Caleb Johnson

EE3752-5

Fall 2018

**Project Report**

**Introduction**

This project is a text-based RPG/choose your own adventure inspired by tabletop RPGs that runs in the Keil uVision IDE. It applies to what we’ve learned in class/in the labs by using the UART for input and output, converting numbers between ASCII and binary, and implementing various subroutines.

**Background**

To understand this project on a technical level, the reader will have to have an understanding of ARM assembly and how to use the LPC2104 UART for input and output. The user should also understand the purpose of an RNG algorithm, to take an input number, perform operations on it, and output another number which isn’t even close to the original number. Besides that, the reader should understand that tabletop RPGs usually work by selecting an action to take, determining what skill would be used for that action, rolling a d20 (20 sided die), and adding your bonus to that skill to the result of the die roll. The final thing the reader should know is that DC is a number which must be met or exceeded by the roll + bonus for an action to succeed. Before being able to start the project, I did research into how to create an effective RNG algorithm and referenced my old labs. As this project was completely digital within the Keil uVision IDE, no materials were needed.

**Method**

I began the project without coding at all, simply deciding how the game should be designed. I first decided how it would work, then wrote a series of obstacles and choices for the player to encounter. Once I decided each choice, I wrote dialog for every choice option, the choice itself, what happens when the choice is succeeded, what happens when the choice fails, and more. I wrote everything down in a Google Docs document, which will be pasted here:

**Heist RPG**

Upon game open, player is given three options, which decide the total points they can have:

Easy- 30 (Average +5)

Medium- 24 (Average +4)

Hard- 18 (Average +3)

Player can then choose values for their skills. A skill value must be between -10 and 10 inclusive, and the sum of all skill values cannot exceed the number of points allowed by their difficulty. There are 6 skills:

Fighting

Stealth

Hacking

Lockpicking

Charisma

Luck

Whenever an action is attempted, a roll is made with a result from 1-20. The player’s skill bonus for that skill is then added. If the result matches or exceeds the action’s DC (Difficulty Class,) the action is successful. Otherwise the action fails. There are three possible results of failing an action:

1. Nothing happens, but the option becomes unavailable.
2. The player must engage in combat.
3. A game over is initiated.

A fighting check can be initiated either by choosing to fight or by having to. If you choose to fight, the DC is -3.

There will be 5 obstacles in the game:

1 Front Door

Try the door and hope it opens [Luck] DC 8 Nothing

Attempt to hack the door [Hacking] DC 5 Game Over

Press the call button and convince the person to let you in [Charisma] DC 5 Game Over

2 Camera

Find blind spots and sneak past [Stealth] DC10 Game Over

Unlock the camera’s control panel and disable it [Lockpicking] DC13 Nothing

Hack into the camera and feed it a loop [Hacking] DC13 Nothing

3 Guard

Sneak past [Stealth] DC15 Fight

Act like you belong [Charisma] DC15 Fight

Fight him! [Fighting] DC15, DC18 Game Over

4 Door

Try door and hope it opens [Luck] DC15 Nothing

Unlock door [Lockpicking] DC15 Nothing

Bash door open [Fighting] DC15 Nothing

Note: If all 3 options above are exhausted, then game over is initiated.

5 Final Guard

Hope he suddenly drops dead from a heart attack [Luck] DC25 Nothing

Convince the guard you work for the bank [Charisma] DC20 Fight

Sneak and pickpocket his keycard on the way [Stealth] DC20 Fight

Sneak past to door to vault [Stealth] DC17 Fight

Unlock door [Lockpicking] DC17 Nothing

Hack door [Hacking] DC17 Nothing

Note: If both above options are exhausted, will have to fight.

Fight him!!! [Fighting] DC20, DC23 Game Over

**Dialogue**

Note: x-y-s/f/c/h = Obstacle x, option y, success/fail/choose to fight/have to fight

1 You approach the back door of the bank. The door has no mechanical lock, but just an electrical one. By the door is an RFID scanner bank employees can use to get in. It also has a button and a speaker, so you can call the person controlling the door and ask to be let in.

1-1 Just to see, you try the door.

1-1-s For some reason, the door is unlocked. You walk into the bank.

1-2-f As expected, the door is locked.

1-2 You jack into the RFID scanner and attempt to hack it.

1-1-s The security on this door is bad. Looking through its code, you change bit is\_locked from a 1 to a 0, unlocking the door.

1-1-f The door is easy to crack, but while messing around with it you accidentally change the bit intruder\_detected from a 0 to a 1, alerting the whole bank. You attempt to make an escape, but are caught and arrested. GAME OVER.

1-3 You press the call button.

1-1-s You tell the person controlling the door that you’re a bank employee who left his wallet at his desk. He sounds as if he’s not paying too much attention and is watching soap operas at his desk, so he lets you in.

1-1-f You try to think of something to say. You end up saying “Lovely night for a bank robbery, huh?” and although it sounds like the person controlling the door wasn’t paying too much attention, he sounds the alarm. You attempt to make an escape, but are caught and arrested. GAME OVER.

2 You walk into the bank. As you found the blueprints for the building, you know where to go. Rather than taking a left and going to the employee area, you take a right and walk towards the executives-only vault. As you walk, you see a security camera with a motion detector attached.

2-1 You attempt to look for blind spots in the camera’s view and when the moment is right, go between cover to avoid it and the motion detector gaze.

2-1-s Having successfully gotten past it, you continue down the hall.

2-1-f You mess up the timing, and end up in the camera’s view. An alarm is sounded. You attempt to make an escape, but are caught and arrested. GAME OVER.

2-2 Seeing a nearby electrical panel with a padlock, you attempt to unlock it so that you may potentially disable the camera.

2-2-s You unlock the control panel, and look inside. Taking a guess, you flip one of the switches to an off position. The camera and the lights in that section of the hallway go out, but other areas seem unaffected. You continue down the hall.

2-2-f The padlock is high quality, and you can’t crack it.

2-3 The camera appears to have wireless capabilities. You pull out your phone and attempt to connect to it.

2-2-s You successfully connect to the camera, and feed it a loop, as well as disabling the motion detector. You continue down the hall.

2-2-f The encryption on the camera’s wireless signal is too complex, and you fail to connect to it.

3 At an intersection of several hallways, there sits a guard in a chair. He’s reading a novel, but appears to still be looking up and and watching fairly often.

3-1 You attempt to sneak past the guard. You hide behind a plant, and wait for the perfect moment.Immediately after the guard looks up, around, then down at his book again, you make your move.

3-1-s Staying low and as far as possible to keep out of his peripheral vision, you move past, taking a right turn and making your way towards the vault.

3-1-f Not being very light of foot, you accidentally kick the wall as you attempt to make a turn down the hallway. The guard looks up, shocked, and draws his baton.

3-2 You decide to act like you belong. You put on your best brave face, and stroll confidently toward the intersection. The guard looks up at you.

3-2-s Attempting to impersonate the executives allowed in this area, you smile and give him a greeting. “Just making a deposit,” you say. The guard, seemingly satisfied, goes back to his book.

3-2-f In this intense situation, you can’t help but feel nervous. The guard gives you a look, and you can feel your face losing its composure. You offer up a shaky “Oh, hi, uh, just, y’know, passing through,” but the guard seems unconvinced. He stands up and draws his baton.

3-3-c You walk right up to the guard, and get in a fighting stance. Shocked, he stands up and draws his baton.

3-3-h The guard approaches with his baton, and begins swinging.

3-3-s Using what you know of boxing, you skillfully dodge his swings and manage to hit him until he’s knocked out. With the threat gone, you take a right and continue towards the vault.

3-3-f You attempt to dodge his attacks, but he is too fast. He swings his baton, hitting you repeatedly, until you can fight no more. He calls for backup, and you are arrested. GAME OVER.

4 You’re now approaching the vault. You come to a large metal door, and from your blueprints you know the vault is on the other side.  The door has no electronic locks, just mechanical.

4-1 You check to see if the door is unlocked.

4-1-s For some reason, it is. You enter the room that contains the vault.

4-1-f Of course it’s locked, why wouldn’t it be?

4-2 You try to pick the lock on the door.

4-2-s You manage to get the tumblers just right, and you hear the satisfying click of the door unlocking. You open it and step into the room that contains the vault.

4-2-f The lock on this door is too well made, and you can’t quite get it.

4-3 You attempt to bash the door down with a well-placed kick.

4-3-s Summoning all your strength, you manage to kick the door so that it opens before you. You enter the room that contains the vault.

4-3-f The door is much too tough, and all you manage to do is sprain your ankle.

4-f You exhaust all your options to open this door. You came this far, but a metal door has thoroughly defeated you. Head hung in shame, you give up and go home. GAME OVER.

5 You enter the room containing the vault. In here is a large guard. He looks much stronger and more alert than the last one. Behind the guard is the vault door.

5-1 You hope with all your heart that this guard suddenly has a heart attack.

5-1-s You can’t believe your eyes, but before you, the guard suddenly drops to the ground, clutching his chest. Still barely believing this is happening, you walk up to him and grab his key card from his belt. You use his key card to open the vault door.

5-1-f Well that was a waste of time.

5-2 You walk right up to the guard, and attempt to convince him you belong here.

5-2-s The guard is wary when you tell him you’re a bank executive. You see him reaching for his stun gun at his side. You feign anger, and threaten that the other executives will hear about this. He pauses. You pull out your phone, saying that you’ll call his bosses right now and get him fired for not letting you in. The guard apologizes profusely for not recognizing you, and allows you entry into the vault. You smile and thank him.

5-2-f Your walk lacks confidence, and in that slight moment of hesitation the guard knows you shouldn’t be here. He draws his stun gun, and begins firing at you.

5-3 You decide stealth is the best option, and manage to stay out of the way of his watchful gaze. As you sneak past, however, you realize how much easier it would be to get into the vault door with his key card.

5-3-s Using the steadiest hands you’ve ever had, you manage to get your hands around the key card and lift it from his belt without him noticing. Using the keycard, you enter the vault.

5-3-f As you reach towards his key card, his hand suddenly shoots out and grabs your arm. He gives you a look that tells you that he knew you were here long before you tried to grab his card. He draws his stun gun, and begins firing at you.

5-4 You decide stealth is the best option, and attempt to stay in the shadows out of his watchful gaze.

5-4-s You get past him to the vault door, but the fight is still not over. You now must open the door without setting off an alarm.

5-4-s-1 You attempt to use your lockpicking skills to open the vault door.

5-4-s-1-s Although this is probably the hardest lock you’ve ever tried at, you successfully get the door open, and the path into the vault lies open. You enter it.

5-4-s-1-f As you try to pick the lock, you accidentally move a tumbler to a position it shouldn’t be in, and an alarm sounds. You look behind you to see the guard, already turning around and raising his stun gun towards you.

5-4-s-2 You attempt to hack into the door to release its maglocks.

5-4-s-2-s The security is tough, but you manage to crack it. Getting access to the system, you release the maglocks, allowing you to open the door. You open the door to the vault and enter.

5-4-s-2-f While trying to hack the door’s controls, you accidentally trip an alarm. As it starts blaring you look behind you to see the guard, already turning around and raising his stun gun towards you.

5-4-f Your footfalls are light, but this guard’s ears are unbelievably good. He suddenly turns and stares directly at you, stun gun raised towards you.

5-5-c You step out and get into a fighting stance, challenging this guard to a one on one fight. Seemingly liking this challenge, he smiles, and draws his stun gun.

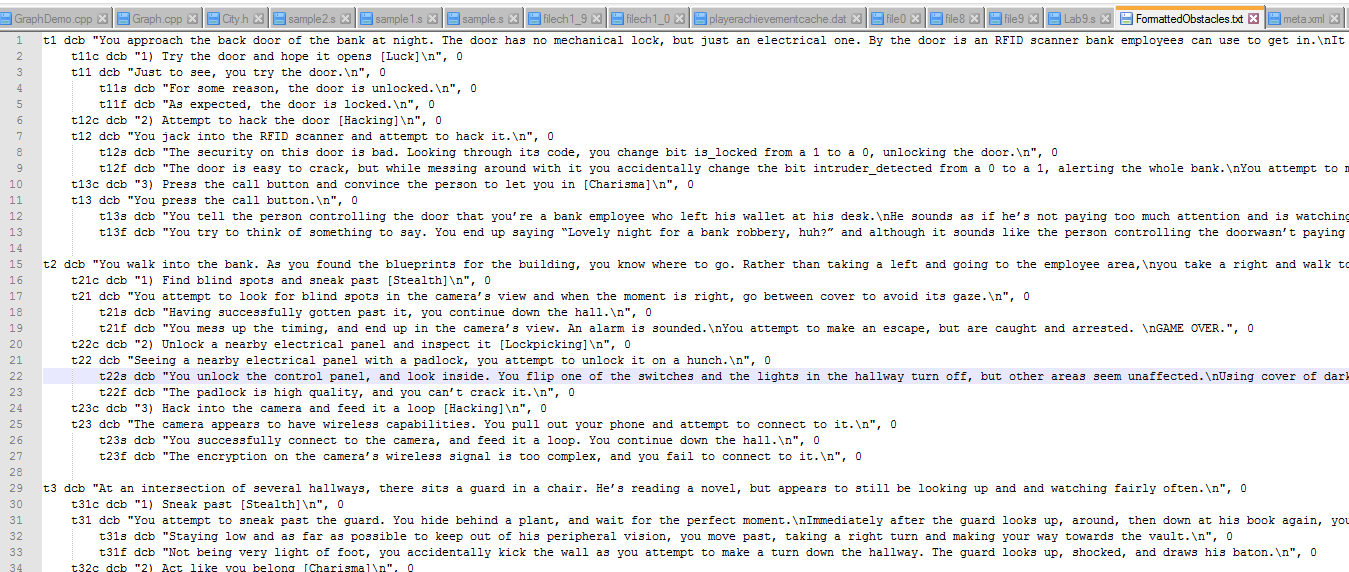
5-5-h The guard fires his stun gun at you, and you try to dodge and get in close to knock him out.

5-5-s The guard is tough and fast, but you’re tougher and faster. You knock him unconscious. Seeing the key card in his belt, you take it and use it to open the vault door. You enter.

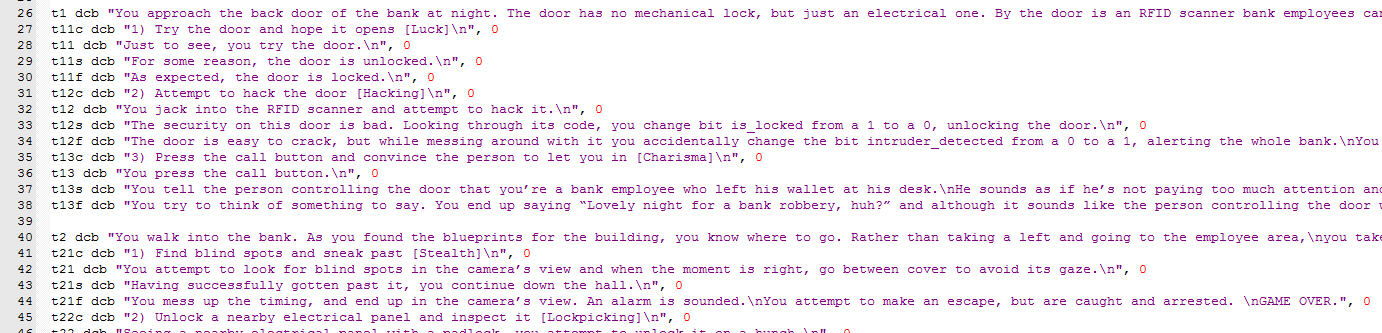
5-5-f You’re not fast enough, and the guard hits you directly in the chest with the stun gun. You fall to the ground, seizing, and are quickly handcuffed by him. You go to jail. GAME OVER.

5-s In the vault is more money than you could ever imagine. You pack your duffle bags until they can fit no more, but even 1/10th of what you could carry would have you set for life. Satisfied with a job well done, you make your way out of the bank. GAME WIN.

As you can see, the non-coding phase of this project was not insignificant, and took a few hours by itself. I then formatted the text from above in Notepad++ in a way that it was almost ARM compilable, but still human readable.



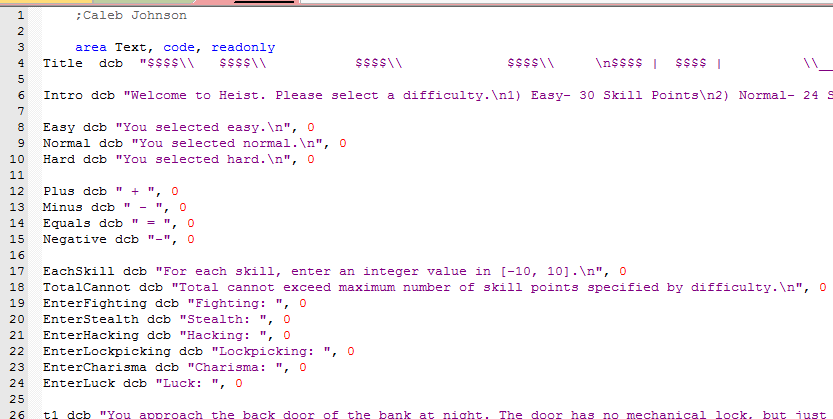
Then I copy/pasted that file into a .s file, and use shift-tab to remove the tabs to make it compilable. That way when I wanted to edit some text, I could edit it in the more readable format then copy it into the .s file.



And I did have to edit the text many times. Rewording sentences, changing the location of /n characters, fixing typos, etc. The biggest change was that, in the original text, the camera had a motion detector attached, but I decided to remove this in the final build to make the electrical panel choice make more sense.

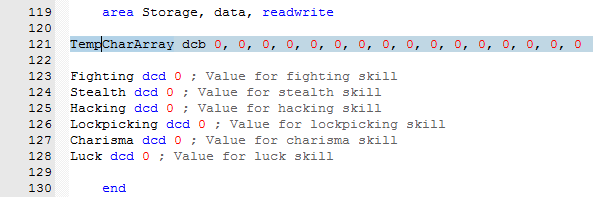
As for the gameplay itself, the thought process during project conception was to create something similar to a tabletop RPG. The player encounters an obstacle, they decide on a way to get past the obstacle, and make a roll to determine the outcome of that choice. Originally, I also planned to implement a combat system, rather than just a fighting skill roll, as many tabletop RPGs have combat systems. However, I eventually decided against this, as I felt it detracted from the core gameplay and felt out of place.

Now, we can move to the technical part of the design. The first thing I needed was a file that could contain all the memory I needed for the program. First this file contains a readonly area which has all the text in the game:



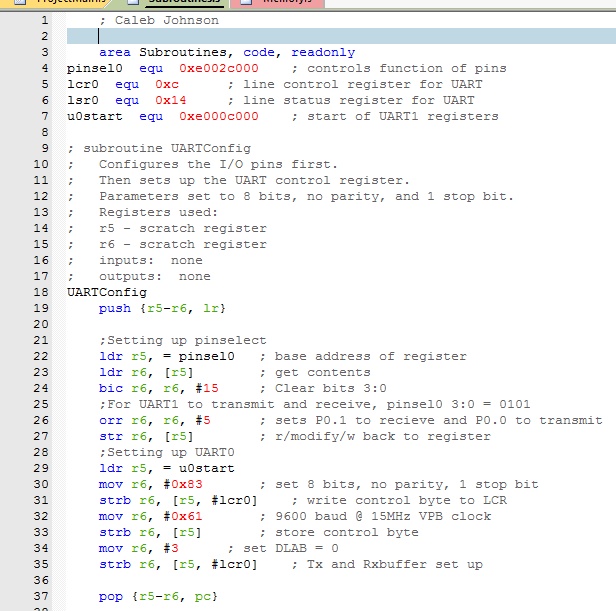
The title is an ASCII title art of “HEIST” with dollar signs, formatted so it will output correctly.

Next, I needed a readwrite area to contain saved memory. I ended up only needing a few things in this: A byte array for saving chars as the player types the, and 6 words for each skill.

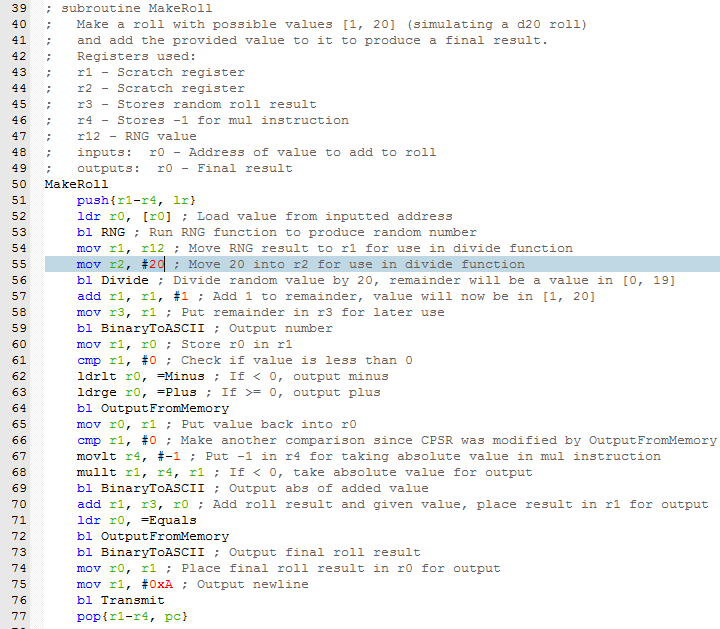


Now, I needed another file containing all my subroutines. This was actually the hardest part of the program, even though many subroutines were adapted or used from past labs. I will go through each subroutine.

The first subroutine is to configure and set up the UART0 to receive and transmit, very similar to subroutines seen in labs 8 and 9.



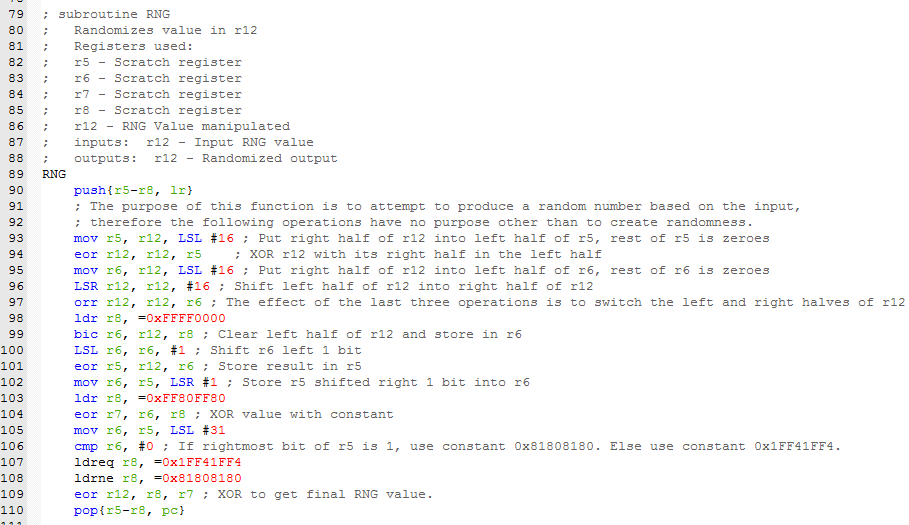
The second subroutine makes a roll. It takes an address of a skill as an input. It makes a roll by running the RNG function. It then divides the number from RNG by 20, producing a remainder in [0, 19] and adds 1 to create a number in [1, 20]. This is similar to doing rand() % 20 + 1 in C. Finally, it adds the number contained in the address of the skill to this roll to produce a final result. As it does this, it outputs the roll, the skill value, and the final result, with the proper symbols.



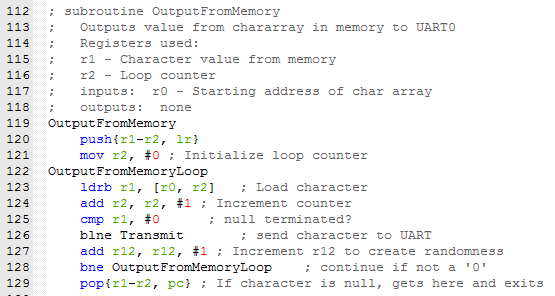
This is an example of an output from running this function.



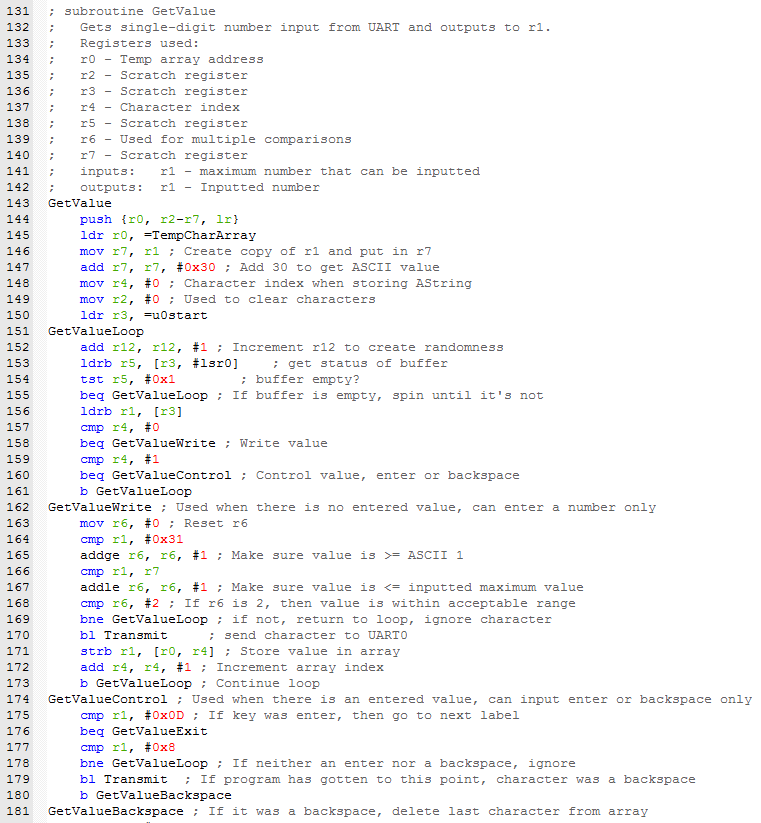
Next is the RNG function. For the whole program, r12 is used as the RNG value. On certain loops in other subroutines, 1 is added to r12 on each iteration of the loop, which creates more randomness as adding 1 to r12 before running RNG can create a completely different RNG value. Each operation of the RNG function is done to create randomness. My RNG function was inspired by this video on Super Mario 64’s RNG function, except I converted the C to ARM and extended it from 16 to 32 bit: <https://www.youtube.com/watch?v=MiuLeTE2MeQ>

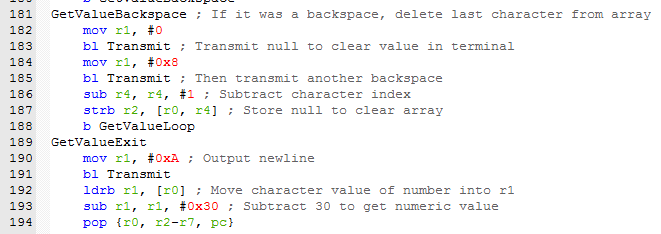


Next is a very simple subroutine. It takes as an input an address of a character array. It then transmits each character of this array until it reaches a null terminator.

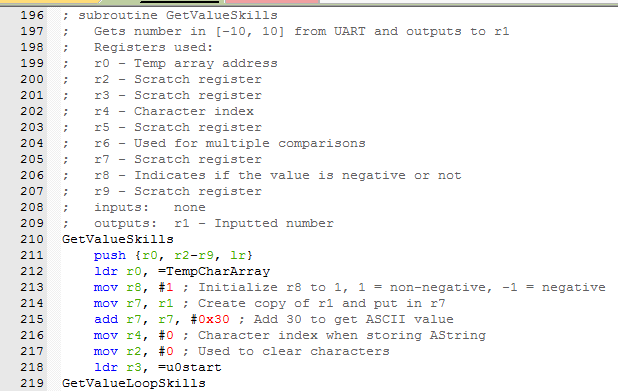


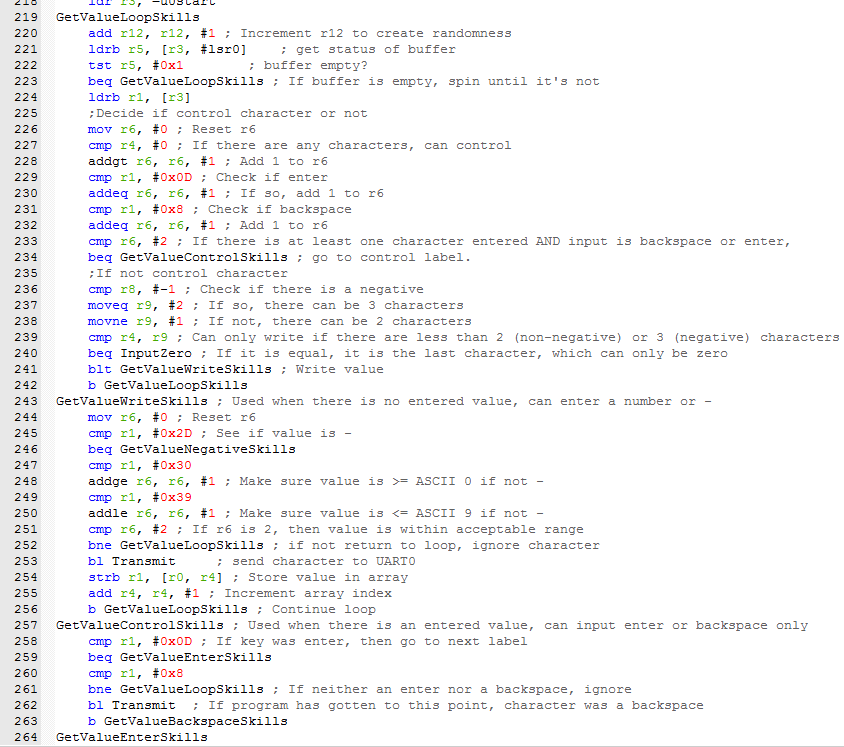
Next is a more complex subroutine. It is for entering a single-digit value, used when selecting choices. If nothing has yet been entered, it allows the user to input a number in [1, input] only. If a number has been entered, the user can then press either backspace to delete the entered character, or enter to input it. This subroutine was adapted from a similar one that allowed variable length. If I had built it from the ground up with the purpose of only allowing one character, it would probably be somewhat simpler.

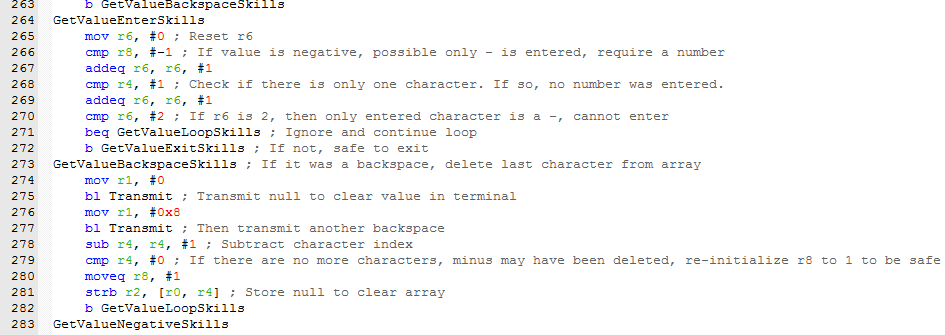


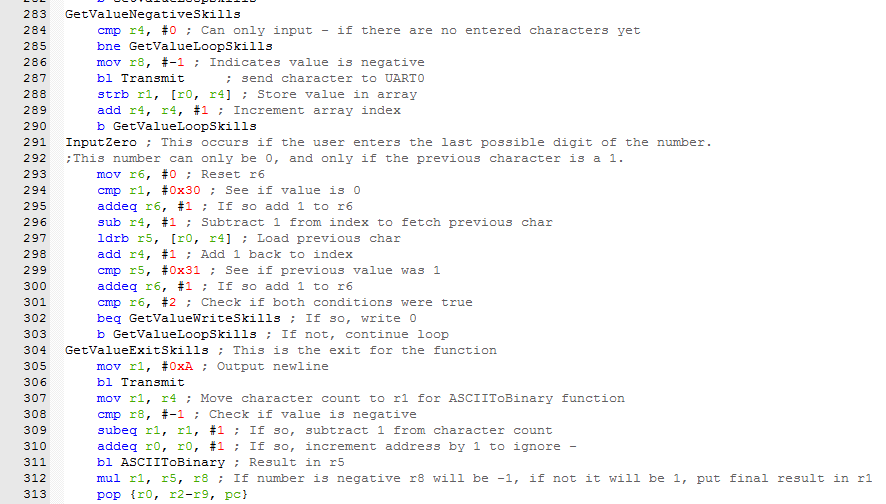


Next, is the hardest subroutine that took me the most time of anything in the project. It allows the user to enter values in [-10, 10], for selecting skill values. When nothing has yet been entered, the user can type a number in [0,9] or a -. Now a backspace can be typed. If a number was entered, the user can press enter, else they can again only type a number. If what has been typed so far is either 1 or -1, a 0 can be typed, no other number can be typed, so the user cannot enter values like 19 or -19. Furthermore, if the absolute value of the number is not 1, then no 0 can be typed, so the user cannot enter values like 90 or -90. It may have been easier to allow the user to enter whatever number they want and then validate it afterwards, but by the time I realized that I had already put too much work into this subroutine.

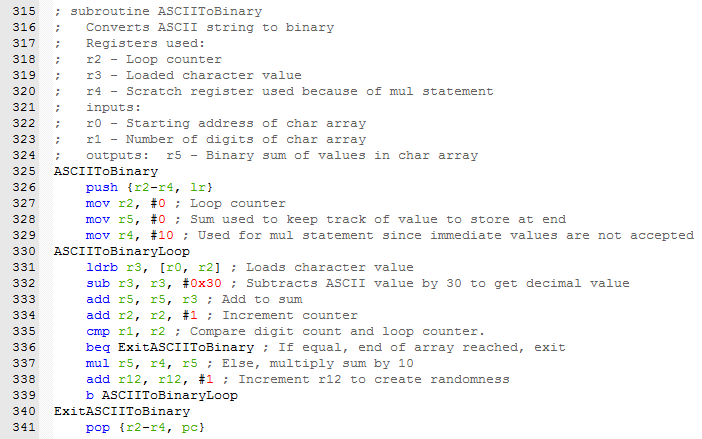




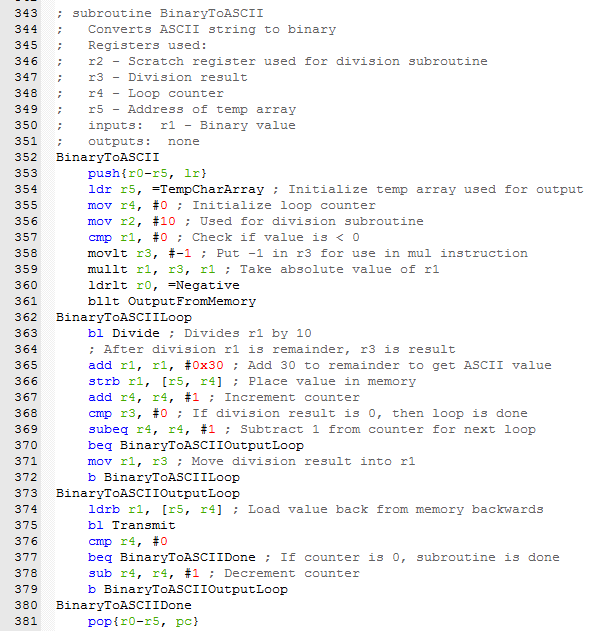




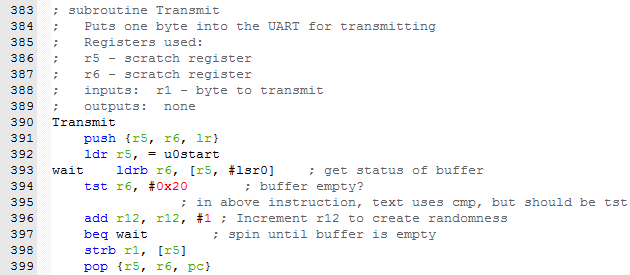
Next is a subroutine used for converting an ASCII representation of a number into binary, used in the previous GetValue subroutines. This subroutine only works with negative numbers, as GetValueSkills sends it the absolute value of the inputted number.



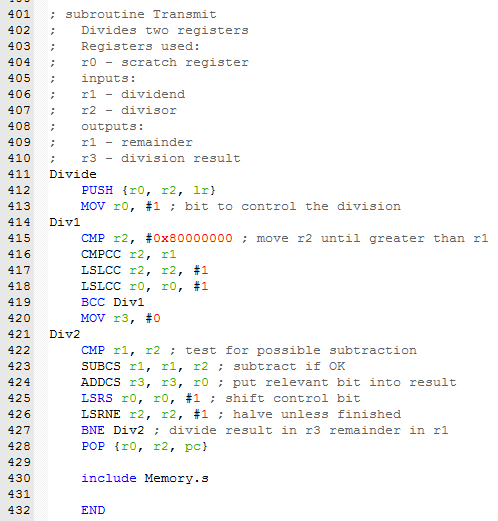
Next is the opposite, a BinaryToASCII subroutine. As the name implies, it converts a binary number to an ASCII string and outputs it. This was adapted from Lab 9, but I added the ability to accept and output negative numbers with a – as well. This subroutine is used three times in MakeRoll: To output the original roll, to output the value of the skill used, and finally to output the final result of the roll. When sending a negative skill value from MakeRoll, the absolute value will be taken (So that the output looks like x – y = z rather than x-y = z) so the only time the functionality for negative numbers is actually used is when the final roll result is negative (so the output looks like x – y = -z rather than x – y = - z).



Next is the transmit subroutine. This is the same one used in Labs 8 and 9, I saw no reason to change it when it works (except I changed it to output r1 instead of r0 for convenience). This is used in almost every other subroutine.



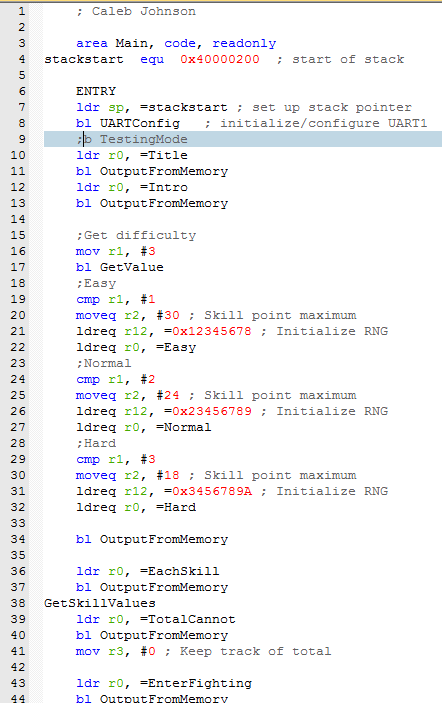
Finally, is the division subroutine from the textbook, used for getting RNG % 20 for making the roll.



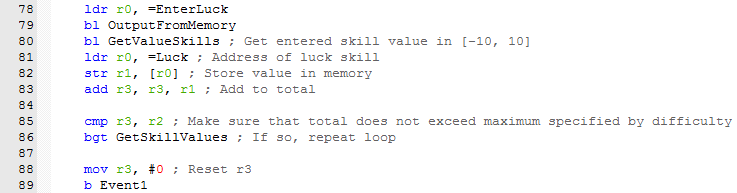
I also include Memory.s in subroutines since values from memory are used both in subroutines and in main.

We can now finally move onto the main file. This file uses all of the subroutines in order to allow the player to select their difficulty, set up their character, and then go through the game making their choices.

Main begins by configuring the UART. It then outputs the title and introductory text. Next, it allows the player to select a difficulty. RNG is then initialized based on the selected difficulty to add to randomness. Next is a loop for entering the character’s skill values.

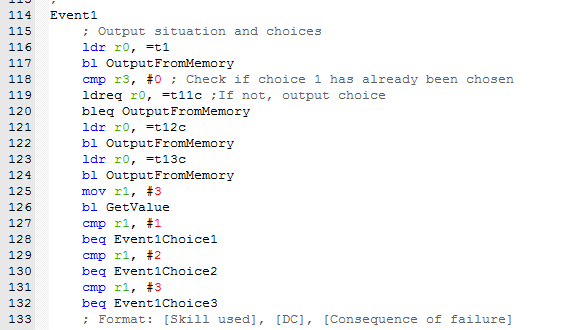


The loop informs the user that the sum of all their skill points cannot exceed the number from their difficulty level. The loop gets a value for each skill and adds it to a sum. At the end, if the sum does exceed the specified number, the loop is repeated. Else, it moves on to the first obstacle.

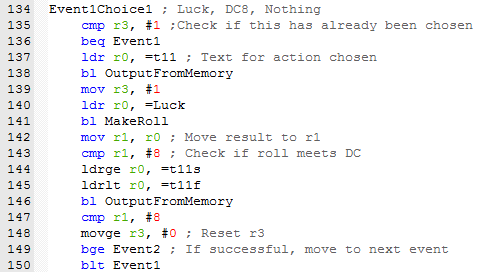


Setting up each obstacle was not hard, but it was somewhat tedious. For each one I had to have a main loop to output what the obstacle is, and what choices are available. Then for each choice, it must output the text for that choice, decide which skill to use, make a roll with that skill, check the DC, output the success or failure text for that choice, move on to the next obstacle if there is a success, and decide what to do on failure. As previously specified, there are 3 possible outcomes to failure. There can be no consequence, but that choice will no longer be available, meaning that for those I have to use a register to keep track of which choices have already been selected and make sure to not display them and not allow the choice if the player types it anyway. There is a game over, which is easy, as then the game just branches to done. Then there is fight, which means the game must branch to the fight choice, which will then decide the DC based on whether the fight was chosen or forced, which means before branching to the fight choice a register must be updated to indicate this. Then, there is the 4th choice of the final obstacle, which leads to two more possible choices, meaning that will be its own loop. But if both of these choices fail, it must branch back to the fighting choice.

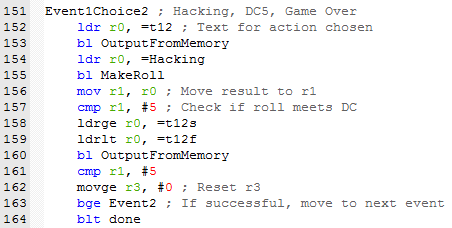
I will not show the code for every obstacle and choice as it would get repetitive, but I will show a select few. Here is the code for obstacle 1. The first choice must be checked for, but the other two don’t since their failure condition is game over.



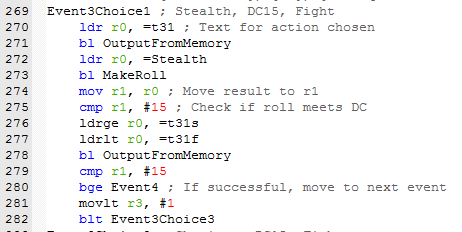
And here is the code for choice 1 of obstacle 1. Upon failure, it updates the r3 to indicate that it has been failed already.



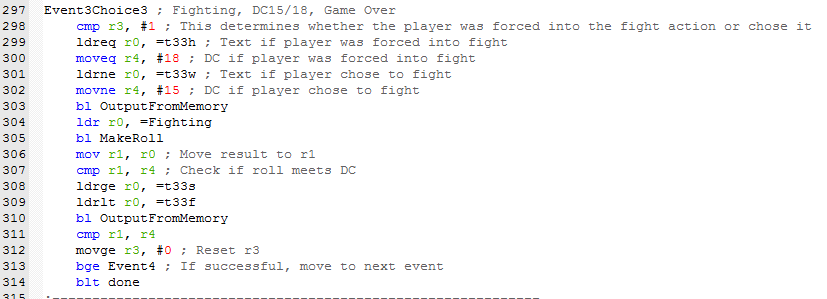
And here is the code for choice 2. Upon failure, it branches to done, as a game over was initiated.



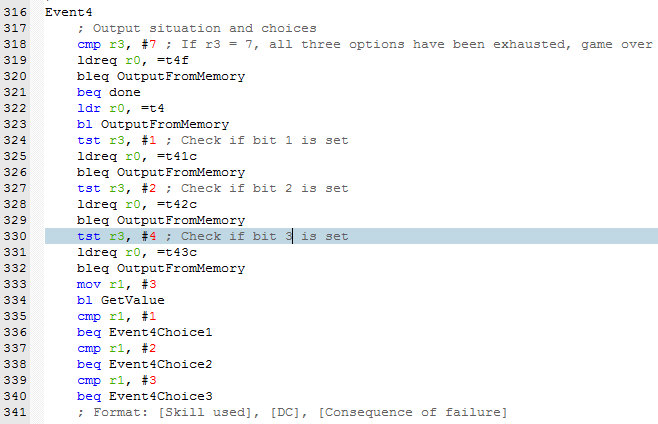
Here is choice 1 of obstacle 3. Upon failure, the player must fight the guard, so it branches to choice 3.



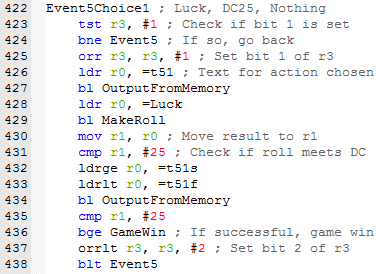
And here is choice 3, which must determine the circumstances that led to the fight to select the difficulty, and branches to done for a game over upon failure.



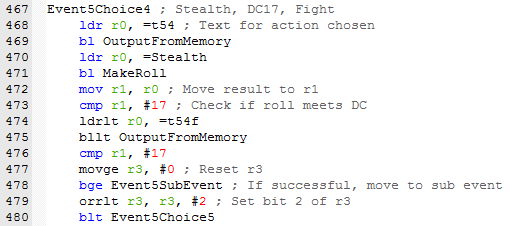
Event 4 is unique because all the choices have no consequence individually, but if all 3 fail, a game over is initiated.



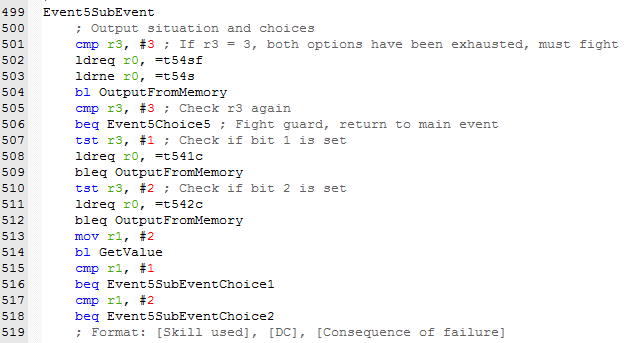
The first choice of obstacle 5 has the highest DC in the game, 25, since it is a no-consequence roll of the final event. This means that even if you have +10 to luck, you must roll a 15 or higher to be successful, which is only a 6% chance, and it is impossible to pass this unless you have at least +5 in luck.



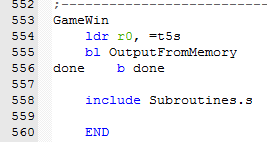
Event 5 choice 4 is unique since it leads to a sub obstacle.



The sub obstacle is similar to obstacle 4, except that when both options are exhausted a fight is initiated instead of a game over.



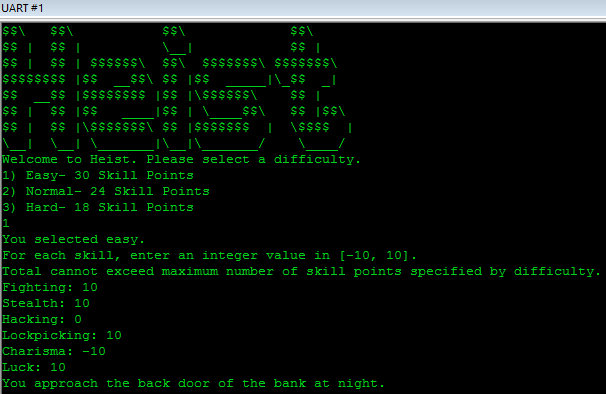
And finally, the last part of main: Outputting the win text if applicable, the finish loop, and including subroutines.

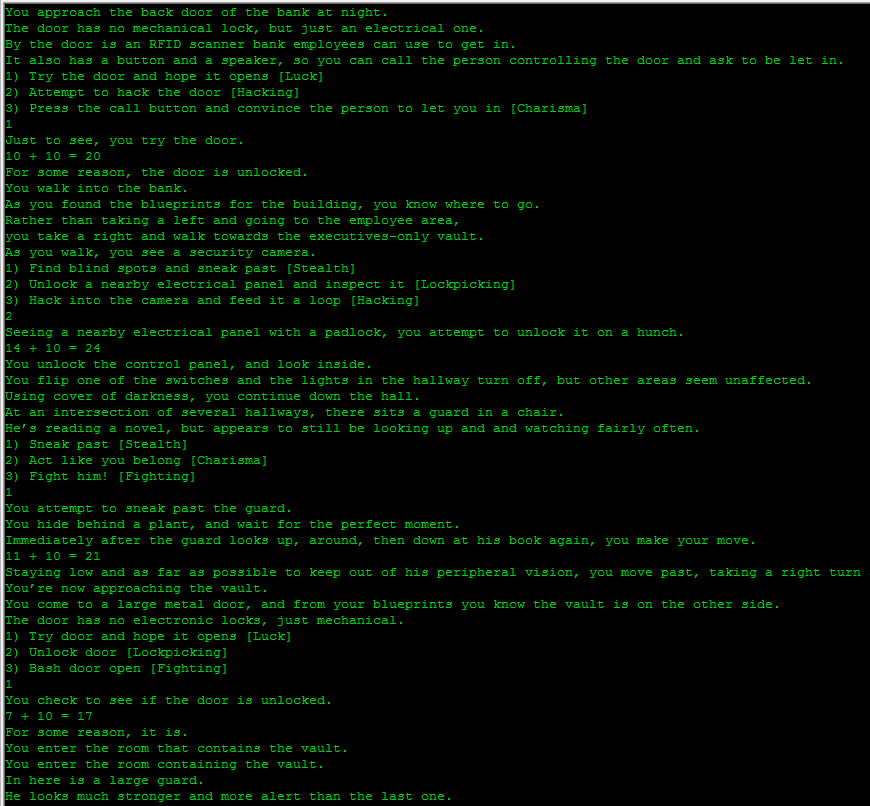


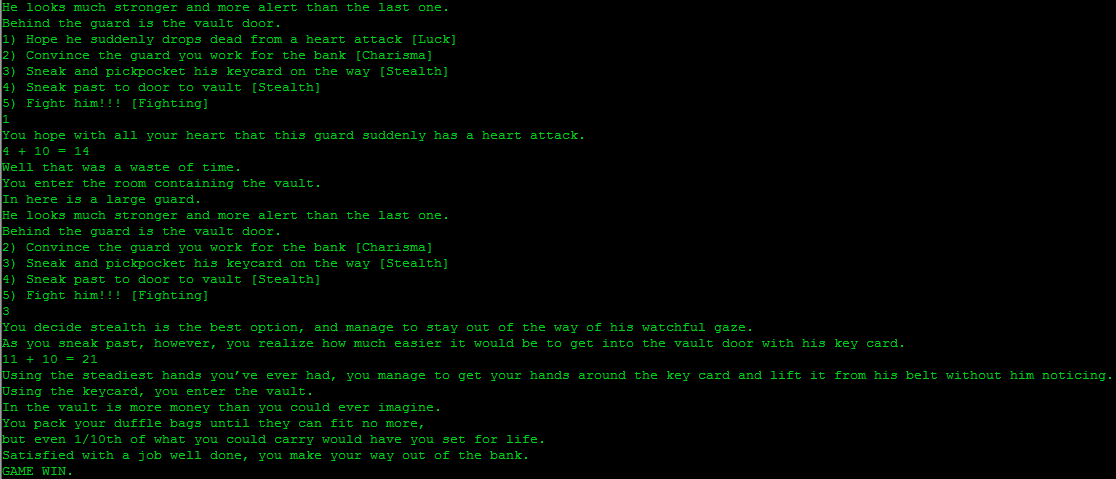
The final design of the game is simply to try to choose your skill values smartly, and then make choices that play to those skills. Some combinations of skills work better than others, so the player may have a better chance of success if they make multiple attempts and try to work out an optimal combination of high skills. As you may have noticed, choices with no consequence have higher DCs than those with consequence, which is fair. In general, a no consequence choice’s DC is 3 higher than a choice with a consequence. There is also strategy to the choices that could lead to fighting: Should I fight him now and get a lower DC, or should I try to do something else and basically get two chances at success? Which choice is better depends on how your skills are set up, again encouraging multiple attempts to figure out an optimal skill setup.

**Testing**

The implementation performs fairly well. The only time it seems slow is when outputting the title, because it contains many characters. Here is what a full playthrough of the game looks like.







The implementation actually looks a bit better than I originally expected, since I discovered that you can edit the RGB values of the UART interface to make it more thematic. Furthermore, while I did originally plan on using an ASCII title, I was happy that I managed to find one that used dollar signs, also very thematic.

**Conclusion/Future Work**

Overall I feel the project came out very well. It does everything I expected it to do and I think it’s at least mildly entertaining. One way I may improve it from a technical perspective, is to clean up the GetValue and GetValueSkill functions, although with Assembly they might not be able to be cleaned up very well. From a gameplay perspective, I considered disallowing negative values. There is a technique in tabletop RPGs called MinMaxing, where the player makes some skills very high and some very low, which is very easy to do in this game since negative values are allowed, and is the easiest way to win the game. Disallowing negative values would make the player have to think more strategically about how to distribute their skill points, but for the purpose of this project I’m okay with allowing it. Although I think the randomness is pretty good in its current state, it could also be improved by incrementing r12 in various more places, and maybe even adding the result of choices to r12.