}

void lower_hp(UINT8 *hit, UINT8 defended, UINT8 *enemy_hp)
{
    if(start_hp == 50)
        (*hit) = 3;
    if(start_hp == 100)
        (*hit) = 10;
    if(defended)
    {
        for(i = ((*hit)-DEFEND); i > 0; --i)
        {
            --(*enemy_hp);
            if((*enemy_hp) == 0)
                state = BATTLE_WIN;
        }
    }
    else
    {
        if(enemy == 0 || CRAB_CAUGHT)
        {
            for(i = (*hit); i > 0; --i)
            {
                --(*enemy_hp);
                if((*enemy_hp) == 0)
                    state = BATTLE_WIN;
            }
        }
    }
}

void enemy_damage(UINT8 *enemy_hp)
{
    if(npc_act == 0)
    {
        if(choice == ITEM && chosen_item == NET && hero_acc > 0)
            return;
        if(choice == ITEM && chosen_item == CLUB && hero_acc > 0)
            lower_hp(&CLUB, 1, enemy_hp);
        if(choice == PUNCH && enemy == 0 && hero_acc >= 3)
            lower_hp(&PUNCH, 1, enemy_hp);
    }
    if(npc_act > 0)
    {
        if(choice == ITEM && chosen_item == NET && hero_acc > 0)
            lower_hp(&NET, 0, enemy_hp);
        if(choice == ITEM && chosen_item == CLUB && hero_acc > 0)
            lower_hp(&CLUB, 0, enemy_hp);
        if(choice == PUNCH && enemy == 0 && hero_acc >= 3)
            lower_hp(&PUNCH, 0, enemy_hp);
    }
}

/**
 * make decision on randomly generated numbers.
 */
void fight(UINT8 *fighter_hp, UINT8 *enemy_hp)
{
    fight_config();
    if(first_fighter >= 1)
    {
        npc_fight();
        damage(fighter_hp);
        if(state != DEAD)
        {
            hero_fight();
            enemy_damage(enemy_hp);
        }
    }
    else
    {
        hero_fight();
        enemy_damage(enemy_hp);
        if(state != BATTLE_WIN)
        {
            npc_fight();
            damage(fighter_hp);
        }
    }
    npc_act = 0;
    npc_acc = 0;
    hero_acc = 0;
    choice = 0;
    chosen_item = 0;
    first_fighter = 0;
```

```c
}

void show_fighter_stats(void)
{
    battle_print("HP\0", 88, 56);
    itoa(hero_hp, h_hp);

    for(i = 0; h_hp[i] != '\0'; ++i)
        h_hp[i] = (h_hp[i]+43);
    battle_print(h_hp, 88, 72);
    /* HP */
    if(start_hp == 10)
    {
        if(hero_hp == start_hp)
            battle_print("[\0", 96, 72);
        battle_print("WORKER\0", 88, 40);
        battle_print("h\\[\0", 122, 72);
    }
    if(start_hp == 50)
    {
        battle_print("LEADERS\0", 88, 40);
        battle_print("[\0", 96, 72);
        battle_print("h`[\0", 122, 72);
    }
    if(start_hp == 100)
    {
        battle_print("EVRY\\\0", 88, 40);
        if(hero_hp == start_hp)
            battle_print("[[\0", 96, 72);
        else
            battle_print("[\0", 96, 72);
        battle_print("h\\[[\0", 122, 72);
    }
}
```

```c
/**
 * level_assets.c
 *
 * Initializes variables needed
 * for all levels {1..3}
 */

#include "level_assets.h"
#include "../../level_2/level_2.h"
#include "../../text/text.h"

UINT8 sprite_width = 8;

UINT8 talking = 1;
UINT8 slept = 0;
UINT8 talked = 0;
UINT8 revolt = 0;
UINT8 REVOLUTION_1 = 0;
UINT8 REVOLUTION_2 = 0;
UINT8 striking = 0;
UINT8 appeared = 0;
UINT8 crabs_to_catch = 1;
UINT8 health_loss = 0;

UINT8 anim_1 = 32;
UINT8 anim_2 = 34;
```

```c

UINT8 hero_posx = 200;
UINT8 hero_posy = 200;
UINT8 fisherman_posx = 200;
UINT8 fisherman_posy = 200;
UINT8 fisherman2_posx = 200;
UINT8 fisherman2_posy = 200;
UINT8 miner_posx = 200;
UINT8 miner_posy = 200;
UINT8 student_posx = 200;
UINT8 student_posy = 200;
UINT8 student2_posx = 200;
UINT8 student2_posy = 200;
UINT8 asakawa_posx = 200;
UINT8 asakawa_posy = 200;
UINT8 bed_posx = 200;
UINT8 bed_posy = 200;

UINT8 hero_hp = 10;
UINT16 asakawa_hp = 100;
```

```c
/**
 * main.c
 */

#include "main.h"

void main(void)
{
    /* this poses the OPTIONS: START, NEW GAME, and QUIT. */
    opening();
    /* if you've gotten as far as level 1 */
    if(option == START)
        level_1_ctrl();
    /* if you've gotten as far as level 2 */
    if(option == LEVEL_2)
        level_2_ctrl();
    if(option == LEVEL_3)
        level_3_ctrl();
}
```

```c
/**
 * start_up.c
 *
 * opening screen handler
 */

#include "start_up.h"
#include "../text/text.h"
#include "../battle/battle.h"
#include "../assets/sprite_palette.h"
#include "../assets/bkg_palette.h"
#include "../assets/level_assets/level_assets.h"

/* actions that can be taken and returned */
UINT8 GAME_OVER = 0;
UINT8 START = 1;
UINT8 LEVEL_2 = 2;
UINT8 LEVEL_3 = 3;
UINT8 NEW_GAME = 4;
```

```
UINT8 QUIT = 5;

/* option chosen */
UINT8 option;

/* chosen decision */
UINT8 choice = 0;

void opening(void)
{
    DISPLAY_OFF;
    // set bkg property
    set_bkg_palette(0, 1, bkg_palette);
    // set sprite properties
    set_sprite_palette(0, 2, sprite_palette);

    sprite_clean(0);
    LETTER_COUNT = 0;
    clear_screen();
    print("PRESSiA\nTOiSTART\0", 56, 54);
    for(i = 0; i < LETTER_COUNT; ++i)
        set_sprite_prop(i, 0);
    SHOW_BKG;
    SHOW_SPRITES;
    DISPLAY_ON;
    delay(2000);
    while(1)
    {
        if(joypad() & J_A)
        {
            option = START;
            break;
        }
    }
}
```

CHAPTER 1 WELCOME TO HELL

```c
/**
 * level_1.c
 */

#include "level_1.h"
#include "../assets/level_assets/level_assets.h"
#include "../battle/battle.h"
#include "asakawa_battle.h"
#include "../text/text.h"
#include "../start_up/start_up.h"
#include "../assets/sprite_palette.h"
#include "../assets/bkg_palette.h"
#include "../assets/sprite_palette.h"

/* scene 3 variables */
UINT8 l1_scene_3_anim = 0;
UINT16 dt = 1500;

/* The big control function */
void level_1_ctrl(void)
{   /** this is a test **/
    set_bkg_palette(0, 1, bkg_palette);
    // set sprite properties
    set_sprite_palette(0, 2, sprite_palette);
    /** the test ends here **/

    wait_vbl_done();
    level_1_bkg_start();
    l1_scene_1();
    i = 0; // helping with animation
    while(enter_miner() == 0)
    {
        enter_miner();
        wait_vbl_done();
    }
    sprite_clean(0);
    LETTER_COUNT = 0;
    miner_intro_setup();
    miner_intro();
    l1_scene_3_anim = 0;
    l1_scene_3_setup();
    while(l1_scene_3_anim < 3)
    {
        l1_scene_3_animate();
        wait_vbl_done();
    }
    l1_scene_3_text_setup();
    l1_scene_3();
    enemy = 0;
    asakawa_battle_ctrl();
}

/**
 * SCENE 1
 */
/* setup the background for the opening screen. */
void level_1_bkg_start(void)
{
    DISPLAY_OFF;
    HIDE_WIN;
    HIDE_BKG;
    HIDE_SPRITES;
    sprite_clean(0);
    LETTER_COUNT = 0;
    SHOW_BKG;
    SHOW_SPRITES;
    DISPLAY_ON;
    /* opening screen */
    print("CHAPTERi\\\nWELCOME\nTOiHELL\0", 48, 64);
    delay(dt);
    sprite_clean(0);
    LETTER_COUNT = 0;

    DISPLAY_OFF;
    HIDE_WIN;
    HIDE_BKG;
    HIDE_SPRITES;

    SHOW_BKG;
    SHOW_SPRITES;
    DISPLAY_ON;

    /* setting up first set of text */
    bkg_clean();
    fisherman();
    print("WEkREiALL\nGOINkiTO\nHELLl\0", 24, 48);
    delay(dt);
    sprite_clean(10);
    LETTER_COUNT = 10;
    you_hear();
    delay(dt);
}

void l1_scene_1(void)
{
    while(talking)
    {
        if(joypad() & J_A)
        {
            ++talking;
            delay(200);
            if(talking == 2)
            {
                sprite_clean(10);
                LETTER_COUNT = 10;
                print("_iMONTHS\nON\0", 24, 48);
            }
            if(talking == 3)
            {
                sprite_clean(10);
                LETTER_COUNT = 10;
                print("KAMCHATKAS\nNORTHERN\nWATERSe\0", 24, 48);
            }
            if(talking == 4)
            {
                sprite_clean(10);
                LETTER_COUNT = 10;
                print("ITiWOULD\nBEiAiCOLD\nDEATH\neee\0", 24, 48);
            }
            if(talking == 5)
            {
                sprite_clean(10);
                LETTER_COUNT = 10;
                print("IiSEEiWE\nHAVEiA\nQUIETiONEe\0", 24, 48);
            }
            if(talking == 6)
            {
                sprite_clean(10);
                LETTER_COUNT = 10;
                print("CkMON\nIkLLiTAKE\nYOUiDOWN\0", 24, 48);
            }
            if(talking == 7)
            {
                sprite_clean(10);
                LETTER_COUNT = 10;
                print("TOiTHE\nSHITiPOTe\0", 24, 48);
            }
            if(talking == 8)
            {
                DISPLAY_OFF;
                // center on the door
                set_bkg_data(0,4,blank_screen_tiles);
                set_bkg_tiles(0,0,20,18,shit_pot);
                DISPLAY_ON;
                HIDE_SPRITES;// this turns sprites off
                sprite_clean(0);
                hide_sprites();
                LETTER_COUNT = 0;
                level_1_sprite_setup();
                break;
            }
        }
    }
}

int level_1_sprite_setup(void)
{
    /* hero on screen at door */
    hero_posx = 70;
    hero_posy = 65;

    /* fisherman on screen at door */
    fisherman_posx = 86;
    fisherman_posy = 65;

    /* miner positioning for the coming scene */
    miner_posx = 150;
    miner_posy = 100;

    SPRITES_8x16;
    /* hero */
    set_sprite_data(0, 8, hero_front_idle);
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    /* fisherman */
    set_sprite_data(8, 8, fisherman_front_idle);
    set_sprite_tile(2, 8);
    set_sprite_tile(3, 8);
    set_sprite_prop(3, S_FLIPX);

    /* miner */
    set_sprite_data(16, 8, miner_walk_left);

    /* display the hero */
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    /* display the fisherman */
    move_sprite(2, fisherman_posx, fisherman_posy);
    move_sprite(3, fisherman_posx+sprite_width, fisherman_posy);

    delay(400); /* a pause before appearing at door */
    SHOW_SPRITES;
}

/**
 * SCENE 2
 */
int enter_miner(void)
{
    UINT8 step_x = 4;
    UINT8 step_y = 0;

    if(miner_posx <= 80) /* if we are far enough to the left */
    {
        step_y = 4;
        step_x = 0;
        set_sprite_data(16, 8, miner_walk_up);
        if(miner_posy <= 90)
        {
            step_y = 0;
            set_sprite_data(16, 8, miner_idle_back);
            delay(100);

            return 1; /* we're done */
        }

        miner_posx -= step_x;
        miner_posy -= step_y;

        set_sprite_tile(4, 16+((i&0x1)*4));
        set_sprite_tile(5, 18+((i&0x1)*4));

        move_sprite(4, miner_posx, miner_posy);
        move_sprite(5, miner_posx+sprite_width, miner_posy);
        delay(80);
    }

    /* start by walking left */
    miner_posx -= step_x;
    miner_posy -= step_y;

    set_sprite_tile(4, 16+((i&0x1)*4));
    set_sprite_tile(5, 18+((i&0x1)*4));
    move_sprite(4, miner_posx, miner_posy);
    move_sprite(5, miner_posx+sprite_width, miner_posy);
    delay(80);

    ++i;

    return 0;
}

/* setting up the miners dialogue / introduction */
void miner_intro_setup(void)
{
    DISPLAY_OFF;
    HIDE_SPRITES;
    HIDE_WIN;
    HIDE_BKG;

    bkg_clean();

    SHOW_BKG;
    SHOW_SPRITES;
    DISPLAY_ON;

    miner();
    print("IiCOME\nFROMiTHE\nYUBARI\nCOAL\nMINESe\0", 24, 48);

}

/* setting up the miners dialogue / introduction */
int miner_intro(void)
{
    DISPLAY_ON;
    talking = 1;
    while(talking)
    {
        if(joypad() & J_A)
        {
            ++talking;
            delay(200);
            if(talking == 2)
            {
                sprite_clean(6);
                LETTER_COUNT = 6;
                print("WORKED\nTHEREiFOR\nSEVEN\nYEARSe\0", 24, 48);
            }
            if(talking == 3)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                fisherman();
                print("AiMINERf\nHUHn\0", 24, 48);
            }
```

```
                if(talking == 4)
                {
                    sprite_clean(10);
                    LETTER_COUNT = 10;
                    print("MUSTkVE\nBEENiNO\nMONEYiIN\nMININGn", 24, 48);
                }
                if(talking == 5)
                {
                    sprite_clean(0);
                    LETTER_COUNT = 0;
                    miner();
                    print("eeeTHERE\nWASiAN\nEXPLOSION\0", 24, 48);
                }
                if(talking == 6)
                {
                    sprite_clean(6);
                    LETTER_COUNT = 6;
                    print("eee\nFROMiTHE\nGASeee\0", 24, 48);
                }
                if(talking == 7)
                {
                    sprite_clean(6);
                    LETTER_COUNT= 6;
                    print("GOODiLIVES\nWEREiLOST\0", 24, 48);
                }
                if(talking == 8)
                {
                    sprite_clean(6);
                    LETTER_COUNT = 6;
                    print("THATiCANkT\nBE\nRETURNED\neee\0", 24, 48);
                }
                if(talking == 9)
                {
                    sprite_clean(0);
                    LETTER_COUNT = 0;
                    fisherman();
                    print("WEiBETTER\nGOl\0", 24, 48);
                }
                if(talking == 10)
                {
                    sprite_clean(10);
                    LETTER_COUNT = 10;
                    print("IiTHINK\nIiHEAR\nASAKAWA\nCOMINGl\0", 24, 48);
                }
                if(talking == 11)
                {
                    sprite_clean(0);
                    LETTER_COUNT = 0;
                    miner();
                    print("WHOkSiTHAT\nn\0", 24, 48);
                }
                if(talking == 12)
                    print("FISHERMANg\nTHE\nMANAGERl\0", 24, 80);
                if(talking == 13)
                    break;
            }
        }
}

/**
 * SCENE 3
 */
/**
 * first make a line of workers
 * with backs to the door
 */
void l1_scene_3_setup(void)
{
    HIDE_WIN;
    HIDE_BKG;
    HIDE_SPRITES;
    DISPLAY_OFF;

    /* set bkg up */
    set_bkg_data(0,4,blank_screen_tiles);
    set_bkg_tiles(0,0,20,18,shit_pot);

    /* hide unused sprites first */
    hide_sprites();

    /* hero back faced to the screen */
    hero_posx = 100;
    hero_posy = 100;

    /* fisherman back faced to the screen */
    fisherman_posx = 120;
    fisherman_posy = 100;

    /* miner back faced to the screen */
    miner_posx = 80;
    miner_posy = 100;

    /* student back faced to the screen */
    student_posx = 60;
    student_posy = 100;

    /* asakawa front in doorway */
    asakawa_posx = 75;
    asakawa_posy = 65;

    SPRITES_8x16;
    /* hero */
    set_sprite_data(0, 8, hero_back_idle);
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    /* fisherman */
    set_sprite_data(8, 8, fisherman_idle_back);
    set_sprite_tile(2, 8);
    set_sprite_tile(3, 10);
    /* miner */
    set_sprite_data(16, 8, miner_idle_back);
    set_sprite_tile(4, 16);
    set_sprite_tile(5, 18);
    /* student */
    set_sprite_data(24, 8, student_idle_back);
    set_sprite_tile(6, 24);
    set_sprite_tile(7, 26);
    /* asakawa */
    set_sprite_data(32, 8, asakawa_front_idle);
    set_sprite_tile(8, 32);
    set_sprite_tile(9, 34);

    /* display the hero */
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    /* display the fisherman */
    move_sprite(2, fisherman_posx, fisherman_posy);
    move_sprite(3, fisherman_posx+sprite_width, fisherman_posy);
    /* display the miner */
    move_sprite(4, miner_posx, miner_posy);
    move_sprite(5, miner_posx+sprite_width, miner_posy);
    /* display the student */
    move_sprite(6, student_posx, student_posy);
    move_sprite(7, student_posx+sprite_width, student_posy);

    delay(400); /* a pause before appearing at door */
    SHOW_SPRITES;
    SHOW_BKG;
    DISPLAY_ON;

    /* after a little wait, asakawa appears in door way */
    delay(400);
    move_sprite(8, asakawa_posx, asakawa_posy);
    move_sprite(9, asakawa_posx+sprite_width, asakawa_posy);
}

/* brief animation for scene 3 */
void l1_scene_3_animate(void)
{
    ++l1_scene_3_anim;
    /* hero */
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    /* fisherman */
    set_sprite_tile(2, 8);
    set_sprite_tile(3, 10);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    /* miner */
    set_sprite_tile(4, 16);
    set_sprite_tile(5, 18);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    /* student */
    set_sprite_tile(6, 24);
    set_sprite_tile(7, 26);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    /* asakawa */
    set_sprite_tile(8, 36-((l1_scene_3_anim&0x1)*4));
    set_sprite_tile(9, 38-((l1_scene_3_anim&0x1)*4));
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    delay(500);
}
/* scene_3 dialogue setup first */
void l1_scene_3_text_setup(void)
{
    DISPLAY_OFF;
    hide_sprites();
    bkg_clean();
    DISPLAY_ON;
    sprite_clean(0);
    LETTER_COUNT = 0;
    asakawa();
    print("LISTENiUP\nYOU\nMAGGOTSl\0", 24, 48);
}

/* asakawa's message */
void l1_scene_3(void)
{
    talking = 1;
    while(talking)
    {
        if(joypad() & J_A)
        {
            ++talking;
            delay(200);
            if(talking == 2)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("NEEDLESS\nTOiSAYf\0", 24, 48);
            }
            if(talking == 3)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("THISiSHIP\nISiOFiTHE", 24, 48);
            }
            if(talking == 4)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("UTMOST\nCONCERN\0", 24, 48);
            }
            if(talking == 5)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("AS\nPEOPLES\nOF\nIMPERIAL\nJAPANf\0", 24, 48);
            }
            if(talking == 6)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("WEiARE\nPROUD\nRIVALS\nOFiRUSSIAl\0",24, 48);
            }
            if(talking == 7)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("THEREkSiAN\nIMPERIAL\nSHIP\0", 24, 48);
            }
            if(talking == 8)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("TO\nPROTECTiUS\0", 24, 48);
            }
            if(talking == 9)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("ASiA\nNATIONiOF\nEXCELLENCE\0", 24, 48);
            }
            if(talking == 10)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("WEiARE\nUNMATCHED\0", 24, 48);
            }
            if(talking == 11)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("WHEN\nCOMPARED\nTOiOTHERSl\0", 24, 48);
            }
            if(talking == 12)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                print("FROMiTHE\nCROWD\nSOMEONE\nMURMURSf\0", 24, 32);
            }
            if(talking == 13)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                print("THATkS\nEXAGERATED\neee\0", 24, 32);
            }
            if(talking == 14)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                l1_scene_3_setup();
                asakawa_shoots_anim();
                DISPLAY_OFF;
                bkg_clean();
                hide_sprites();
                DISPLAY_ON;
                asakawa();
                print("THISiKIND\nOFiTALKiIS\0", 24, 48);
            }
            if(talking == 15)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("NOT\nTOLERABLE\0", 24, 48);
            }
            if(talking == 16)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("THIS\nATTITUDE\nIS\nMUTINOUSf\0", 24, 48);
            }
            if(talking == 17)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("ANDiWILL\nBE\nCONSIDERED\nTREASONl\0", 24, 48);
            }
            if(talking == 18)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                // Battle prompt
                DISPLAY_OFF;
                clear_screen();
                print("FIGHT\0", 64, 32);
                print("ASAKAWA\0", 56, 48);
                DISPLAY_ON;
                delay(100);
                DISPLAY_OFF;
                delay(100);
                DISPLAY_ON;
                delay(100);
                DISPLAY_OFF;
                delay(100);
                DISPLAY_ON;
                delay(300);
                break;
            }
        }
    }
}

void asakawa_shoots_anim(void)
{
    set_sprite_data(32, 32, asakawa_air_shot);
    anim_1 = 32;
    anim_2 = 34;
    /* asakawa shooting setup */
    for(i = 0; i < 7; ++i)
    {
        set_sprite_tile(8, anim_1+=4);
        set_sprite_tile(9, anim_2+=4);
        delay(500);
    }
    delay(500);
}
```

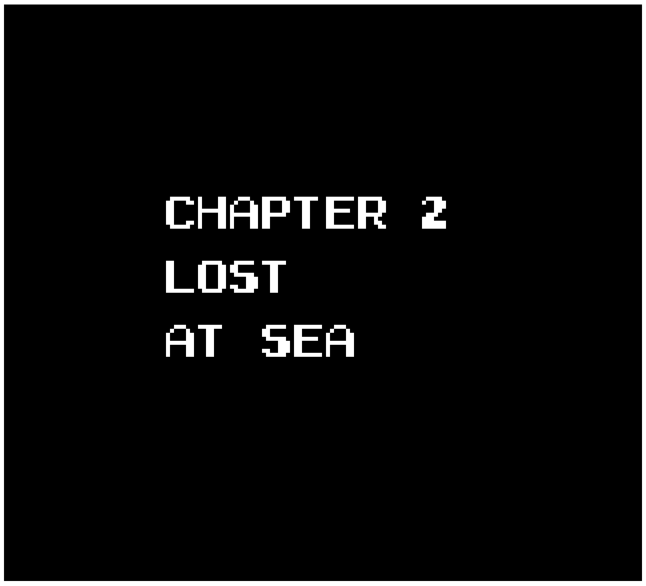
```
/**
 * asakawa_battle.c
 *
 * methods for handling battles with asakawa
 */

#include <stdio.h>
#include <stdlib.h>
#include "../text/text.h"
#include "../battle/battle.h"
#include "asakawa_battle.h"
#include "../assets/level_assets/level_assets.h"
#include "../start_up/start_up.h"

UINT8 state = BATTLE_CHOICE;

/* hp settings for this level */
UINT8 start_hp = 10;

/**
 * If level is:
 *  0) run first battle
 *  1) run second battle
 *  2) run third battle
 */
void asakawa_battle_ctrl(void)
{
    state = BATTLE_CHOICE;
    choice = 0;
    battle_menu();
    while(state != DEAD || option != GAME_OVER)
    {
        wait_vbl_done();
        battle_toggle_ctrl();
        while(state == FIGHTING)
        {
            choice_handler(arrow_y);
            fight(&hero_hp, &asakawa_hp);
            delay(400);
            // basically does back but only when everything is done
            if(state == BATTLE_WIN && REVOLUTION_2)
                break;
            if(state != DEAD)
            {
                DISPLAY_OFF;
                battle_menu();
                state = BATTLE_CHOICE;
                choice = 0;
                DISPLAY_ON;
                battle_menu();
            }
        }
        if((state == BATTLE_WIN) && REVOLUTION_2)
            break;
    }
    /* the battle that leads to level 2 */
    if(!revolt && !REVOLUTION_2)
        option = LEVEL_2;
}
```



```c
/**
 * level_2.c
 */

#include "level_2.h"
#include <stdio.h>
#include <stdlib.h>
#include "crab_catch.h"
#include "../assets/level_assets/level_assets.h"
#include "../text/text.h"
#include "../battle/battle.h"
#include "../start_up/start_up.h"
#include "../level_3/level_3.h"
#include "crab_catch.h"

/* save this for later when checking collisions with sprites */
UINT8 arr_size = 9;
UINT8 sprite_positions[] = {
    48, 64,
    64, 64,
    148,124,
    132, 64,
    16, 132,
    80,112 /* fisherman2_posx & fisherman2_posy */
};

UINT8 GOT_INFO = 0;
UINT8 moving = 1;
UINT8 left = 0;
UINT8 screen_x = 95;

void level_2_ctrl(void)
{
    wait_vbl_done();
    level_2_bkg_start();
    l2_scene_1();
    talking = 1;
    wait_vbl_done();
    l2_scene_1_fisherman_enter();
    fisherman_walk_away();
    while(moving)
    {
        option = LEVEL_2;
        wait_vbl_done();
        hero_walk();
        pos_check_shit_pot();
    }
    while(option != LEVEL_3)
    {
        GOT_INFO = 0;
        asakawa_enters_deck();
        asakawa_before_work();
        crab_catch_ctrl();
        asakawa_enters_deck();
        asakawa_after_work();
        delay(500);
        DISPLAY_OFF;
        shit_pot_setup();
        shit_pot_sprites();
        delay(500);
        DISPLAY_ON;
        moving = 1;
        if(option != LEVEL_3)
        {
            while(moving && option != LEVEL_3)
            {
                option = LEVEL_2;
                hero_walk();
                pos_check_shit_pot();
                if(conv_check())
                    break;
            }
        }
        caught_crabs = 0;
    }
    option = LEVEL_3;
}

void level_2_bkg_start(void)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    HIDE_SPRITES;
    HIDE_BKG;

    set_sprite_palette(0, 3, sprite_palette);
    /* Load the black clear screen */
    set_bkg_tiles(0,0,20,18, black_screen);

    SHOW_SPRITES;
    SHOW_BKG;
    delay(2000);
    print("CHAPTERi]\0", 48, 64);
    print("LOST\0", 48, 80);
    print("ATiSEA\0", 48, 96);
    for(i = 0; i < LETTER_COUNT; ++i)
        set_sprite_prop(i,1);
    delay(2000);
}

 void shit_pot_setup(void)
```

```
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    hide_sprites();
    set_bkg_data(0, 4, blank_screen_tiles);
    set_bkg_tiles(0,0,20,18,shit_pot);
}

void shit_pot_sprites(void)
{
    hero_posx = 80;
    hero_posy = 80;
    student_posx = 48;
    student_posy = 64;
    fisherman_posx = 64;
    fisherman_posy = 64;
    fisherman2_posx = 80;
    fisherman2_posy = 112;
    student2_posx = 148;
    student2_posy = 124;
    miner_posx = 132;
    miner_posy = 64;
    bed_posx = 16;
    bed_posy = 132;

    SPRITES_8x16;
    set_sprite_data(0, 8, hero_front_idle);
    set_sprite_data(8, 4, student_front_idle);
    set_sprite_data(12, 4, fisherman_front_idle);
    set_sprite_data(16, 4, miner_front_idle);
    set_sprite_data(20, 4, bed);
    // hero
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    // student
    set_sprite_tile(2, 8);
    set_sprite_tile(3, 8);
    set_sprite_prop(3, S_FLIPX);
    move_sprite(2, student_posx, student_posy);
    move_sprite(3, student_posx+sprite_width, student_posy);
    // student2
    set_sprite_tile(4, 8);
    set_sprite_tile(5, 8);
    set_sprite_prop(5, S_FLIPX);
    move_sprite(4, student2_posx, student2_posy);
    move_sprite(5, student2_posx+sprite_width, student2_posy);
    // fisherman
    set_sprite_tile(6, 12);
    set_sprite_tile(7, 12);
    set_sprite_prop(7, S_FLIPX);
    move_sprite(6, fisherman_posx, fisherman_posy);
    move_sprite(7, fisherman_posx+sprite_width, fisherman_posy);
    // miner
    set_sprite_tile(8, 16);
    set_sprite_tile(9, 16);
    set_sprite_prop(9, S_FLIPX);
    move_sprite(8, miner_posx, miner_posy);
    move_sprite(9, miner_posx+sprite_width, miner_posy);
    // bed
    set_sprite_tile(10, 20);
    set_sprite_tile(11, 22);
    move_sprite(10, bed_posx, bed_posy);
    move_sprite(11, bed_posx+sprite_width, bed_posy);

    if(((option == LEVEL_3) && (GOT_INFO) && (slept)) || appeared)
    {
        // fisherman2
        set_sprite_tile(12, 12);
        set_sprite_tile(13, 12);
        set_sprite_prop(13, S_FLIPX);
        move_sprite(12, fisherman2_posx, fisherman2_posy);
        move_sprite(13, fisherman2_posx+sprite_width, fisherman2_posy);
        appeared = 1;
        arr_size += 2;
    }

}

UINT8 sprite_collide_shit_pot(UINT8 *sprite_pos)
{
    for(i = 0; i < arr_size; i+=2)
    {
        if((hero_posx < (sprite_pos[i]+sprite_width)) &&
                ((hero_posx+sprite_width) > sprite_pos[i]) &&
                (hero_posy < (sprite_pos[i+1]+sprite_width)) &&
                ((hero_posy+sprite_width) > sprite_pos[i+1]))
        {
            hero_posx = sprite_pos[i]-8;
            hero_posy = sprite_pos[i+1]+8;
            move_sprite(0, hero_posx, hero_posy);
            move_sprite(1, hero_posx+sprite_width, hero_posy);
            return sprite_pos[i];
        }
    }
    return 0;
}

UINT8 conv_check(void)
{
    UINT8 sprite;
    UINT8 old_hero_posx;
    UINT8 old_hero_posy;
    if((sprite=sprite_collide_shit_pot(sprite_positions)) > 0)
    {

        // student in top left corner
        if(sprite == student_posx || sprite == fisherman_posx)
        {
            bkg_clean();
            old_hero_posx = hero_posx;
            old_hero_posy = hero_posy;
            if(!revolt)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                hide_sprites();
                workers();
                you_hear();
                delay(1000);
                if(option == LEVEL_2)
                {
                    sprite_clean(8);
                    LETTER_COUNT = 8;
                    print("AiFISHING\0", 24, 48);
                    print("BOATiWAS\0", 24, 64);
                    print("LOSTl\0", 24, 80);
                    delay(1000);
                    damn_that_asakawa();
                    GOT_INFO = 1;
                }
            }
            if(revolt)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                hide_sprites();
                workers();
                print("KEEPiUS\0",24, 48);
                print("OUTiOFiITl\0", 24, 64);
            }
        }
        // student in bottom right corner
        else if(sprite == student2_posx)
        {
            bkg_clean();
            old_hero_posx = hero_posx;
            old_hero_posy = hero_posy;
            sprite_clean(0);
            LETTER_COUNT = 0;
            hide_sprites();
            student();
            print("IiDONkT\0", 24, 48);
            print("WANTiTO\0", 24, 64);
            print("DIEiIN\0", 24, 80);
            print("KAMCHATKA\0", 24, 96);
            print("...\0",24,112);
        }
        else if(sprite == miner_posx)
        {
            bkg_clean();
            old_hero_posx = hero_posx;
            old_hero_posy = hero_posy;
            if(option == LEVEL_2)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                hide_sprites();
                miner();
                print("ITiWASiTHE\0", 24, 48);
                print("SAMEiAT\0", 24, 64);
                print("THEiMINESl\0", 24, 80);
            }
            if(option == LEVEL_3 && !revolt)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                hide_sprites();
                miner();
                print("YOUiHEARln\0", 24, 48);
                delay(500);
                sprite_clean(6);
                LETTER_COUNT = 6;
                print("THEiBOATkS\0", 24, 48);
                print("BACKl\0", 24, 64);
                GOT_INFO = 1;
            }
            else if(option == LEVEL_3 && revolt)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                miner();
                print("OFiCOURSE\0", 24, 48);
                print("IkLLiJOIN\0", 24, 64);
                print("THEiFIGHTl\0", 24, 80);
                talked = 1;
            }
        }
        else if(sprite == bed_posx)
        {
            sprite_clean(0);
            LETTER_COUNT = 0;
            hide_sprites();
            set_bkg_data(0, 4, blank_screen_tiles);
            set_bkg_tiles(0, 0, 20, 18, black_screen);
            print("SLEEPING\0", 56, 56);
            for(i = 0; i < LETTER_COUNT+3; ++i)
                set_sprite_prop(i, 1);
            delay(500);
            print("e\0", 75, 72);
            delay(500);
            print("e\0", 84, 72);
            delay(500);
            print("e\0", 92, 72);
            delay(800);
            sprite_clean(0);
            LETTER_COUNT = 0;
            print("PRESSiA\0", 56, 88);
            print("TOiWAKE\0", 56, 104);
            for(i = 0; i < LETTER_COUNT; ++i)
                set_sprite_prop(i, 1);
            slept = 1;
            if(option == LEVEL_2 && (!GOT_INFO));
            else if((option == LEVEL_2) && (GOT_INFO))
            {
                clear_screen();
                sprite_clean(0);
                LETTER_COUNT = 0;
                print("\0", 24, 32);
                option = LEVEL_3;
                return 1;
            }
        }
        else if((sprite == fisherman2_posx) && (appeared))
        {
            bkg_clean();
            old_hero_posx = hero_posx;
            old_hero_posy = hero_posy;
            if(!revolt)
            {
                talking = 1;
                sprite_clean(0);
                LETTER_COUNT = 0;
```

```
                    fisherman();
                    print("WEiWERE\nLOSTiAT\nSEAf\0", 24, 48);
                    while(talking)
                    {
                        if(joypad() & J_A)
                        {
                            ++talking;
                            delay(200);
                            if(talking == 2)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("ANDiWASHED\nASHOREiIN\nRUSSIAl\0", 24, 48);
                            }
                            if(talking == 3)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("THOSE\nRUSSKIES\0", 24, 48);
                            }
                            if(talking == 4)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("AREiONiTO\nSOMETHINGe\0", 24, 48);
                            }
                            if(talking == 5)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("WITHiTHEIR\nTALK\0", 24, 48);
                            }
                            if(talking == 6)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("ABOUT\nREVOLUTION\0", 24, 48);
                            }
                            if(talking == 7)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("WEiCAN\nORGANIZE\nTOOl\0", 24, 48);
                            }
                            if(talking == 8)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("YOUfiIf\nANDiA\nCOUPLE\nOTHERSl\0", 24, 48);
                            }
                            if(talking == 9)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("WEiCOULD\nORGANIZE\nTHEiCREWl\0", 24, 48);
                            }
                            if(talking == 10)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("WHATiDkYOU\nTHINKn\0", 24, 48);

                                print("AiYESl\nBiNOeee\0", 24, 88);
                                REVOLUTION_1 = 1;
                                break;
                            }
                        }
                    }
                }
                if(revolt && !talked)
                {
                    sprite_clean(0);
                    LETTER_COUNT = 0;
                    fisherman();
                    print("THEREkS\0", 24, 48);
                    print("NOTiENOUGH\0", 24, 64);
                    print("TALKiYETl\0", 24, 80);
                }
                if(revolt && talked)
                    REVOLUTION_1 = 1;

                if(talked)
                {
                    talking = 1;
                    sprite_clean(0);
                    LETTER_COUNT = 0;
                    fisherman();
                    print("WEkVEiGOT\nTHE\nNUMBERSl\0", 24, 48);
                    while(talking)
                    {
                        if(joypad() & J_A)
                        {
                            ++talking;
                            delay(200);
                            if(talking == 2)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("WITHOUT\nUS\0", 24, 48);
                            }
                            if(talking == 3)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("SHIPS\nWOULDNkT\nBUDGEe\0", 24, 48);
                            }
                            if(talking == 4)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("WITHOUT\nUSf\0", 24, 48);
                            }
                            if(talking == 5)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("THEiRICH\nWOULDNkT\nMAKEiA\nDIMEl\0", 24, 48);
                            }
                            if(talking == 6)
                            {
                                sprite_clean(10);
                                LETTER_COUNT = 10;
                                print("LETkSiGIVE\nkEMiHELL\nTOGETHERl\0", 24, 48);
                                striking = 1;
                                REVOLUTION_1 = 1;
                                break;
                            }
                        }
                    }
                }
            }
            // double check that we didn't mistakenly collide
            for(i = 0; i < 12; i+=2)
            {
                if(sprite_positions[i] == sprite)
                {
                    talking = 1;
                    break;
                }
                else
                    talking = 0;
            }
            while(talking)
            {
                if(!REVOLUTION_1)
                {
                    if(joypad() & J_A)
                    {
                        shit_pot_setup();
                        shit_pot_sprites();
                        hero_posx = old_hero_posx;
                        hero_posy = old_hero_posy;
                        move_sprite(0, hero_posx, hero_posy);
                        move_sprite(1, hero_posx+sprite_width, hero_posy);
                        talking = 0;
                    }
                }
                if(REVOLUTION_1) // prompt for revolution 1
                {
                    if(joypad() & J_A)
                    {
                        REVOLUTION_1 = 0;
                        health_loss = 0;
                        crabs_to_catch = 1;
                        if(revolt && talked)
                            return 1;
                        if(!revolt)
                        {
                            sprite_clean(0);
                            LETTER_COUNT = 0;
                            fisherman();
                            print("GOODl\0", 24, 48);
                            delay(500);
                            sprite_clean(10);
                            LETTER_COUNT = 10;
                            print("GOiSEEiWHO\0", 24, 48);
                            print("CANiHELPl\0", 24, 64);
                            delay(1000);
                            revolt = 1;
                        }
                    }
                    if(joypad() & J_B)
                    {
                    sprite_clean(0);
                    LETTER_COUNT = 0;
                    fisherman();
                    print("THATkSiTOO\0", 24, 48);
                    print("BADeee\0", 24, 64);
                    revolt = 0;
                    option = LEVEL_3;
                    REVOLUTION_1 = 0;
                    ++health_loss;
                    crabs_to_catch+=2;
                    delay(500);
                    REVOLUTION_1 = 0;
                }
            }
        }
    }
    return 0;
}

/**
 * Should probably make hero_pos and fisherman_pos
 * global at some point.
 */
void l2_scene_1(void)
{
    // waking up the hero
    sprite_clean(0);
    LETTER_COUNT = 0;
    delay(500);
    print("WAKEiUPl\0", 56, 75);
    for(i = 0; i < LETTER_COUNT; ++i)
        set_sprite_prop(i,1);
    delay(800);
    // reset sprite properties
    for(i = 0; i < LETTER_COUNT; ++i)
        set_sprite_prop(i, 1);
    // start the next part
    delay(1000);
    sprite_clean(0);
    LETTER_COUNT = 0;
    hide_sprites();

    hero_posx = 80;
    hero_posy = 75;
    fisherman_posx = 160;
    fisherman_posy = 75;

    // reset sprite properties back to normal
    for(i = 0; i < MAX_SPRITES; ++i)
        set_sprite_prop(i, 0);
    // hero lying down in shit pot
    shit_pot_setup();
    SPRITES_8x16;
    set_sprite_data(0, 4, hero_lie_down);
    set_sprite_data(4, 4, hero_front_idle);
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    // hero stands up
    delay(500);
    set_sprite_tile(0, 4);
    set_sprite_tile(1, 6);
    // setting up fisherman data for his walk in
    set_sprite_data(8, 8, fisherman_walk_side);
    set_sprite_tile(2, 8);
    set_sprite_tile(3, 10);
    /**
     * enter fisherman (the -5 adds a little extra distance)
     * i = 160 is the right side of the screen.
     */
    for(i = 0; fisherman_posx > 95; ++i)
    {
        delay(50);
        move_sprite(2, fisherman_posx, fisherman_posy);
        move_sprite(3, fisherman_posx+sprite_width, fisherman_posy);
        set_sprite_tile(2, (8+(4*(0x1&i))));
        set_sprite_tile(3, (10+(4*(0x1&i))));
        --fisherman_posx;
    }
    set_sprite_data(0, 4, hero_walk_sideways);
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);

    delay(500);
    DISPLAY_OFF;
    bkg_clean();
    DISPLAY_ON;
    sprite_clean(0);
    LETTER_COUNT = 0;
    fisherman();
    print("LETkSiGET\0", 24, 48);
    print("TOiWORKl\0", 24, 64);
}

void l2_scene_1_fisherman_enter(void)
{
    delay(1000);
    sprite_clean(10);
    LETTER_COUNT = 10;
    print("YOUkLL\0", 24, 48);
    print("NEEDiTHESE\0", 24, 64);
    delay(1000);
    sprite_clean(0);
    LETTER_COUNT = 0;
    DISPLAY_OFF;
    set_bkg_tiles(0,0,20,18,blank_screen);
    DISPLAY_ON;
    print("YOU\0", 72, 32);
    print("ACQUIREiA\0", 48, 48);
    print("CLUBiAND\0", 52, 64);
    print("FISHING\0", 56, 80);
    print("NET\0", 72, 96);
    items += 2;
    delay(1000);
    sprite_clean(0);
    LETTER_COUNT = 0;
    DISPLAY_OFF;
    bkg_clean();
    DISPLAY_ON;
    fisherman();
    print("NOWiLETkS\0", 24, 48);
    print("GETiGOINGl\0", 24, 64);
    while(talking)
    {
        if(joypad() & J_A)
            talking = 0;
    }
}

void fisherman_walk_away(void)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    hide_sprites();
    set_bkg_data(0, 4, blank_screen_tiles);
    set_bkg_tiles(0,0,20,18,shit_pot);

    // setting up fisherman data for his walk in
    SPRITES_8x16;
    set_sprite_data(0, 4, hero_walk_sideways);
    set_sprite_data(4, 4, fisherman_walk_up);

    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    set_sprite_tile(2, 4);
    set_sprite_tile(3, 6);

    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);
    move_sprite(2, fisherman_posx, fisherman_posy);
    move_sprite(3, fisherman_posx+sprite_width, fisherman_posy);

    /**
     * enter fisherman (the -5 adds a little extra distance)
     * i = 160 is the right side of the screen.
     */
    for(i = 0; fisherman_posy > 50; ++i)
    {
        delay(50);
        if(i&0x1)
        {
            set_sprite_prop(2, S_FLIPX);
            set_sprite_prop(3, S_FLIPX);
            move_sprite(2, fisherman_posx+sprite_width, fisherman_posy);
            move_sprite(3, fisherman_posx, fisherman_posy);
        }
        else
        {
            set_sprite_prop(2, S_FLIPX&0x0);
            set_sprite_prop(3, S_FLIPX&0x0);
            move_sprite(2, fisherman_posx, fisherman_posy);
            move_sprite(3, fisherman_posx+sprite_width, fisherman_posy);
        }
        --fisherman_posy;
    }
    // fisherman disappears
    fisherman_posx = 200;
    fisherman_posy = 200;
    move_sprite(2, fisherman_posx, fisherman_posy);
    move_sprite(3, fisherman_posx, fisherman_posy);
}

```

```
void hero_walk(void)
{
    if(joypad() & J_UP)
    {
        set_sprite_data(0, 4, hero_walk_up);
        set_sprite_tile(0, 0);
        set_sprite_tile(1, 2);
        // make sure sprites aren't flipped
        set_sprite_prop(0, S_FLIPX&0x0);
        set_sprite_prop(1, S_FLIPX&0x0);
        move_sprite(0, hero_posx, hero_posy);
        move_sprite(1, hero_posx+sprite_width, hero_posy);
        if((hero_posy+hero_posx)&0x1)
        {
            set_sprite_prop(0, S_FLIPX);
            set_sprite_prop(1, S_FLIPX);
            move_sprite(0, hero_posx+sprite_width, hero_posy);
            move_sprite(1, hero_posx, hero_posy);
        }
        else
        {
            set_sprite_prop(0, S_FLIPX&0x0);
            set_sprite_prop(1, S_FLIPX&0x0);
            move_sprite(0, hero_posx, hero_posy);
            move_sprite(1, hero_posx+sprite_width, hero_posy);
        }
        left = 0;
        delay(50);
        hero_posy-=3;
    }
    if(joypad() & J_DOWN)
    {
        set_sprite_data(0, 4, hero_walk_down);
        set_sprite_tile(0, 0);
        set_sprite_tile(1, 2);
        // make sure sprites aren't flipped
        set_sprite_prop(0, S_FLIPX&0x0);
        set_sprite_prop(1, S_FLIPX&0x0);
        move_sprite(0, hero_posx, hero_posy);
        move_sprite(1, hero_posx+sprite_width, hero_posy);
        if((hero_posx+hero_posy)&0x1)
        {
            set_sprite_prop(0, S_FLIPX);
            set_sprite_prop(1, S_FLIPX);
            move_sprite(0, hero_posx+sprite_width, hero_posy);
            move_sprite(1, hero_posx, hero_posy);
        }
        else
        {
            set_sprite_prop(0, S_FLIPX&0x0);
            set_sprite_prop(1, S_FLIPY&0x0);
            move_sprite(0, hero_posx, hero_posy);
            move_sprite(1, hero_posx+sprite_width, hero_posy);
        }
        left = 0;
        delay(50);
        hero_posy+=3;
    }
    if(joypad() & J_LEFT)
    {
        set_sprite_data(0, 8, hero_walk_sideways);
        if(!left)
        {
            set_sprite_prop(0, S_FLIPX);
            set_sprite_prop(1, S_FLIPX);
        }
        if((hero_posx+hero_posy)&0x1)
        {
            set_sprite_tile(0, 4);
            set_sprite_tile(1, 6);
        }
        else
        {
            set_sprite_tile(0, 0);
            set_sprite_tile(1, 2);
        }
        move_sprite(0, hero_posx+sprite_width, hero_posy);
        move_sprite(1, hero_posx, hero_posy);
        left = 1;
        delay(50);
        hero_posx-=3;
    }
    if(joypad() & J_RIGHT)
    {
        set_sprite_data(0, 8, hero_walk_sideways);
        // make sure the sprites aren't flipped
        set_sprite_prop(0, S_FLIPX&0x0);
        set_sprite_prop(1, S_FLIPX&0x0);
        if((hero_posx+hero_posy)&0x1)
        {
            set_sprite_tile(0, 4);
            set_sprite_tile(1, 6);
        }
        else
        {
            set_sprite_tile(0, 0);
            set_sprite_tile(1, 2);
        }
        move_sprite(0, hero_posx, hero_posy);
        move_sprite(1, hero_posx+sprite_width, hero_posy);
        left = 0;
        delay(50);
        hero_posx+=3;
    }
}

void asakawa_enters_deck(void)
{
    hero_posx = 80;
    hero_posy = 116;
    sprite_clean(0);
    LETTER_COUNT = 0;
    hide_sprites();
    set_bkg_data(0,4,blank_screen_tiles);
    set_bkg_tiles(0,0,32,18,deck);
    SPRITES_8x16;
    set_sprite_data(0, 8, hero_walk_sideways);
    set_sprite_tile(0, 0);
    set_sprite_tile(1, 2);
    set_sprite_prop(0, S_FLIPX&0x0);
    set_sprite_prop(1, S_FLIPX&0x0);
    move_sprite(0, hero_posx, hero_posy);
    move_sprite(1, hero_posx+sprite_width, hero_posy);

    fisherman_posx = 48;
    fisherman_posy = 128;

    set_sprite_data(8, 8, fisherman_walk_side);
    for(i = 2; i < 7; i+=2)
    {
        set_sprite_tile(i, 10);
        set_sprite_tile(i+1, 8);
        set_sprite_prop(i, S_FLIPX);
        set_sprite_prop(i+1, S_FLIPX);
    }
    move_sprite(2, fisherman_posx, fisherman_posy);
    move_sprite(3, fisherman_posx+sprite_width, fisherman_posy);
    move_sprite(4, fisherman_posx-20, fisherman_posy);
    move_sprite(5, fisherman_posx-20+sprite_width, fisherman_posy);
    move_sprite(6, fisherman_posx-40, fisherman_posy);
    move_sprite(7, fisherman_posx-40+sprite_width, fisherman_posy);

    // asakawa
    asakawa_posx = 156;
    asakawa_posy = 116;
    set_sprite_data(16, 8, asakawa_walk_side);
    set_sprite_tile(8, 16);
    set_sprite_tile(9, 18);
    move_sprite(8, asakawa_posx, asakawa_posy);
    move_sprite(9, asakawa_posx+sprite_width, asakawa_posy);
    while(asakawa_posx > 120)
    {
        delay(50);
        --asakawa_posx;
        set_sprite_tile(8, 16+(4*(asakawa_posx&0x1)));
        set_sprite_tile(9, 18+(4*(asakawa_posx&0x1)));
        move_sprite(8, asakawa_posx, asakawa_posy);
        move_sprite(9, asakawa_posx+sprite_width, asakawa_posy);
    }
    delay(500);
}

void asakawa_before_work(void)
{
    DISPLAY_OFF;
    bkg_clean();
    sprite_clean(0);
    LETTER_COUNT = 0;
    DISPLAY_ON;
    talking = 1;
    asakawa();
    print("GETiTHESE\0", 24, 48);
    print("CRABS\0", 24, 64);
    print("READYiTO\0", 24, 80);
    print("CANl\0", 24, 96);
    // asakawa's instructions on how to play
    while(talking)
    {
        if(joypad() & J_A)
        {
            ++talking;
            delay(200);
            if(talking == 2)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("CATCHiWITH\0", 24, 48);
                print("YOURiNETl\0", 24, 64);
            }
            if(talking == 3)
            {
                sprite_clean(8);
                LETTER_COUNT = 8;
                print("ANDiCRUSH\0", 24, 48);
                print("SHELLS\0", 24, 64);
                print("WITHiTHE\0", 24, 80);
                print("CLUBl\0", 24, 96);
            }
            if(talking == 4)
                talking = 0;
        }
    }
}

void asakawa_after_work(void)
{
    DISPLAY_OFF;
    bkg_clean();
    sprite_clean(0);
    LETTER_COUNT = 0;
    DISPLAY_ON;
    asakawa();
    print("BACKiDOWN\0", 24, 48);
    print("TOiTHE\0", 24, 64);
    print("SHITiPOTl\0", 24, 80);
    delay(1000);
}

void pos_check_shit_pot(void)
{
    if(hero_posy <= 56)
    {
        hero_posy = 56;
        if(hero_posx >= 48 && hero_posx <= 108)
        {
            if(revolt)
            {
                hero_posy = 56;
                hero_posx = hero_posx;
            }
            else
                moving = 0;
        }
    }
    if(hero_posx <= 8)
        hero_posx = 8;
    if(hero_posy >= 144)
        hero_posy = 144;
    if(hero_posx >= 152)
        hero_posx = 152;
}
```

```
/**
 * crab_catch.c
 *
 * methods for handling
 * crab battles
 */

#include <rand.h>
#include <stdlib.h>
#include "crab_catch.h"
#include "../level_1/asakawa_battle.h"
#include "../assets/level_assets/level_assets.h"
#include "../text/text.h"
#include "../assets/sprites/hero_back_idle.h"
#include "../start_up/start_up.h"
#include "../battle/battle.h"

UINT8 crab = 0;
UINT8 caught_crabs = 0;
UINT8 CRAB_CAUGHT = 0;
UINT8 CRAB_HP = 5;

/**
 * master method for a work day
 */
void crab_catch_ctrl(void)
{
    items = 2;
    battle_num = 1;
    state = BATTLE_CHOICE;
    /**
     * 1. opening screen chooses between two types of crabs & anounces
     *    their coming
     *       * king crab
     *       * normal crab
     * 2. then battle with that crab
     *       * battle relies on your ability to first catch,
     *       * then batter the crab.
     *       * if that sequence is not followed, it's a miss
     * 3. repeat crabs_to_catch many times.
     */
    while(caught_crabs < crabs_to_catch && state != DEAD &&
            option != GAME_OVER)
    {
        hero_hp=(10-health_loss);
        crab_catch_setup();
        delay(500);
        battle_menu();
        while(state != DEAD && state != BATTLE_WIN &&
                option != GAME_OVER)
        {
            wait_vbl_done();
            battle_toggle_ctrl();
            while(state == FIGHTING && option != GAME_OVER)
            {
                choice_handler(arrow_y);
                fight(&hero_hp, &CRAB_HP);
                delay(400);
                if(option == GAME_OVER || state == DEAD)
                    break;
                if(state != DEAD && state != BATTLE_WIN)
                {
                    DISPLAY_OFF;
                    battle_menu();
                    state = BATTLE_CHOICE;
                    choice = 0;
                    DISPLAY_ON;
                    battle_menu();
                }

            }
            if(option == GAME_OVER || state == DEAD)
                break;
        }
        if(state == BATTLE_WIN && option != GAME_OVER)
        {
            clear_screen();
            sprite_clean(0);
            LETTER_COUNT = 0;
            print("YOUiWINl", 56, 80);
            delay(500);
            CRAB_CAUGHT = 0;
            ++caught_crabs;
            state = BATTLE_CHOICE;
        }
    }
    delay(1000);
}

void crab_catch_setup(void)
{
    seed = DIV_REG;
    seed |= (UWORD)DIV_REG << 8;
    initarand(seed);

    /* which crab are we going to use? */
    crab = rand()&0x1;

    /* normal crab is chosen */
    if(crab == 0)
    {
        sprite_clean(0);
        LETTER_COUNT = 0;
        clear_screen();
        print("AiCRABl\0", 56, 80);
        delay(1000);
        sprite_setup(8, hero_back_idle, 8, norm_crab);
        CRAB_HP = 5;
        enemy = 1;
    }
    /* king crab is chosen */
    if(crab == 1)
    {
        sprite_clean(0);
        LETTER_COUNT = 0;
        clear_screen();
        print("AiKING\0", 56, 80);
        print("CRABl\0", 60, 96);
        delay(1000);
        sprite_setup(8, hero_back_idle, 8, king_crab);
        CRAB_HP = 10;
        enemy = 2;
    }
}
```

CHAPTER 3
REVOLUTION

```c
/**
 * level_3.c
 */

#include "level_3.h"
#include "../text/text.h"
#include "../level_1/asakawa_battle.h"
#include "../battle/battle.h"
#include "../level_1/level_1.h"
#include "../level_2/level_2.h"
#include "../level_2/crab_catch.h"
#include "../assets/level_assets/level_assets.h"
#include "../start_up/start_up.h"
#include "../assets/sprites/student_lie_down.h"
#include "../end/end.h"

void level_3_ctrl(void)
{
    option = LEVEL_3;
    GOT_INFO = 0;
    DISPLAY_OFF;
    clear_screen();
    delay(500);
    sprite_clean(0);
    LETTER_COUNT = 0;
    print("CHAPTERi^\nREVOLUTION\0", 48, 56);
    DISPLAY_ON;
    delay(2000);
    DISPLAY_OFF;
    shit_pot_setup();
    shit_pot_sprites();
    DISPLAY_ON;
    moving = 1;
    while(!striking)
    {
        option = LEVEL_3;
        while(moving)
        {
            option = LEVEL_3;
            hero_walk();
            pos_check_shit_pot();
            if(conv_check())
                break;
        }
        if(!striking)
            leaves_shit_pot();
    }
    delay(1000);
    DISPLAY_OFF;
    sprite_clean(0);
    LETTER_COUNT = 0;
    shit_pot_setup();
    shit_pot_sprites();
    DISPLAY_ON;

    /* asakawa shooting */
    anim_1 = 32;
    anim_2 = 34;
    asakawa_posx = 75;
    asakawa_posy = 65;
    set_sprite_data(32, 32, asakawa_air_shot);
    set_sprite_tile(14, anim_1);
    set_sprite_tile(15, anim_2);
    move_sprite(14, asakawa_posx, asakawa_posy);
    move_sprite(15, asakawa_posx+sprite_width, asakawa_posy);
    for(i = 0; i < 7; ++i)
    {
        set_sprite_tile(14, anim_1+=4);
        set_sprite_tile(15, anim_2+=4);
        delay(500);
    }
    state = BATTLE_CHOICE; // this is for tests
    delay(500);
    DISPLAY_OFF;
    bkg_clean();
    sprite_clean(0);
    LETTER_COUNT = 0;
    DISPLAY_ON;
    asakawa();
    print("IkVEiHEARD\0", 24, 48);
    print("TALKiOFiA\0", 24, 64);
    print("STRIKEl\0", 24, 80);
    delay(1000);
    sprite_clean(8);
    LETTER_COUNT = 8;
    asakawa();
    print("SHOWiME\0", 24, 48);
    print("YOUR\0", 24, 64);
    print("LEADERSl\0", 24, 80);
    delay(1000);
    DISPLAY_OFF;
    /* happens in level 1 too */
    clear_screen();
    sprite_clean(0);
    LETTER_COUNT = 0;
    print("FIGHT\0", 64, 32);
    print("ASAKAWA\0", 56, 48);
    DISPLAY_ON;
    enemy = 0;
      items = 2;
      hero_hp = 50;
      asakawa_hp = 200; /* this will later go back down to 100 */
      start_hp = 50;
      sprite_clean(0);
      LETTER_COUNT = 0;
      state = BATTLE_CHOICE;
      option = LEVEL_3;
      asakawa_battle_ctrl();
      state = BATTLE_CHOICE;
      option = LEVEL_3;
      bkg_clean();
      sprite_clean(0);
      LETTER_COUNT = 0;
      asakawa();
      print("NOWiTAKE\0", 24, 48);
      print("THEMiAWAYl\0", 24, 64);
      delay(1000);
      DISPLAY_OFF;
      set_bkg_data(0, 4, blank_screen_tiles);
      set_bkg_tiles(0, 0, 20, 18, black_screen);
      DISPLAY_ON;
      delay(2000);
      DISPLAY_OFF;
      SPRITES_8x16;
      shit_pot_setup();
      shit_pot_sprites();

      team_battle_show();

      set_sprite_data(24, 4, student_lie_down);
      set_sprite_tile(4, 24);
      set_sprite_tile(5, 26);
      move_sprite(4, student2_posx, student2_posy);
      move_sprite(5, student2_posx+sprite_width, student2_posy);
      DISPLAY_ON;

      delay(2000);
      bkg_clean();
      sprite_clean(0);
      LETTER_COUNT = 0;
      workers();
      print("ITkSi\nBERIBERIe\0", 24, 48);
      talking = 1;
      while(talking)
      {
          if(joypad() & J_A)
          {
              ++talking;
              delay(200);
              if(talking == 2)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  damn_that_asakawa();
              }
              if(talking == 3)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  print("NOBODYkS\nONiOUR\nSIDEf\0", 24, 48);
              }
              if(talking == 4)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  print("EXCEPTiOUR\nSELVESe\0", 24, 48);
              }
              if(talking == 5)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  print("WEiALL\nSHOULDkVE\nACTED\nTOGETHERl\0", 24, 48);
              }
              if(talking == 6)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  print("THEREkDiBE\nNOiONEiTO\nWORK\0", 24, 48);
              }
              if(talking == 7)
              {
                  sprite_clean(8);
                  LETTER_COUNT= 8;
                  print("IF\nTHEYiTOOK\nUSiALLl\0", 24, 48);
              }
              if(talking == 8)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  print("LETS\nDOiIT\nAGAINf\0", 24, 48);
              }
              if(talking == 9)
              {
                  sprite_clean(8);
                  LETTER_COUNT = 8;
                  print("ONEiMORE\nTIMEl\0", 24, 48);
              }
              if(talking == 10)
              {
                  REVOLUTION_2 = 1;
                  enemy = 0;
                  items = 2;
                  hero_hp = 100;
                  /* this will later go back down to 100 */
                  asakawa_hp = 100;
                  start_hp = 100;
                  option = LEVEL_3;
                  state = BATTLE_CHOICE;
                  break;
              }
          }
      }
      while(1)
      {
          asakawa_battle_ctrl();
          if(state == BATTLE_WIN)
              break;
          if(option == GAME_OVER || state == DEAD)
          {
              option = LEVEL_3;
              state = BATTLE_CHOICE;
              hero_hp = 100;
          }

      }
      ending();
  }

  void team_battle_show(void)
  {
      hero_posx = 200;
      hero_posy = 200;
      fisherman2_posx = 200;
      fisherman2_posy = 200;
      miner_posx = 200;
      miner_posy = 200;
      move_sprite(0, hero_posx, hero_posy);
      move_sprite(1, hero_posx+sprite_width, hero_posy);
      move_sprite(8, miner_posx, miner_posy);
      move_sprite(9, miner_posx+sprite_width, miner_posy);
      move_sprite(12, fisherman2_posx, fisherman2_posy);
      move_sprite(13, fisherman2_posx+sprite_width, fisherman2_posy);
      fisherman_posy = student2_posy-20;
      student_posy = fisherman_posy;
      fisherman_posx = student2_posx;
      student_posx = student2_posx-20;
      move_sprite(2, student_posx, student_posy);
      move_sprite(3, student_posx+sprite_width, student_posy);
      move_sprite(6, fisherman_posx, fisherman_posy);
      move_sprite(7, fisherman_posx+sprite_width, fisherman_posy);
  }

  void leaves_shit_pot(void)
  {
      sprite_clean(0);
      LETTER_COUNT = 0;
      asakawa_enters_deck();
      asakawa_before_work();
      crab_catch_ctrl();
      if(state == DEAD)
          return;
      asakawa_enters_deck();
      asakawa_after_work();
      delay(500);
      DISPLAY_OFF;
      shit_pot_setup();
      shit_pot_sprites();
      delay(500);
      DISPLAY_ON;

      moving = 1;
      GOT_INFO = 0;
      slept = 0;
      caught_crabs = 0;
  }
```

```
/**
 * end.c
 *
 * THE END
 */

#include <gb/gb.h>
#include "../text/text.h"
```

```
#include "../assets/level_assets/level_assets.h"
#include "../battle/battle.h"
#include "../assets/sh_tiles.h"
#include "../assets/sickle_hammer.h"

void ending(void)
{
    sprite_clean(0);
    LETTER_COUNT = 0;
    bkg_clean();
    delay(500);
    talking = 1;
    print("THE\nPROLETARO\nHAVE\nNOTHING\nTOiLOSE", 24, 32);
    while(talking)
    {
        if(joypad() & J_A)
        {
            ++talking;
            delay(200);
            if(talking == 2)
            {
                sprite_clean(0);
                LETTER_COUNT = 0;
                print("BUTiTHEIR\nCHAINSe", 24, 32);
            }
            delay(1000);
            clear_screen();
            delay(100);
            bkg_clean();
            delay(100);
            clear_screen();
            delay(100);
            bkg_clean();
            delay(100);
            clear_screen();
            delay(2000);
            sprite_clean(0);
            LETTER_COUNT = 0;
            print("KANI\nKOUSEN", 60, 96);
            set_bkg_data(0, 18, sh_tiles);
            set_bkg_tiles(0, 0, 20, 18, sickle_hammer);
            break;
        }
    }
}
```

Bibliography

Bowen-Struyk, Heather. *Rethinking Japanese Proletarian Literature*. Ann Arbor: The University of Michigan, 2001.

DP. "GameBoy CPU Manual." Accessed April 28, 2019. http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf.

Field, Norma. "Commercial Appetite and Human Need: The Accidental and Fated Revival of Kobayashi Takiji's *Cannery Ship*." *The Asia-Pacific Journal*, vol. 8-8-09 (February 22, 2009).

Hope, Michael. "GameBoy Developers Kit." Last modified 2001/02/28. http://gbdk.sourceforge.net/.

Huizinga, John. *Homo Ludens: A Study of the Play Element in Culture*. London: Roy Publishers, 1950.

Kain, Amamiya. *Suffering Forces Us to Think beyond the Right-Left Barrier*. Edited by Frency Lunning. Translated by Jodie Beck. Minneapolis: University of Minnesota Press, 2010.

Marx, Karl. *The Marx-Engels Reader*. Edited by Robert C. Tucker. New York: Norton & Company, 1978.

McLuhan, Marshall. *Understanding Media: The Extensions of Man*. New York: McGraw-Hill Book Company, 1964.

Nintendo. "Company History." Nintendo - Corporate Information | Company History. 2019. Accessed April 29, 2019. https://www.nintendo.com/corp/history.jsp.

Reynolds, Daniel. "The Vitruvian Thumb: Embodied Branding and Lateral Thinking with the Nintendo Game Boy." *Game Studies* volume 16, Issue 1 (October 2016).

Shockey, Nathan. *The Typographic Imagination: Reading and Writing in Japan's Age of Modern Print Media*. New York: Columbia University Press, 2019.

Takiji, Kobayashi. *The Crab Cannery Ship*. Translated by. Zeljko Cipris. Honolulu: University of Hawai'i Press, 2013.

Takiji, Kobayashi. *"Kani Kôsen" Kobayashi Takiji no Sekai*. Tôkyô: Shirakaba Bungaku-kan Takiji Raiburarī, 2008.

Yôichi, Komori. "Introduction." *The Crab Cannery Ship*. Translated by Željko Cipriŝ. Honolulu: University of Hawai'i Press, 2013.

Ludography

Square Enix. (2007). *Final Fantasy Tactics A2: Grimoire of the Rift* [Nintendo DS], Kyoto: Nintendo.