Alan Sanchez
130942939
4/19/25

Final Report

The work I'm submitting is my original work. My program is a video game that allows users to manage hunger and money. Users must play through four minigames to make 200 dollars before the end of the month (the fifth minigame). My project reads integer input from users, prints out strings, does basic math calculations, uses random number generators, uses the stack to store values, plays sound, sleeps the program, uses memory-mapped input, and uses the bitmap display.

High-Level Description

For the high-level description of the program, see the attached C source code.

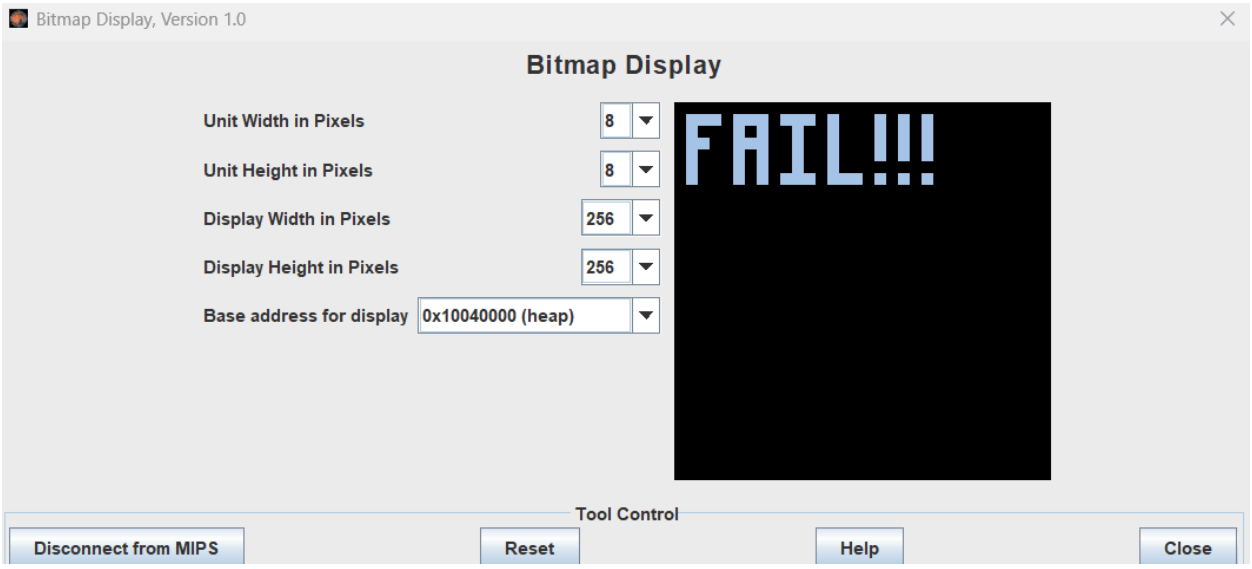
Tests

Summary of Tests

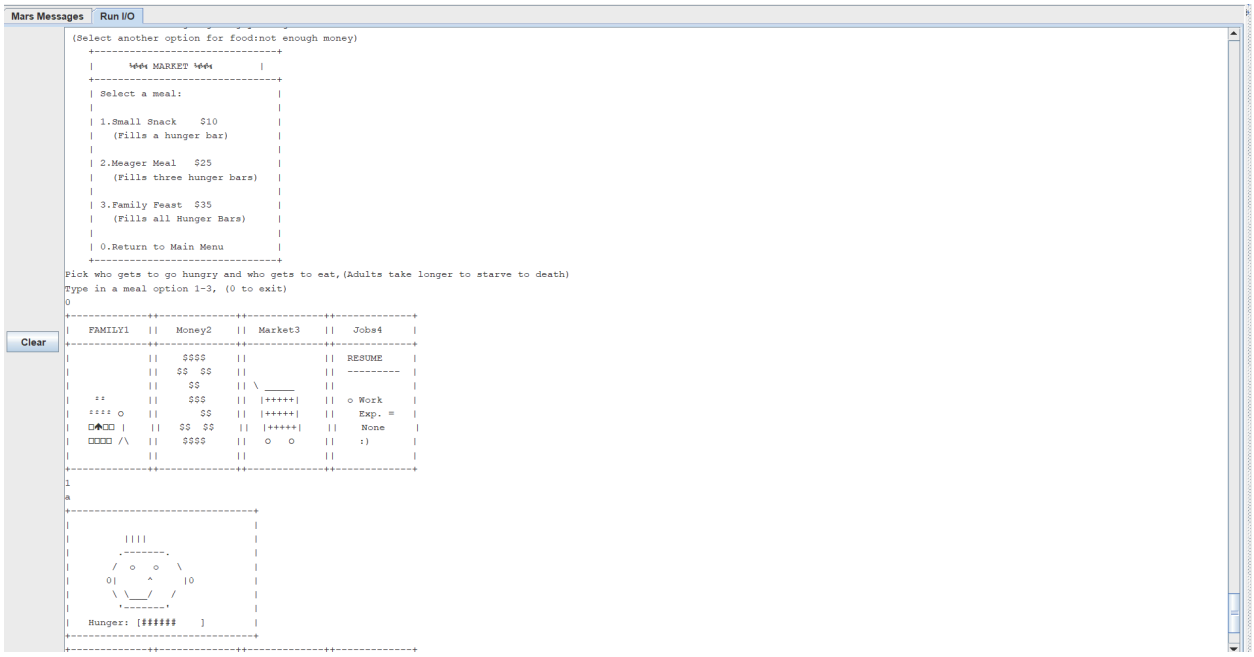
I ran the whole program from start to finish a multitude of times. I found that some functions were not properly implementing stack management. The bitmap tool also needs to be loaded before the program is loaded, or else MARS will freeze and not work properly. After I ran some tests with these issues fixed, the program worked properly. It is rare for the hunger bars to fully deplete, so a game over screen rarely, if ever, occurs. Audio seems to stutter and not be consistent. This is a known issue that occurs with MARS. The program works properly and can iterate a gameplay loop. Attached you will find a video of a gameplay loop as well as screenshots of the program working.

Attachments

YouTube Link: Pauperism Playthrough




```
Enter your bobby pin location now:
5
You are sweatign bullets
Input your next guess
7
Your heart is beating fast
Input your next guess
4
(HINT: when you are really cold the melody does not sound right)
Input your next guess
8
You got caught, you were imprisoned and fined for 50$
+-------------++-------------++-------------++-------------+
|   FAMILY1   ||   Money2    ||  Market3    ||   Jobs4     |
+-------------++-------------++-------------++-------------+
|             ||    $$$$     ||             ||  RESUME     |
|             ||   $$  $$    ||             ||  ---------  |
|             ||     $$      || \ _____     ||             |
|    ᶻᶻ       ||     $$$     ||  |+++++|    ||  o Work     |
|   ᶻᶻᶻᶻ o    ||       $$    ||  |+++++|    ||    Exp. =   |
|   □/▲□□ |   ||   $$  $$    ||  |+++++|    ||     None    |
|   □□□□ /\   ||    $$$$     ||   o   o     ||     :)      |
|             ||             ||             ||             |
+-------------++-------------++-------------++-------------+
4
Ayy, homie, step up! We shootin dice.
Aight, listen up-7 or 11, you win. 2, 3, or 12? You out.
Any other number? Thats your point, hit it again before a 7
Aint no freebies, though-if you aint got no bread, you aint playin.
So put some money down or keep it movin.
But if you ain't got no money, why you even standin here? Broke boy need to step back.
(You walk home defeated...)
+-------------++-------------++-------------++-------------+
|   FAMILY1   ||   Money2    ||  Market3    ||   Jobs4     |
+-------------++-------------++-------------++-------------+
|             ||    $$$$     ||             ||  RESUME     |
|             ||   $$  $$    ||             ||  ---------  |
|             ||     $$      || \ _____     ||             |
|    ᶻᶻ       ||     $$$     ||  |+++++|    ||  o Work     |
|   ᶻᶻᶻᶻ o    ||       $$    ||  |+++++|    ||    Exp. =   |
|   □/▲□□ |   ||   $$  $$    ||  |+++++|    ||     None    |
|   □□□□ /\   ||    $$$$     ||   o   o     ||     :)      |
|             ||             ||             ||             |
+-------------++-------------++-------------++-------------+
```



Tasks

The following is a list of tasks that were completed within the timeline:

- Basic gameplay loop
- MMIO features
- Music Features
- Four minigames
- Menu functionality

The following features were not completed within the timeline:

- At least seven minigames
- A Dance Dance Revolution minigame
- More piano functionality
- A textbox that printed out dialogue
- Main menu ASCII graphics
- Longer gameplay loops
- Implemented hamburger minigame

Conclusion

If I could do it all over again, firstly, I would use a lot of macros. Secondly, I would not be as ambitious and settle for a more ambitious project, as this project was a time sink. Thirdly, I would implement more minigames that kept in mind the tools that I had. However, video game design was really fun and had me learn new features to keep my ideas fresh.

Estimated Total Time
~100 Hours.

C Source Code

Main File

```
#include <stdio.h>
#include <string.h>
#include "headers.h"


int main(void) {
	printStartup();
	difficultySelector();
	return 0;
}
```

.C File

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "headers.h"

int i, j;
int money = 0;
int hunger1=11; //11 for n/a 0 for dead
int hunger2=11;
int hunger3=11;
int day=0;
char names[3][20];
int familySize=0;

void printStartup() {
	char row[60] = "                                _             ";
	char row2[60] = "                               (_)            ";
	char row3[60] = " _ __   __ _ _   _ _ __   ___ _ __ _ __ _ ___ _ __ ___  ";
```

```
	char row4[60] = "| ' \\  / _` | | | | '_ \\ / _ \\ '__| / __| '_ ` _ \\ ";
	char row5[60] = "| |_) | (_| | |_| | |_) |  __/ |  | \\__ \\ | | | | |";
	char row6[60] = "| .__/ \\__,_|\\__,_| .__/ \\____|_|  |_|___/_| |_| |_|";
	char row7[60] = "| |                | |                              ";
	char row8[60] = "|_|                |_|                               ";

	printf("%s\n", row);
	printf("%s\n", row2);
	printf("%s\n", row3);
	printf("%s\n", row4);
	printf("%s\n", row5);
	printf("%s\n", row6);
	printf("%s\n", row7);
	printf("%s\n", row8);

	printf("You are a illegal immigrant from mexico. Due to the nature of your citizenship you can not find a stable income.\n");
	printf("You must make ends meet in 7 days. You have...\n");
	printf("(Select game difficulty by picking how many dependents you have. Enter numbers 1-3 to pick an option)\n");
	printf("1(Easy).A son to take care of.\n");
	printf("2(Medium).A son and a wife to take care of.\n");
	printf("3(Insane).A wife and 2 kids to take care of.\n");
}
void tutorial() {
	FILE *f;
	char cont[500];

	f = fopen("tutorial.txt", "r");

	if (f == NULL) {
		printf("Error\n");
		return;
	}

	while (fgets(cont, sizeof(cont), f)) {
		printf("%s", cont);
	}

      loadMainMenu();
```

```
	fclose(f);
}
void difficultySelector() {
	int gameDifficulty = 0;

	while (1) {
		scanf("%d", &gameDifficulty);

		if (gameDifficulty == 1) {
			printf("(You selected easy mode, wuss)\n");
			loadFamily(1);
			tutorialSelector();

		} else if (gameDifficulty == 2) {
			printf("(You selected medium mode, coward)\n");
			loadFamily(2);
			tutorialSelector();

		} else if (gameDifficulty == 3) {
			printf("(You selected insane mode, Mr.Macho)\n");
			loadFamily(3);
			tutorialSelector();

		} else {
			printf("(Seriously??? I said numbers 1-3!!!)\n");
		}
	}


}
void tutorialSelector() {
	char yayOrNay[3];
	printf("(Do you want to load the tutorial? Type yes or no with a space after.)\n");
	scanf("%s", yayOrNay);

	if(strcmp(yayOrNay,"yes")==0) {
		printf("You said yes to the tutorial.\n");
		tutorial();
	}
	else if (strcmp(yayOrNay, "no")==0) {
```

```
                printf("You said no to the tutorial.\n");
                    loadMainMenu();
            }
}
void loadFamily(int x) {
    //to do create feature where the family menu is updated for each difiiculty
    //can overwrite a string
    if(x==1){
        printf("What is your sons name? (Enter a name up to 20 char):");
        scanf("%s", names[0]);
        familySize=1;
        hunger1=10;

    }
    else if(x==2){
        printf("What is your sons name? (Enter a name up to 20 char):");
        scanf("%s", names[0]);
        printf("What is your wifes name? (Enter a name up to 20 char):");
        scanf("%s", names[1]);
        hunger1=10;
        hunger2=10;
        familySize=2;
    }
    else if(x==3){
       printf("What is your sons name? (Enter a name up to 20 char):");
        scanf("%s", names[0]);
        printf("What is your wifes name? (Enter a name up to 20 char):");
        scanf("%s", names[1]);
         printf("What is your daughters name? (Enter a name up to 20 char):");
        scanf("%s", names[2]);
        hunger1=10;
        hunger2=10;
        hunger3=10;
        familySize=3;
    }

}
int randomChange(int product) {
            int x = 0;
```

```
	while (x < product) {
		x = (rand() % 50) + product;
	}

	return x;
}
void cashierGame() {
	int score = 0;
	int customerAmount = 0; // between 1 and 50 but greater than product amount
	float productAmount = 0.00; // between 1 and 100 make it discreet
	float cashierChange = 0.00; // can be exact number
	float correctAmount = 0.00;

	srand(time(NULL));

	for (i = 0; i < 5; i++) {
		productAmount = (rand() % 99) + 1;
		customerAmount = randomChange((int)productAmount);
		correctAmount =customerAmount-productAmount ;
		printf("The product costs:%.2f\n",productAmount);
		printf("The customer gave you:%d\n",customerAmount);
		printf("Input the customers change in this format: xx.xx(i.e 2.54, 0.50, 99.20)\n");
		scanf("%f", &cashierChange);
		if(fabs(cashierChange - correctAmount) < 0.01) {
			score++;
			printf("Correct change given %d more customer(s) to go.\n", 4-i);
		}
		else {
			score--;
			printf("Wrong change given %d more customer(s) to go.\n", 4-i);
		}

	}


	if (score == 1) {
		money += 5;
		printf("You did a bad job. You got 5$\n");
		printf("Balance:%d", money);
	} else if (score == 2) {
```

```
                money += 10;
                printf("You did a mediocre job. You got 10$\n");
                printf("Balance:%d", money);
        } else if (score == 3) {
                money += 15;
                printf("You did an ok job. You got 15$\n");
                printf("Balance:%d", money);
        } else if (score == 4) {
                money += 20;
                printf("You did a good job. You got 20$\n");
                printf("Balance:%d", money);
        } else if (score == 5) {
                money += 30;
                printf("You did a PERFECT job. You got 30$\n");
                printf("Balance:%d", money);
        } else {
                printf("You did a horrible job. You did not get paid.\n");
        }

    day+=1;
    loadMainMenu();
}
    int hamburger[8];



    void converter(){
       for(i=0;i<8;i++){
          if (i == 0 || i == 7) {
             printf("Bun\n");
          } else if (i == 1) {
             printf("Mayonnaise\n");
          } else if (i == 2) {
             printf("Tomatos\n");
          } else if (i == 3) {
             printf("Lettuce\n");
          } else if (i == 4) {
             printf("Meat\n");
          } else if (i == 5) {
             printf("Onions\n");
```

```
            } else if (i == 6) {
                printf("Ketchup\n");
            } else if (i == 7) {
                printf("Mustard\n");
            }
        }
    }


void fastFoodMinigame(){
    //so you got 5 customers and you must quickly identify
    //whats missing in the order and put it down
    //so we need rng machine
    //1 is true 0 is false
    int guess;
    int wrong;
    int right=0;


    srand(time(NULL));


    printf("You are Mr.Hamburger checker\n");
    printf("You will be told what ingredients go in each hamburger.\n");
    printf("You must correct the order if its wrong, you will be paid based on how fast you
are.\n");
    printf("A hamburger consists of:A bun, mayonaise, tomatos, lettuce, meat, onions, some
ketchuph, some mustard and a bun respectively. \n");
    printf("Match the order from top to bottom\n");
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 8; j++) {
            hamburger[j] = 1;
        }

        wrong = rand() % 8;
        hamburger[wrong] = 0;

        printf("\nCustomer %d's order:\n", i + 1);
        for (int j = 0; j < 8; j++) {
            if (j == 0 || j == 7) {
```

```
        printf("Bun\n");
    } else if (j == 1) {
        printf("Mayonnaise\n");
    } else if (j == 2) {
        printf("Tomatoes\n");
    } else if (j == 3) {
        printf("Lettuce\n");
    } else if (j == 4) {
        printf("Meat\n");
    } else if (j == 5) {
        printf("Onions\n");
    } else if (j == 6) {
        printf("Ketchup\n");
    } else if (j == 7) {
        printf("Mustard\n");
    }
}

printf("\nWhich ingredient is missing? Enter a number:\n");
printf("1. Bun (Top)\n");
printf("2. Mayonnaise\n");
printf("3. Tomatoes\n");
printf("4. Lettuce\n");
printf("5. Meat\n");
printf("6. Onions\n");
printf("7. Ketchup\n");
printf("8. Mustard\n");
printf("9. Bun (Bottom)\n");

scanf("%d", &guess);
guess--;

if (guess == wrong) {
    right++;

    printf("You correctly fixed the order!\n");
} else {
    printf("Wrong! The missing ingredient was: ");
    if (wrong == 0 || wrong == 7) {
        printf("Bun\n");
```

```
        } else if (wrong == 1) {
            printf("Mayonnaise\n");
        } else if (wrong == 2) {
            printf("Tomatoes\n");
        } else if (wrong == 3) {
            printf("Lettuce\n");
        } else if (wrong == 4) {
            printf("Meat\n");
        } else if (wrong == 5) {
            printf("Onions\n");
        } else if (wrong == 6) {
            printf("Ketchup\n");
        } else if (wrong == 7) {
            printf("Mustard\n");
        }
    }
}

if (right == 1) {
                money += 5;
                printf("You did a bad job. You got 5$\n");
                printf("Balance:%d", money);
        } else if (right == 2) {
                money += 10;
                printf("You did a mediocre job. You got 10$\n");
                printf("Balance:%d", money);
        } else if (right == 3) {
                money += 15;
                printf("You did an ok job. You got 15$\n");
                printf("Balance:%d", money);
        } else if (right == 4) {
                money += 20;
                printf("You did a good job. You got 20$\n");
                printf("Balance:%d", money);
        } else if (right == 5) {
                money += 30;
                printf("You did a PERFECT job. You got 30$\n");
                printf("Balance:%d", money);
        } else {
                printf("You did a horrible job. You did not get paid.\n");
```

```
        }

    day+=1;
    loadMainMenu();



}

void rollDice(int wager) {
    srand(time(NULL));
    int dice;
    int point;
    int dummyRoll;

    printf("Press any number key to roll, may luck be on your side: ");
    scanf("%d",&dummyRoll);

    dice=rand()%12+2;
    printf("You rolled a: %d\n",dice);

    if(dice==7||dice==11) {
        printf("Ayy, you hit that %d! You won, player. Looks like you just came up!\n",dice);
        money+=wager;
    } else if(dice==2||dice==3||dice==12) {
        printf("Tough break, homie. You rolled a %d. Better luck next time.\n",dice);
        money-=wager;
    } else {
        point=dice;
        printf("Aight, that's your point: %d. Now hit it again before a 7, or it's a wrap.\n",point);

        while(1) {
            printf("Press any number key to roll again: ");
            scanf("%d",&dummyRoll);

            dice=rand()%12+2;
            printf("You rolled a: %d\n",dice);

            if(dice==7) {
                printf("You rolled a 7. Tough break, homie. You lose.\n");
```

```
            money-=wager;
            break;
        } else if(dice==point) {
            printf("You hit your point! You win!\n");
            money+=wager;
            break;
        } else {
            printf("Aight, you rolled a %d. Roll again and see what's up.\n",dice);
        }
    }
  }
}

void diceLooper() {
  int decision;
  int wager;

  while(1) {
    printf("What's it gon' be? You rollin' or you foldin'?\n");
    if(money<=0) {
        printf("But if you ain't got no money, why you even standin' here? Broke ass need to step
back.\n");
        printf("(You walk home defeated...)\n");
        day++;
        loadMainMenu();
        return;
    }

    printf("You have: $%d\n",money);
    printf("Enter your wager: ");
    while(scanf("%d",&wager)!=1||wager>money||wager<=0) {
        printf("Man, you ain't got it like that. Put down somethin' real or get up outta here.\n");
        printf("Input a valid wager: ");
        while(getchar()!='\n'); // Clear invalid input
    }

    printf("Bet, you put your dough down—let's see if you can back it up or if you finna lose
that lil' change.\n");
    rollDice(wager);
```

```
        printf("Aight, you did your thing. You feelin' lucky for another round or chillin' for
now?\n");
        printf("Press 1 for another round, 2 to end betting: ");
        while(scanf("%d",&decision)!=1||(decision!=1&&decision!=2)) {
            printf("You tryna play games? Enter 1 or 2.\n");
            while(getchar()!='\n'); // Clear invalid input
        }

        if(decision==2) {
            printf("You went back home.\n");
            day++;
            loadMainMenu();
            return;
        }
    }
}

void diceGambleMinigame() {
    printf("Ayy, homie, step up! We shootin’ dice.\n");
    printf("Aight, listen up—7 or 11, you win. 2, 3, or 12? You out.\n");
    printf("Any other number? That’s your point, hit it again before a 7 or you lose.\n");
    printf("Ain’t no freebies, though—if you ain’t got no bread, you ain’t playin’.\n");
    printf("So put some money down or keep it movin’.\n");
    diceLooper();
}


void loadMinigame(int y) {
    if(day==0){
            if(y==1) {
                cashierGame();

            }
            else if(y==2) {
                lockPick();
            }
    }
    else if(day==1){
        if(y==1){
            fastFoodMinigame();
```

```
        }
        else if(y==2){
            diceGambleMinigame();
        }


    }
}
void lockPick() {
        int lpLoc=0; //lock pick location
        int userGuess=0; //user guess
        int attemptNum=(rand()%2)+2;
        srand(time(NULL));
        lpLoc=(rand()%10);

        //lock pick is a hot or cold game where the user gets closer and closer to the acutal
number
        //user has limited amount of attempts before they are busted
        //user gets hot or cold hint by flavor text

        printf("You are breaking into a safe where you work\n");
        printf("You get a random number of attempts before your lockpicking set breaks\n");
        printf("Press a number between 0-9 to move the bobby pin\n");
        printf("Enter your bobby pin location now:\n");
        // FOR DEBUGGING  printf("lp loc %d",lpLoc );
        for(i=0; i<attemptNum; i++) {
                scanf("%d", &userGuess);

                if(userGuess==lpLoc) {
                        printf("You broke into the safe and found 100$ thief...\n");
                        money+=100;
                        day+=1;
    loadMainMenu();
                        return;
                }
                else {
                        int closeness = abs(lpLoc - userGuess);
                        if (closeness == 1) {
                                printf("CLICK CLICK(super close)\n");
                        } else if (closeness == 2) {
```

```
                        printf("CLICK click(close)\n");
                    } else if (closeness == 3) {
                        printf("click click(getting there)\n");
                    } else if (closeness == 4) {
                        printf("click (cold)\n");
                    } else {
                        printf("...(super cold)\n");
                    }

                }

        }
printf("You got caught, you were imprisoned and fined for 350$\n");
money -=350;
day+=1;
   loadMainMenu();
}
void newspaperMenu() {
        char choice = '0';
   if(day==0){
        while (choice != '1' && choice != '2') {
            printf("========== Newspaper =======\n");
            printf("|Looking-for-|(Steal from)|\n");
            printf("|cashier-----|(The place )|\n");
            printf("|Hiring------|(That is   )|\n");
            printf("|Immediately-|(Hiring   ))|\n");
            printf("|------------|------------|\n");
            printf("|------------|------------|\n");
            printf("|    [1]     |    [2]     |\n");
            printf("This is the newspaper select a job by pressing 1 or 2\n");

            while (getchar() != '\n');

            scanf(" %c", &choice);

            if (choice == '1' || choice == '2') {
                loadMinigame(choice - '0');
            } else {
                printf("Invalid selection. Choose 1 or 2.\n");
            }
```

```
        }
    }
    else if(day==1){
        while (choice != '1' && choice != '2') {
                printf("========= Newspaper =======\n");
                printf("|Looking-for-|(Gamble----)|\n");
                printf("|fast-food---|(Your-money)|\n");
                printf("|worker------|(at-a-shady)|\n");
                printf("|Immediately-|(dice------)|\n");
                printf("|------------|place-------|\n");
                printf("|------------|------------|\n");
                printf("|    [1]     |    [2]     |\n");
                printf("This is the newspaper select a job by pressing 1 or 2\n");

                while (getchar() != '\n');

                scanf(" %c", &choice);

                if (choice == '1' || choice == '2') {
                        loadMinigame(choice - '0');
                } else {
                        printf("Invalid selection. Choose 1 or 2.\n");
                }
    }

    }

}
//create a day cycle
//fill out function stubs
    void familyMenu(){
        int maxbar=10;

       if (familySize >= 1) {

printf(" ┌──────────────────────────────────────────────────────
────────────────────┐ \n");
        printf(" │
 ┌─────┬────┬────┬──────────────────────────────────────────────
─────┐    │ \n");
```

```
    printf("│ │ ┌─┼──┼──┴─┐ %-10s              │  │\n", names[0]);
    printf("│ ┌─┐  o  o  ┌─┐                  │  │\n");
    printf("│ └─┘ ──────── └─┘  Hunger ┌──────────────┐          │
│\n");
    printf("│ │ └──────────┘        └");

    for (int i = 0; i < maxbar; i++) {
      if (i < hunger1) {
        printf("@");
      } else {
        printf(" ");
      }
    }
    printf("                 ───┘          │  │\n");
    printf("│
└──────────────────────────────────────────────────────────
────┘   │\n");
  }

  if (familySize >= 2) {

printf("├──────┬┬┬┬┬┬┬┬┬┬──────────────────────────────────────────
──────────────────┤\n");
    printf("│
┌───│││││││││─────────────────────────────────────────────
────┐   │\n");
    printf("│ │││││──────┐│ %-10s              │  │\n", names[1]);
    printf("│ │││ O O │││                  │  │\n");
    printf("│ │││ ───── ││ Hunger ┌──────────────┐          │  │\n");
    printf("│ ││└──────────┘│    └");

    for (int i = 0; i < maxbar; i++) {
      if (i < hunger2) {
        printf("@");
      } else {
        printf(" ");
      }
    }
    printf("                 ───┘          │  │\n");
```

```
    printf("│
└││─────────────│──────────────────────────────────────────────────
─────┘     │\n");
  }

  if (familySize >= 3) {
printf("├───┴┴─────────────────│──────────────────────────────────────────
──────────────────────┤\n");
    printf("│
┌───┬┬┬┬┬┬┬┬┬┬──────────────────────────────────────────────────────────
────┐     │\n");
    printf("│ │ ┌┼┼┼┼┼┼┼┼┼┼┼┐ %-10s                  │    │\n", names[2]);
    printf("│ ┌──┐ │ o  o │ ┌──┐                    │    │\n");
    printf("│ └──┘ │ ───────┤ └──┘ Hunger ┌──────────────┐        │
│\n");
    printf("│ │▲ └┼───────────┼┘ ▲     └");

    for (int i = 0; i < maxbar; i++) {
      if (i < hunger3) {
        printf("@");
      } else {
        printf(" ");
      }
    }
    printf("                ────┘          │    │\n");
    printf("│
└───┴───────────┴──────────────────────────────────────────────────────
────┘     │\n");

printf("└──────────────────────────────────────────────────────────────
──────────────────────┘\n\n");
  }
    loadMainMenu();
  }
  void moneyMenu(){
    printf("You have:$%d dollars\n", money);
    loadMainMenu();
  }
  void marketMenu(){
```

```
      char market[] =
" ┌──────────────────────────────────────┐ \n"
" │  Welcome to the market   │\n"
" ├──────────────────────────────────────┤ \n"
" │  Type a number to select │\n"
" │  a meal.                 │\n"
" │                          │\n"
" │  1.Small Meal 5$         │\n"
" │   (Fills 1 hunger bar    │\n"
" │    of each family        │\n"
" │    member).              │\n"
" │  2.Medium Meal $10       │\n"
" │    (Fills 2 hunger bars  │\n"
" │    of each family        │\n"
" │    member).              │\n"
" │  3. FAMILY MEAL $50      │\n"
" │    (Fully replenishes    │\n"
" │    all hunger of each    │\n"
" │    family member).       │\n"
" └──────────────────────────────────────┘ \n";
int choice=0;

 printf("%s", market);
    printf("Input a number to pick a meal\n");
    scanf("%d", &choice);
    while(choice!=5){
        if (choice == 1) {
        printf("You picked the Small Meal 5$.\n");
        hunger1+=1;
        hunger2+=1;
        hunger3+=1;
    } else if (choice == 2) {
        printf("You picked the Medium Meal 10$.\n");
        hunger1+=2;
        hunger2+=2;
        hunger3+=2;
    } else if (choice == 3) {
        printf("You picked the FAMILY MEAL 50$.\n");
        hunger1=10;
        hunger2=10;
```

```
        hunger3=10;
    } else {
        printf("Dude come on weve been through this.\n");
    }


    }



loadMainMenu();
    }



    void loadMainMenu() {
    int choice=0;
    char menu[]=

" ┌──────────────┬──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────┬──────────────┐ \n"
    "│ FAMILY1 │                                   │ Money2 │\n"
    "├──────────────┤                             ├──────────────┤\n"
    "│      │                              │  $$$$  │\n"
    "│      │                              │ $$  $$ │\n"
    "│      │                              │  $$   │\n"
    "│      │                              │   $$$  │\n"
    "│  ▲▲     │                              │    $$ │\n"
    "│▲▲▲▲ O O│                                │ $$  $$ │\n"
    "│ │┌┐│ │ ││                                 │  $$$$  │\n"
    "│ └┘┘ /\/\\│                                 │      │\n"
    "├──────────────┤                             ├──────────────┤\n"
    "│   ▼▼▼  │                               │      │\n"
    "│  ▼  ▼ │                              │      │\n"
    "│ │   │ │                             │      │\n"
    "│ ┌┴────────┼┐ │                                │ ▼▼▼▼  │\n"
    "│ │Target│ │                           │ ▼   ▼ │\n"
    "│ │     ││                            │ ┌──────────┐ │\n"
    "│ └──────────┘ │                                ││  ▼▼  ││\n"
    "│      │                              ││    ││\n"
```

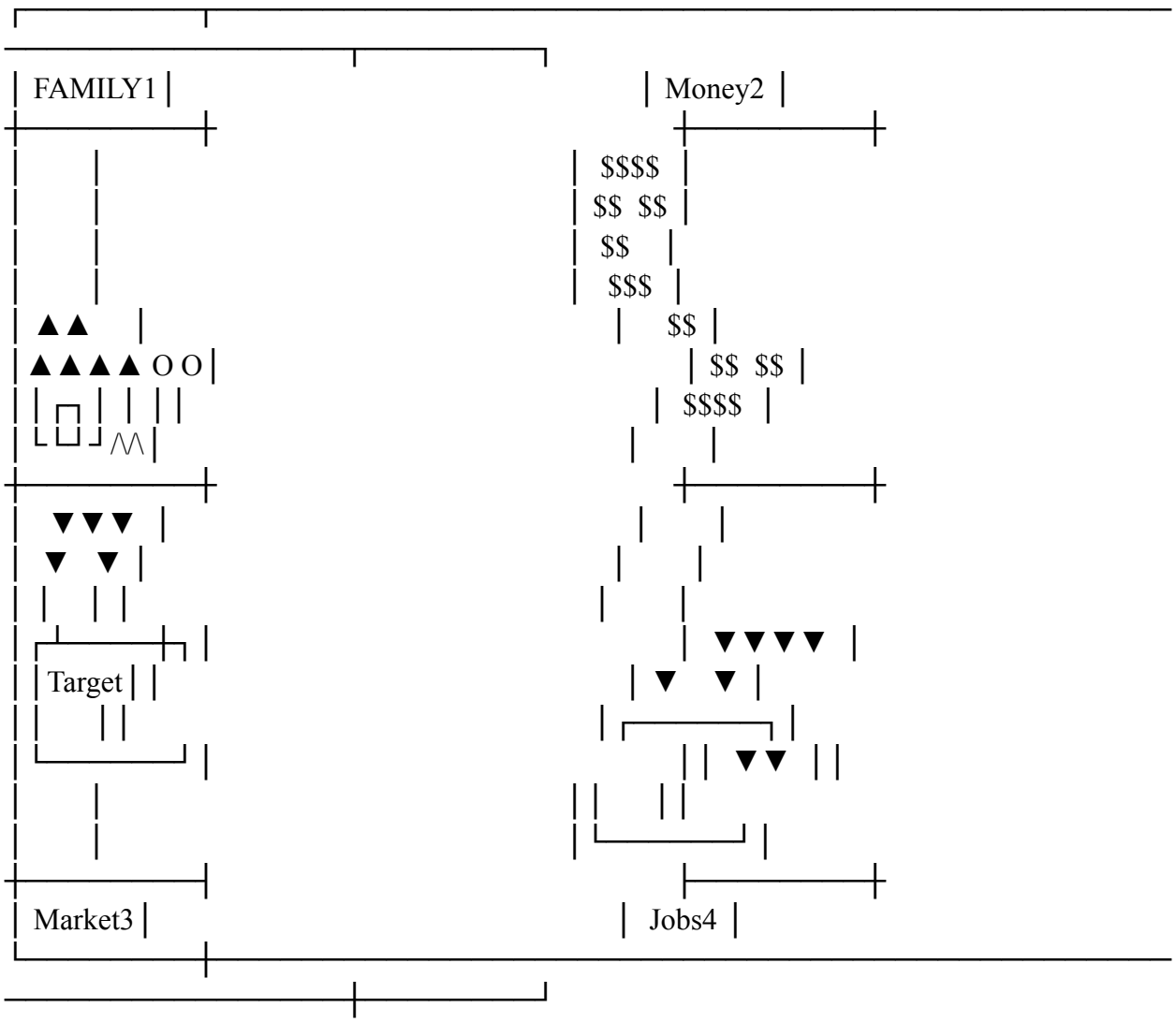
```
        "│      │                                  │ └──────────┘ │\n"
        "├────────────┤                                     ├────────────┼\n"
        "│ Market3 │                                   │ Jobs4 │\n"
    "└────────────┴────────────────────────────────────────────────────────────────────────────
────────────────────────┴────────────┘ \n";

    while (choice < 1 || choice > 4) {
        printf("%s", menu);
        scanf("%d", &choice);

        if (choice < 1 || choice > 4) {
            printf("Im not mad just dissapinted\n");
        }
    }

    if (choice == 1) {
        familyMenu();
    } else if (choice == 2) {
        moneyMenu();
    } else if (choice == 3) {
        marketMenu();
    } else if (choice == 4) {
        newspaperMenu();
    }
}
```

.h File

```
#ifndef HEADERS_H
#define HEADERS_H
void printStartup();
void tutorial();
void difficultySelector();
void tutorialSelector();
void loadFamily();
void lockPick();
void converter();
void fastFoodMinigame();
void rollDice();
void diceLooper();
```

```
void diceGambleMinigame();
void loadMinigame();
void newspaperMenu();
void moneyMenu();
void familyMenU();
void marketMenu();
void loadMainMenu();
#endif
```

.Txt File



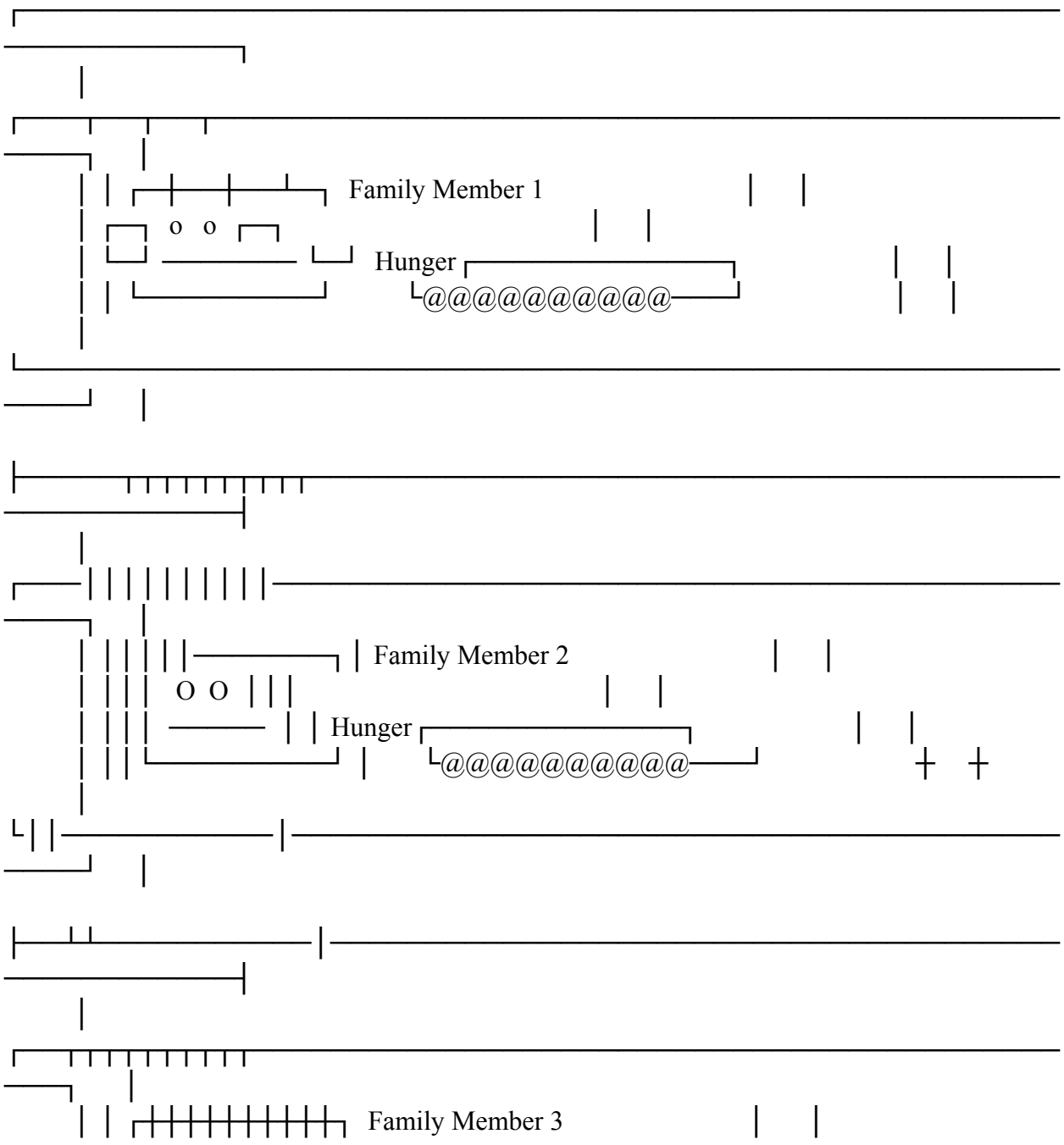
This is your menu, the box in the middle will show what you are currently doing.

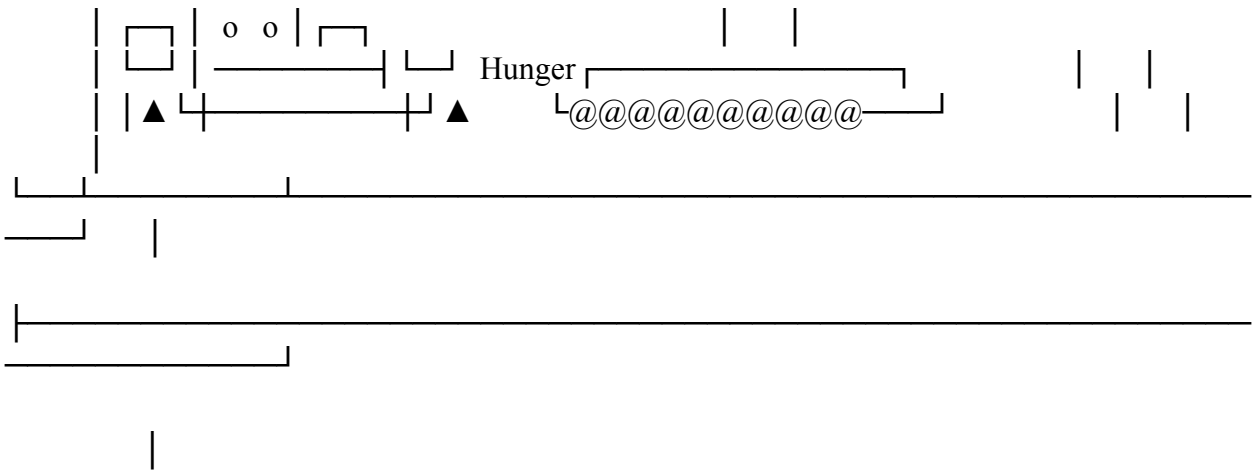
To access a menu type the number right after the menu title.

The money menu will tell you how much money you have right now.

The family screen will show each family member their hunger and current health.

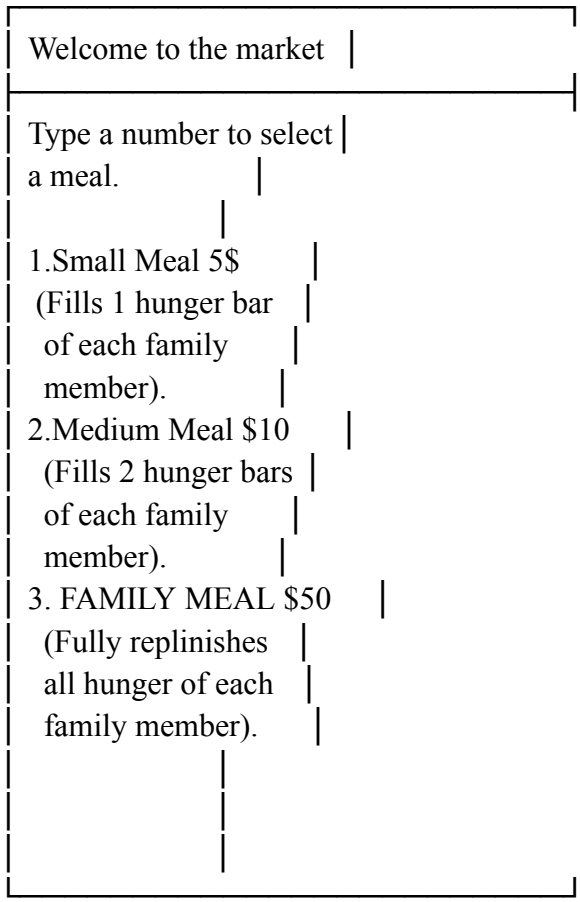
Clicking on the family tab will tell you each members status.





Make sure that their hunger does not get too low.

You can feed them by buying food through the market screen.



========= Newspaper =======

Looking-for-	(Steal from)
cashier-----	(The place)
Hiring------	(That is)
Immediately-	(Hiring)
[1]	[2]

This is the newspaper screen you will have two options to pick from.

Choose wisely as you will only be able to do one job per day.

Jobs change each day.

Try to make 500$ in order to make ends meet.

MIPS Source Code

Main File

```
.data
line1:   .asciiz  " _ \n"
line2:   .asciiz  " (_) \n"
line3:   .asciiz  " _ __ __ _ _ _ _ __ ___ _ __ _ ___ _ __ ___ \n"
line4:   .asciiz  "| ' \\ / _` | | | '_ \\ / _ \\ '__| / __| '_ ` _ \\ \n"
line5:   .asciiz  "| |_) | (_| | |_| | |_) | __/ | | \\__ \\ | | | | |\n"
line6:   .asciiz  "| .__/ \\__,_|\\__,_| .__/ \\___|_| |_|___/_| |_| |_|\n"
line7:   .asciiz  "| |               | | \n"
line8:   .asciiz  "|_|               |_| \n"
introDialogue: .asciiz  "You are a immigrant. Due to your lack of experience you can not find a stable income.\nYou must make 200$ in order to make ends meet in 4 days. You have...\n(Select game difficulty by picking how many dependents you have. Enter numbers 1-3 to pick an option)\n1(Easy).A son to take care of.\n2(Medium).A son and a wife to take care of.\n3(Insane).A wife and 2 kids to take care of.\n"
errorDiffSelect:          .asciiz  "I didnt say those numbers:( you make me sad\nTry again...\n"
diff1:   .asciiz  "(You selected easy mode, wuss)\n"
diff2:   .asciiz  "(You selected medium mode, coward)\n"
diff3:   .asciiz  "(You selected insane mode, Mr.Macho)\n"
diff4:   .asciiz  "(Seriously??? I said numbers 1-3!!!\nTry again\n"
tutPrompt:       .asciiz  "(Do you want to load the tutorial? Type 1.yes 2.no)\n"
```

```
txtFile: .asciiz  "asciiTutorial.txt"
buffer: .space  9000                        #string that holds txt
errorPrint:       .asciiz  "Could not open file\n"
errorReadPrint:          .asciiz  "Could not read file\n"
promptSon:     .asciiz  "Enter your sons name(20 Chars):\n"
promptWife:    .asciiz  "Enter your Wifes name(20 Chars):\n"
promptDaughter:       .asciiz  "Enter your daughters name(20 Chars):\n"
stateSonName:          .asciiz  "Your sons name is:\n"
stateWifeName:         .asciiz  "Your Wifes name is:\n"
stateDaughterName:   .asciiz  "Your daughters name is:\n"
sonName:       .space  20
daughterName:          .space  20
wifeName:      .space  20
familyFile:      .asciiz  "familyFile.txt"
hunger1:         .asciiz  "|   Hunger: ["  #Hunger bar is universal  it prints out the beggining and
end while the rest is printed out by a function according to current huner
hunger2:         .asciiz"]       |\n"
hunger3:         .asciiz"#"
hunger4:         .asciiz" "
sonGUI1:         .asciiz  "+--------------------------------+\n"
sonGUI2:         .asciiz  "|                                |\n"
sonGUI3:         .asciiz  "|         ||||                   |\n"
sonGUI4:         .asciiz  "|        .-------.               |\n"
sonGUI5:         .asciiz  "|       /  o   o   \\            |\n"
sonGUI6:         .asciiz  "|      0|     ^     |0          |\n"
sonGUI7:         .asciiz  "|       \\\ \\___/   /            |\n"
sonGUI8:         .asciiz  "|        '-------'               |\n"
sonGUI10:      .asciiz  "+--------------------------------+\n"
wifeGUI1:      .asciiz  "+--------------------------------+\n"
wifeGUI2:      .asciiz  "|                                |\n"
wifeGUI3:      .asciiz  "|      ||||||||||               |\n"
wifeGUI4:      .asciiz  "|     ||||||||||||              |\n"
wifeGUI5:      .asciiz  "|     |||| o   o  ||             |\n"
wifeGUI6:      .asciiz  "|    ||||     |   |||           |\n"
wifeGUI7:      .asciiz  "|     ||\\ /_____\\  /||          |\n"
wifeGUI8:      .asciiz  "|     || '-------'  ||          |\n"
wifeGUI10:     .asciiz  "+--------------------------------+\n"
daughterGUI1:          .asciiz  "+--------------------------------+\n"
daughterGUI2:          .asciiz  "|                                |\n"
daughterGUI3:          .asciiz  "|        |||||||||             |\n"
```

```
daughterGUI4:          .asciiz  "|     ||.||||||||||          |\n"
daughterGUI5:          .asciiz  "|     | / o   o  \\|          |\n"
daughterGUI6:          .asciiz  "|     ||    \\    ||         |\n"
daughterGUI7:          .asciiz  "|     |\\ \\___/  / |          |\n"
daughterGUI8:          .asciiz  "|     O '-------'   O        |\n"
daughterGUI10:         .asciiz  "+-------------------------------+\n"
mainMenuL1:  .asciiz  "+-------------++-------------++-------------++-------------+\n"
mainMenuL2:  .asciiz  "|  FAMILY1  ||  Money2  ||  Market3  ||  Jobs4    |\n"
mainMenuL3:  .asciiz  "+-------------++-------------++-------------++-------------+\n"
mainMenuL4:  .asciiz  "|           ||   $$$$    ||           ||  RESUME    |\n"
mainMenuL5:  .asciiz  "|           ||  $$  $$   ||           ||  ---------  |\n"
mainMenuL6:  .asciiz  "|           ||    $$     || \\ _____    ||           |\n"
mainMenuL7:  .asciiz  "|   ▲▲      ||    $$$    ||  |+++++|   ||  o Work    |\n"
mainMenuL8:  .asciiz  "|  ▲▲▲▲ O   ||     $$    ||  |+++++|   ||   Exp. =   |\n"
mainMenuL9:  .asciiz  "|  ▏┌┐▕ |   ||  $$  $$   ||  |+++++|   ||   None    |\n"
mainMenuL10: .asciiz  "|  ┗┘┗ /\\   ||   $$$$    ||  O   O    ||   :)      |\n"
mainMenuL11: .asciiz  "|           ||           ||           ||           |\n"
mainMenuL12: .asciiz  "+-------------++-------------++-------------++-------------+\n"
escapeString:  .asciiz  "Press 1 to return to the Main Menu\n"
moneyStatus:  .asciiz  "WOW!!! You have, "
moneyStatus1:.asciiz  "$!\nCan I have one?\n(REMEMBER YOU NEED 200$$$$$ TO MAKE
ENDS MEET!!!)\n"
marketMenuUI:          .asciiz "   +-------------------------------+\n   |     ▼▼▼ MARKET ▼▼▼
|\n   +-------------------------------+\n   | Select a meal:              |\n   |                             |\n   |
1.Small Snack   $10          |\n   |  (Fills a hunger bar)       |\n   |                          |\n   |
2.Meager Meal   $25          |\n   |  (Fills three hunger bars)  |\n   |                          |\n   |
3.Family Feast  $35          |\n   |  (Fills all Hunger Bars)    |\n   |                          |\n   |
0.Return to Main Menu        |\n   +-------------------------------+\n"
marketMenuUI1:         .asciiz  "Pick who gets to go hungry and who gets to eat,(Adults take
longer to starve to death)\nType in a meal option 1-3, (0 to exit)\n"
marketMenuUI2:         .asciiz  "Type in the lucky family member(1.Son 2.Wife 3.Daughter)\n"
marketMenuUI3:         .asciiz  "Looks like someones going hungry tonight...\n (Select another
option for food:not enough money)\n"
marketMenuUI4:         .asciiz  "Good Job! you fed your Son\n"
marketMenuUI5:         .asciiz  "Good Job! you fed your Wife\n"
marketMenuUI6:         .asciiz  "Good Job! you fed your Daughter\n"
invalidFamilySizerError:         .asciiz  "You selected a family member that does not exist, pick a
valid family member\n"
exitMessage:  .asciiz  "Sucessfully returned to mainMenu, terminating\n"
```

```
gameoverHunger:     .asciiz  "You are a horrible provider, one or more family members starved to death, GAME OVER!\n"
cashierGame1:.asciiz  "The product costs:\n"
cashierGame2:.asciiz  "The customer gave you:\n"
cashierGame3:.asciiz  "Input customers change as a single integer\n"
score0:          .asciiz  "You did a horrible job. You did not get paid.\n"
score1:          .asciiz  "You did a bad job. You got 5$\n"
score2:          .asciiz  "You did a mediocre job. You got 10$\n"
score3:          .asciiz  "You did an ok job. You got 15$\n"
score4:          .asciiz  "You did a good job. You got 20$\n"
score5:          .asciiz  "You did a PERFECT job. You got 30$\n"
balance:         .asciiz  "Balance:\n"
cashierGame0:.asciiz  "Welcome to your first job as a cashier.\nYou must calculate the correct change for customers.\n"
cashierGame4:.asciiz  "Correct!\n"
cashierGame5:.asciiz  "WRONG!Keep it up and you will be fired...\n"
lockPickEpilogue:     .asciiz  "Shortly after you clocked out of your last shift you were fired as your workplace was involved in a insurance scandal.\nWith no other recourse, you are forced to steal from your previous employer.\n(Listen to the musical melody it will give you hints to how close you are to unlocking the safe)\nYou get a random number of attempts before your lockpicking set breaks\nPress a number between 0-9 to move the bobby pin\nEnter your bobby pin location now:\n"
lockPickCorrect:      .asciiz  "You broke into the safe and found 150$ thief...\n\n"
lockPickClose1:       .asciiz  "Your palms are sweaty\n Input your next guess\n"
lockPickClose2:       .asciiz  "Your heart is beating fast\nInput your next guess\n"
lockPickClose3:       .asciiz  "Butterflies crawl up your stomach\nInput your next guess\n"
lockPickClose4:       .asciiz  "You are sweatign bullets\nInput your next guess\n"
lockPickNotClose:     .asciiz  "(HINT: when you are really cold the melody does not sound right)\nInput your next guess\n"
lockPickGameOver:   .asciiz"You got caught, you were imprisoned and fined for 50$\n"
errorLP:         .asciiz  "Invalid Input, valid numbers are 0-9. Restarting Minigame\n"
diceGamblingIntro1:   .asciiz  "Ayy, homie, step up! We shootin dice.\n"
diceGamblingIntro2:   .asciiz  "Aight, listen up-7 or 11, you win. 2, 3, or 12? You out.\n"
diceGamblingIntro3:   .asciiz  "Any other number? Thats your point, hit it again before a 7\n"
diceGamblingIntro4:   .asciiz  "Aint no freebies, though-if you aint got no bread, you aint playin.\n"
diceGamblingIntro5:   .asciiz  "So put some money down or keep it movin.\n"
diceGamblingLoop1:  .asciiz  "Whats it gonna be? You rollin or you foldin?\n"
diceGamblingLoop2:  .asciiz  "You have:"
diceGamblingLoop3:  .asciiz  "Enter your wager:\n"
```

```
dicegamblingError:     .asciiz  "Man, you ain't got it like that. Put down somethin real or get up outta here.\nInput a valid wager:\n"
diceGamblingLoop4:  .asciiz  "Bet, you put your dough down let's see if you can back it up or if you finna lose that lil change.\n"
diceGamblingNoMoney:        .asciiz  "But if you ain't got no money, why you even standin here? Broke boy need to step back.\n(You walk home defeated...)\n"
rollDialouge:   .asciiz  "Press any number to roll, Good Luck!\n"
rollDialouge1: .asciiz  "You rolled a:\n"
winDice:        .asciiz  "Ayy, you hit that! You won, player. Looks like you just came up!\n"
lossDice:       .asciiz  "Tough break, homie. Better luck next time.\n"
pointDice:      .asciiz  "Aight, thats your point, Now hit it again before a 7, or its a wrap.\n"
pointDescriber:         .asciiz  "Your point is: \n"
dicePointLoopDialouge:        .asciiz  "Press any number key to roll again:\n"
dicePointLoopDialouge1:       .asciiz  "Aight, you rolled a:\n"
dicePointLoopDialouge2:       .asciiz  "Roll again and see whats up.\n"
returnDialogue:         .asciiz  "Press 1 to return to main menu, press any other number to continue gambling\n"
introDialogue11:        .asciiz  "(FOR THIS JOB YOU NEED TO OPEN THE MMIO AND CONNECT THE TOOL)\nHey there buddy-o!\nYou look like you got talent, would you join my band!!!???\n(Turn on microphone for this minigame, speak your answer)\n"
introDialogue1:         .asciiz  "(Speak          Now:)\n"
introDialogue2:         .asciiz  "ANSWER     DETECTED:YOU SAID YES\n       Awesome buddy-o          do you know how       to play the keyboard?(Speak into the microphone)\n"
introDialogue3:         .asciiz  "ANSWER     DETECTED:YOU SAID YES          I KNOW HOW TO PLAY THE KEYBOARD\nAwesome! Well lets see here im going to let you use the keyboard now\n Its a special 4 note-keyboard, that uses numbers 1-4 as keys\nI found this badboy in a secondhand-store!\n When youve mastered it summon me by pressing 0\n"
practiceDialogue:       .asciiz  "Press keys 1-4 to play music. 0 to exit:\n"
minigameStartDialogue:        .asciiz  "Welp buck-o, Im going          to play the song once for you\n   I expect you toget it on the first try.\nWe will pay you accordingly to your performance at the concert\n"
scoreBad:       .asciiz  "Never touch an instrument again! I dont even know you\n(You did terrible, you got paid:0$\n"
scoreOk:        .asciiz  "The crowd looks relatively happy I guess...\n(You did an okay job, you got paid:100$\n"
scoreGood:     .asciiz  "WOW WHERE DID YOU LEARN HOW TO PLAY LIKE THAT!!! \n(You did a good job, you got paid:150$\n"
notePat:        .word   1,3,1,3,4
        .word   2,3,4,2,4
        .word   2,4,2,2,4
```

```
userInput:                      .space          60
finalDialogue: .asciiz  "Well you made it to the end of the week without anyone dying\nLets see if you made enough money to pay rent\nOpen the bitmap with these settings and press any number when you are ready to print results\n"
finalDialogue1:         .asciiz  "UNIT WIDTH AND HEIGHT:8\nDISPLAY WIDTH AND HEIGHT:256\n BASE ADRESS:(0X10040000)\n"
realDialogue:  .asciiz  "Press keys 1-4 to play, your order will be recorded and scored\n"
.text
.macro ioCheck
main:
                        #Function to make sure user input is safe, takes in a lower and upper bound and checks if input is valid
        li      $v0,5
        syscall

        move    $t3,$v0


                        #s6 lower bound
                        #s7 upper bound
                        #t3 input

        blt     $t3,$s6,error
        bgt     $t3,$s7,error



        move    $v0,$t3
        j       end
error:
        li      $v0, 4  #print error if invalid
        la      $a0, errorDiffSelect
        syscall
        j       main    #loops if error
end:
.end_macro

.macro colorPastelBlue
        li      $a1, 0xA7C7E7           #Loads the color blue for bitmap functions
.end_macro
```

```
.macro pixelWidth
	li	$a3, 32#Since sleep thread is not accurate we must compensate #pixel=(256
pixels ÷ 8 pixels per unit = 32#Since sleep thread is not accurate we must compensate units)
calculates the pixel width
.end_macro

.macro baseAdress
	li	$a2, 0x10040000		#Loads base adress for bitmap function
.end_macro
main:
	li	$s0, 0 #money variable
	li	$s1, 11 #hunger for family member 1
	li	$s2, 11 #hunger for family member 2
	li	$s3, 11 #hunger for family member 3
	li	$s4, 0 #day
	li	$s5, 0 #family size
			# s6 and s7 are used as extra register for macro calls


	jal	printStartup	#prints out the start menu
	jal	difficultSelect	#loads the difficulty select
	jal	tutSelect	#loads tutorial select
	jal	mainMenu	#loads the main menu which is the gameplay loop

	li	$v0,10 #ends program in case of errors
	syscall

printStartup:
	la	$a0, line1	#we initialize the syscall to print out strings then print out an array.
	li	$v0, 4
	syscall
	la	$a0, line2
	syscall
	la	$a0, line3
	syscall
	la	$a0, line4
	syscall
	la	$a0, line5
	syscall
```

```
        la      $a0, line6
        syscall
        la      $a0, line7
        syscall
        la      $a0, line8
        syscall
        la      $a0, introDialogue
        syscall
        jr      $ra     #ends and returns

difficultSelect:
        addi    $sp, $sp, -4   # Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)

while:  #a while loop just in case the user decides to pick another number besides the ones
provided
        #multi branch if statement that will call the function that loads difficulty
        #scanf and place result in t0
        #t0 contains the difficulty variable


        li      $s6,1   #arguements to make sure user input is valid
        li      $s7,4
        ioCheck         #check that user input is valid
        move    $t0, $v0        #the return value is stored in v0 so to use we put it in t0

        beq     $t0, 1, ifdiff1   #branch if diffuclty==1, initalizes game with that difficulty
        beq     $t0, 2, ifdiff2 #branch if diff==2
        beq     $t0, 3, ifdiff3 #branch if diff==3

        j       ifdiff4 #if the program somehow reaches down here it loops
ifdiff1: # depending on branch taken load family with a hunger variable of 10 so that they are
active now
        li      $s1, 10
        li      $s5, 1 #set family size
        la      $a0, diff1      #prints out dialogue saying what difficulty is selcted
        li      $v0, 4
        syscall
        j       promptFamily #asks user for family name
```

```
ifdiff2:
        li      $s1, 10
        li      $s2, 10#sets the son and wife as active NPCS
        la      $a0, diff2
        li      $v0, 4  #prints out selected difficulty string
        li      $s5, 2
        syscall
        j       promptFamily #prompts uer for family name

ifdiff3:
        li      $s1, 10
        li      $s2, 10
        li      $s3, 10#sets all family members as active
        la      $a0, diff3
        li      $v0, 4  #lets user know what difficulty is selcted
        li      $s5, 3
        syscall
        j       promptFamily #prompts usr for family names

ifdiff4:
        la      $a0, diff4      #error message, put here just in case
        li      $v0, 4
        syscall
        j       while
promptFamily:
        la      $a0, promptSon
        li      $v0, 4 #prompt to enter string
        syscall

        li      $v0, 8
        la      $a0, sonName #sonName has the name of son
        li      $a1, 20
        syscall

                #Print son name
        la      $a0, stateSonName
        li      $v0, 4
        syscall

        la      $a0, sonName
```

```
        li      $v0, 4
        syscall
        beq     $s5, 1, endDiffSelect #if just son end program else prompt wife and daughter

        la      $a0, promptWife
        li      $v0, 4 #prompt to enter wife name
        syscall

        li      $v0, 8
        la      $a0, wifeName #wifename is where  wifes name is sotred
        li      $a1, 20
        syscall

        la      $a0, stateWifeName #state wife name
        li      $v0, 4
        syscall

        la      $a0, wifeName
        li      $v0, 4
        syscall
        beq     $s5, 2, endDiffSelect  #go back to main if family=2 else continue prompting

        la      $a0, promptDaughter
        li      $v0, 4 #prompt to enter daughters name
        syscall

        li      $v0, 8
        la      $a0, daughterName #stores daughter name in array
        li      $a1, 20
        syscall

        la      $a0, stateDaughterName      #prints daughter name
        li      $v0, 4
        syscall

        la      $a0, daughterName
        li      $v0, 4
        syscall
        j       endDiffSelect #can let the program fall through to endDiff but for safe
programming we add label
```

```
endDiffSelect:
        lw      $ra, 0($sp)     #pops return register from stack and returns
        addi    $sp, $sp, 4
        jr      $ra

tutSelect:
        addi    $sp, $sp, -4    #Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)

        li      $v0, 4
        la      $a0, tutPrompt
        syscall #print out tut prompt

        li      $s6,1
        li      $s7,2
        ioCheck         #Safeguard for user input

        move    $t0, $v0        #store user input to be compared

        beq     $t0, 1, tutYes
        lw      $ra, 0($sp)     #pops return register from stack and returns
        addi    $sp, $sp, 4
        jr      $ra

tutYes:
        li      $v0, 13         #gets syscall for file reading
        la      $a0, txtFile
        li      $a1, 0 #0 for no flags
        li      $a2, 0 #0 to read
        syscall


        bltz    $v0, errorOpen          #If file can not be opened then we print out error
        move    $t1, $v0

        li      $v0, 14         #syscall to read file
        move    $a0, $t1
        la      $a1, buffer
        li      $a2, 9000
```

```
        syscall

        bltz    $v0, errorRead#if file can not be read then we print out error

        li      $v0, 4  #buffer to print out file contents
        la      $a0, buffer
        syscall

        li      $v0, 16         #closes file
        move    $a0, $t1
        syscall

        lw      $ra, 0($sp)     #pops return register from stack and returns
        addi    $sp, $sp, 4
        jr      $ra

errorOpen:
        li      $v0, 4  #prints error and rreturns
        la      $a0, errorPrint
        syscall

        lw      $ra, 0($sp)     #pops return register from stack and returns
        addi    $sp, $sp, 4
        jr      $ra

errorRead:
        li      $v0, 4
        la      $a0, errorReadPrint #if not enough memory is allocated then this error will print
and text will be trunancated
        syscall

        lw      $ra, 0($sp)    #pops the ra from stack loads it to t0 and we return to t0 which is
        addi    $sp,$sp,4
        jr      $ra
printMainMenu:
                        # So we load the sys register w 4 to print out lines then we load each
individual line and print it out
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)    # Save return address
```

```
        la      $a0, mainMenuL1     #prints out the main menu array by using syscall 4
        li      $v0, 4
        syscall
        la      $a0, mainMenuL2
        syscall
        la      $a0, mainMenuL3
        syscall
        la      $a0, mainMenuL4
        syscall
        la      $a0, mainMenuL5
        syscall
        la      $a0, mainMenuL6
        syscall
        la      $a0, mainMenuL7
        syscall
        la      $a0, mainMenuL8     #still printing out the array
        syscall
        la      $a0, mainMenuL9
        syscall
        la      $a0, mainMenuL10
        syscall
        la      $a0, mainMenuL11
        syscall
        la      $a0, mainMenuL12
        syscall #array print is over



        lw      $ra, 0($sp)         #returns call
        addi    $sp, $sp, 4
        jr      $ra

mainMenu:
        jal     printMainMenu       #print mainmenu so user can select
        li      $s6,1
        li      $s7,4
        ioCheck
        move    $t0, $v0
```

```
	beq	$t0, 1, callfamMenu	#if user input=1 then jump to the call familly menu since doing so directly will omit print instructions
	beq	$t0, 2, callmoneyMenu		#if user input=2 then jump to the money menu
	beq	$t0, 3, callmarketMenu		#if user input=3 then jump to the market menu
	beq	$t0, 4, calljobMenu	#if user input=4 then jump to the jobs menu

	j	mainMenu
callmoneyMenu:
			#Implented these functions to avoid stack overflow all of them simply link the function then return to mainmenu
	jal	moneyMenu
	j	mainMenu
callmarketMenu:
	jal	marketMenu
	j	mainMenu
calljobMenu:
	jal	jobMenu
	j	mainMenu
minigame2:
			#This is the beggining of the lockpicking mingame
			#Lockpicking minigame essentialy has players play hot or cold. Users must guess the right number from 0-9, the closer there guess the nicer the jingle will play
	li	$v0,4
	la	$a0,lockPickEpilogue #prints out dialogue
	syscall

	li	$t4,0	#t4 is our i=0

	jal	RNGlockPickLoc	#t0 contains lockPick location 1-10
	jal	RNGguess	#t2 contains number of guess's 2-4
lockPickLoop:
			#We now do a for loop for gameplay each time user input is checked and validated
	li	$v0,5
	syscall
	move	$t1,$v0		#$t1 contains user guess

	blt	$t1,0,inputErrorLP
	bgt	$t1,9,inputErrorLP
```

```
	jal	inputValidatorLP	#makes sure we have the right input

	addi	$t4,$t4,1	#increments i
	beq	$t4,$t2,gameOverLockpick	#if user input is the same amount as number of guesses then we get a game over screen

	jal	guessApproximation	#$t3 contains the abs value of how close the guess is (uses an abs formula to calcualte this)


	beq	$t1,$t0,correctGuessLockpick	#if the correct lockpick get a victory screen
	beq	$t3,1,close1	#else print cascading statements, as well as play jingles that sound better the closer the guess
	beq	$t3,2,close2
	beq	$t3,3,close3
	beq	$t3,4,close4
	j	notClose	#if user is completely off play wrong jingle and give a hint
inputErrorLP:
	li	$v0,4
	la	$a0,errorDiffSelect
	syscall			#prints error

	subi	$t4,$t4,1

	j	lockPickLoop	#sub guess and returns user to menu
RNGlockPickLoc:
			#generates random numbers from 1-10 and stores them in t0
	li	$v0,42	#you do not need to seed a value with time you can just do rng generator 0
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,9	#a1 is the upper range of values, mars does from 0-upperrange
	syscall
	addi	$t0,$a0,1

	jr	$ra
RNGguess:
			#generates random numbers from 2-4 and stores them in t2
	li	$v0,42	#you do not need to seed a value with time you can just do rng generator 0
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,2	#a1 is the upper range of values, mars does from 0-upperrange
	syscall
```

```
        addi    $t2,$a0,3       #due to a bug need to make it 3 to get range 2-4

        jr      $ra
guessApproximation:
        sub     $t3,$t0,$t1     #uses the absolute value of closeness to play jingle/approximation
        abs     $t3,$t3

        jr      $ra     #returns
gameOverLockpick:
        li      $v0,4
        la      $a0,lockPickGameOver        #user fails and gets game over screen
        syscall

        subi    $s0,$s0,50      #fine user 50 dollars
        jal     gameOverMidi        #play failure melody

        j       dayLightCycle#increment gampley loop
correctGuessLockpick:

        li      $v0,4
        la      $a0,lockPickCorrect
        syscall                 #victory screen

        jal     correctGuessMidi        #play victory song
        addi    $s0,$s0,150     #add 150 dollars to  user

        j       dayLightCycle#increment gameplay loop
close1:
        li      $v0,4   #print a flavor text
        la      $a0,lockPickClose1
        syscall

        jal     close1Midi      #play jingle that sounds almost right

        j       lockPickLoop  #loop
close2:
        li      $v0,4   #flavor text
        la      $a0,lockPickClose2
        syscall
```

```
        jal     close2Midi      #Play jingle that sounds closer

        j       lockPickLoop  #loop
close3:
        li      $v0,4
        la      $a0,lockPickClose3    #flavor text
        syscall

        jal     close3Midi      #play jingle that sounds kind of close

        j       lockPickLoop  # loop
close4:
        li      $v0,4
        la      $a0,lockPickClose4    #same as previous functions, just plays closer melody
        syscall

        jal     close4Midi

        j       lockPickLoop
notClose:
        li      $v0,4
        la      $a0,lockPickNotClose
        syscall                 #if not close give player a hint

        jal     notCloseMidi  #plays completly wrong jingle

        j       lockPickLoop

gameOverMidi:
        jr      $ra
correctGuessMidi:
                        #This is the victory jingle, each note is coded and subsequently time using
the respective syscalls



                        #first note played by program seems off always play one note before cont.
        li      $v0,31 #syscall for midi        #syscall for midi
        li      $a0, 71         #frequeuncy of note which is pitch (how high/low)
        li      $a1,1000        #duration in ms (1e-3)
        li      $a2,0   #Instrument Number (piano)
```

```
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 73		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 75#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall
		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 76#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
```

```
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 78#pitch		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 80#pitch		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 82		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
```

```
	li	$a0, 83		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	jr	$ra
close1Midi:
			#This is the close guess jingle, each note is coded and subsequently timed
sing the respective syscalls

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 71		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for
delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 73		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for
delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 75#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
```

```
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall
		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 76#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 78#pitch		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 80#pitch		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
```

```
	syscall

		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 82		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	jr	$ra
close2Midi:

			#This is the close guess jingle, each note is coded and subsequently timed sing the respective syscalls
	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 71		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 73		#frequeuncy of note which is pitch (how high/low)
```

```
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 75#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 76#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 78#pitch		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall
```

```
	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	jr	$ra
close3Midi:
```

```
                #This is the close guess jingle, each note is coded and subsequently timed
sing the respective syscalls
        li      $v0,31 #syscall for midi        #syscall for midi
        li      $a0, 71         #frequeuncy of note which is pitch (how high/low)
        li      $a1,1000        #duration in ms (1e-3)
        li      $a2,0   #Instrument Number (piano)
        li      $a3,127         #volume
        syscall

        li      $v0,32 #Since sleep thread is not accurate we must compensate      #syscall for
delay, delays between notes not 100% accurate
        li      $a0, 1000       #in miliseconds
        syscall

        li      $v0,31 #syscall for midi        #syscall for midi
        li      $a0, 73         #frequeuncy of note which is pitch (how high/low)
        li      $a1,1000        #duration in ms (1e-3)
        li      $a2,0   #Instrument Number (piano)
        li      $a3,127         #volume
        syscall

        li      $v0,32 #Since sleep thread is not accurate we must compensate      #syscall for
delay, delays between notes not 100% accurate
        li      $a0, 1000       #in miliseconds
        syscall

        li      $v0,31 #syscall for midi        #syscall for midi
        li      $a0, 75#frequeuncy of note which is pitch (how high/low)
        li      $a1,1000        #duration in ms (1e-3)
        li      $a2,0   #Instrument Number (piano)
        li      $a3,127         #volume
        syscall

        li      $v0,32 #Since sleep thread is not accurate we must compensate      #syscall for
delay, delays between notes not 100% accurate
        li      $a0, 1000       #in miliseconds
        syscall

        li      $v0,31 #syscall for midi        #syscall for midi
        li      $a0, 100        #frequeuncy of note which is pitch (how high/low)
```

```
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for
delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for
delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for
delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall
```

```
	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	jr	$ra
close4Midi:

			#This is the close guess jingle, each note is coded and subsequently timed sing the respective syscalls
	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 71	#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 73	#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
```

```
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100	#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall
		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100	#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
```

```
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall

	jr	$ra
notCloseMidi:
		#This is the not close guess jingle, each note sounds wrong, jingle is way off and hint is given to user
		#first note played by program seems wonky always play one note before cont.
	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127		#volume
	syscall
```

```
	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100	#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall
		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100	#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall
```

```
	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

		li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
	li	$a3,127	#volume
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#syscall for delay, delays between notes not 100% accurate
	li	$a0, 1000	#in miliseconds
	syscall

	li	$v0,31 #syscall for midi	#syscall for midi
	li	$a0, 100		#frequeuncy of note which is pitch (how high/low)
	li	$a1,1000	#duration in ms (1e-3)
	li	$a2,0	#Instrument Number (piano)
```

```
	li	$a3,127		#volume
	syscall

	jr	$ra
inputValidatorLP:
	blt	$t1,0,invalidInputLP	#makes sure input is valid
	bgt	$t1,9,invalidInputLP

	jr	$ra
invalidInputLP:
	li	$v0,4
	la	$a0,errorLP	#prints error message
	syscall

	subi	$t4,$t4,1	#takes away giess

	j	lockPickLoop
minigame3:
				#Dice gambling minigame, players can wager money then roll dice from 2-12.



	li	$v0,4
	la	$a0,diceGamblingIntro1
	syscall

	la	$a0,diceGamblingIntro2
	syscall

	la	$a0,diceGamblingIntro3		#prints out the intro dialogue
	syscall

	la	$a0,diceGamblingIntro4
	syscall

	la	$a0,diceGamblingIntro5		#finishes printing out intro dialogue
	syscall
```

```
diceLoop:
	blez	$s0,notEnoughMoney#main gameplay loop, checks if player has a valid amount
of money

	li	$v0,4
	la	$a0,diceGamblingLoop2	#prompts player for wager
	syscall

	li	$v0,1
	move	$a0,$s0		#prints out how much the user has
	syscall



	li	$v0,11 #ascii for $
	la	$a0,36
	syscall

	la	$a0,10 #ascii newline
	syscall

	li	$v0,5
	syscall
	move	$t0,$v0		#$t0 contains the wager amount
inputValidator:
	blez	$t0,invalidInputGambling
	bgt	$t0,$s0,invalidInputGambling		#makes sure that the user put a correct input

	li	$v0,4
	la	$a0,diceGamblingLoop4	#tells player to input a right number
	syscall

	jal	diceRoll	#loops back to function
	j	diceLoop
notEnoughMoney:
	li	$v0,4
	la	$a0,diceGamblingNoMoney
	syscall			#since player does not have money print out game over screen

	j	dayLightCycle#return player to main gaemplay loop
```

```
invalidInputGambling:
        li      $v0,4
        la      $a0,dicegamblingError          #invalid input message
        syscall

        li      $v0,5
        syscall

        j       inputValidator #loops until valid input is placed

diceRoll:
                        #t0 contains the wager amount
                        #t1 contains dice
                        #t2 contains point
                        #dice is seeded with a syscall values 2-12
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)

        li      $v0,4
        la      $a0,rollDialouge        #wait loop so player rolls
        syscall

        li      $v0,12 #char so game does not crash
        syscall
                        #range is from 1-12
        li      $v0,42 #you do not need to seed a value with time you can just do rng generator 0
        li      $a0,0   #this is the slected rng generator, generator 0 automitcally seeds for you
        li      $a1,10 #a1 is the upper range of values, mars does from 0-upperrange
        syscall
        addi    $t1,$a0,2       #add 2 to increase bounds

        li      $v0,4
        la      $a0,rollDialouge1       #dialogue
        syscall

        li      $v0,1   #print number rolled
        move    $a0,$t1
        syscall

        li      $v0,11 #print newline
```

```
	la	$a0,10
	syscall

	beq	$t1,7,diceWin #if the player rolls 7 or 11 its an automatic win
	beq	$t1,11,diceWin
	beq	$t1,2,diceLoss #if the player rolls a 2,3,12 its a loss
	beq	$t1,3,diceLoss
	beq	$t1,12,diceLoss		#else whatever number the player rolled becomes the 'point' and game continues, player must roll the point before a 7 to win

	lw	$ra, 0($sp)
	addi	$sp, $sp,4
	jr	$ra
dicePoint:
	li	$v0,4
	la	$a0,pointDice #lets player know they enter the point phase
	syscall


	move	$t2,$t1 #stores the point int for later use

	li	$v0,4
	la	$a0,pointDescriber	#continues describing
	syscall

	li	$v0,1
	move	$a0,$t2#prints the point
	syscall

	li	$v0,11 #char input so user rolls when pressing button
	li	$a0, 10
	syscall
dicePointLoop:
	li	$v0,4
	la	$a0,dicePointLoopDialouge	#repeats point phase until player wins or loses
	syscall

	li	$v0,12 #char input so user rolls when pressing button
	syscall
```

```
	li	$v0,42 #you do not need to seed a value with time you can just do rng generator 0
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,10 #a1 is the upper range of values, mars does from 0-upperrange
	syscall
	addi	$t1,$a0,2	#seeds value again for safety

	beq	$t1,7,diceLoss
	beq	$t1,$t2,diceWin		#if player rolls a 7 they lose, if they roll their point they win

	li	$v0,4
	la	$a0,dicePointLoopDialouge1 #ocontinue looping
	syscall

	li	$v0,1
	move	$a0,$t1#prints numbr rolled
	syscall

	li	$v0,11 #print newline
	la	$a0,10
	syscall

	li	$v0,4
	la	$a0,dicePointLoopDialouge2 #continues looping
	syscall

	j	dicePointLoop
diceWin:
	li	$v0,4
	la	$a0,winDice	#if player wins prompt victory message and match player wager
	syscall
	add	$s0,$s0,$t0

	la	$a0,returnDialogue	#if player wants to keep playing he can do so
	syscall
	li	$v0,5	#player must press 1 to continue playing
	syscall
	beq	$v0,1,dayLightCycle	#increments daylight cycle if player does not want to keep
playing

	j	diceLoop
```

```
diceLoss:
	li	$v0,4
	la	$a0,lossDice	#if player loses play loser message
	syscall
	sub	$s0,$s0,$t0	#take away wager from player

	la	$a0,returnDialogue
	syscall
	li	$v0,5	#prompt user to move on to the next day or keep playing
	syscall
	beq	$v0,1,dayLightCycle	#return to main menu if player doesnt want to play else contineu playing


	j	diceLoop
minigame4:


					#minigame 4 is about a player practicing piano with MMIO inputs and then trying to recreate the music heard
	li	$v0,4
	la	$a0,introDialogue11	#intro dialogue for minigame 4
	syscall

	la	$a0,introDialogue1	#print dialogue
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate
	li	$a0,10000		#I faked microphone input by sleeping the thread for 10 seconds
	syscall

	li	$v0,4
	la	$a0,introDialogue2	#prompts user for fake microphone input :)
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate
	li	$a0,10000	#wait for 10 seconds
	syscall
```

```
	li	$v0,4
	la	$a0,introDialogue3	#let the player know to open MMIO tool
	syscall

	jal	practiceKeyboard	#first player gets to practice using the piano
	jal	playSong	#player will then hear a song
	jal	realKeyboard	#Player will then try to play song
	jal	scoring	#if the player gets enough notes right they get a good score if not they fail



	addi	$s4,$0,4
	j	calljobMenu

playSong:

	addi	$sp,$sp,-4	#save return adress to stack
	sw	$ra,0($sp)

	li	$v0,4
	la	$a0,minigameStartDialogue	#Prints out more dialogue
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate
	li	$a0,10000	#waits for "microphone input" by sleeping thread
	syscall

	li	$v0,	31	#syscall for midi	#syscall for midi
	li	$a0,	61	#pitch	#pitch
	li	$a1,300	#ms duration but not accurate	#duration in ms, its not accurate
	li	$a2,1	#piano instrument
	li	$a3,	110	#volume	#volume of instrument
	syscall

	li	$v0,32 #Since sleep thread is not accurate we must compensate	#since ms delay its not accurate we compenstate by sleeping thread
	li	$a0,300	#ms duration but not accurate# .3 seconds
```

```
syscall

li      $v0,    31      #syscall for midi
li      $a0,    78      #pitch
li      $a1,100                 #ms duration
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
li      $a0,100
syscall

li      $v0,    31      #syscall for midi
li      $a0,    80      #pitch
li      $a1,700         #ms duration but not accurate
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
li      $a0,700         #ms duration but not accurate
syscall

li      $v0,    31      #syscall for midi
li      $a0,    77      #pitch
li      $a1,120
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
li      $a0,120
syscall

li      $v0,    31      #syscall for midi
li      $a0,    78      #pitch
li      $a1,200         #ms duration but not accurate
li      $a2,1
li      $a3,    110     #volume
```

```
syscall

li	$v0,32 #Since sleep thread is not accurate we must compensate
li	$a0,200		#ms duration but not accurate
syscall

li	$v0,	31	#syscall for midi
li	$a0,	80	#pitch
li	$a1,700		#ms duration but not accurate
li	$a2,1
li	$a3,	110	#volume
syscall

li	$v0,32 #Since sleep thread is not accurate we must compensate
li	$a0,700		#ms duration but not accurate
syscall

li	$v0,	31	#syscall for midi
li	$a0,	77	#pitch
li	$a1,100
li	$a2,1
li	$a3,	110	#volume
syscall

li	$v0,32 #Since sleep thread is not accurate we must compensate
li	$a0,100
syscall

li	$v0,	31	#syscall for midi
li	$a0,	78	#pitch
li	$a1,200		#ms duration but not accurate
li	$a2,1
li	$a3,	110	#volume
syscall

li	$v0,32 #Since sleep thread is not accurate we must compensate
li	$a0,200		#ms duration but not accurate
syscall

li	$v0,	31	#syscall for midi
```

```
li      $a0,    80      #pitch
li      $a1,300         #ms duration but not accurate
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
li      $a0,300         #ms duration but not accurate
syscall

li      $v0,    31      #syscall for midi
li      $a0,    77      #pitch
li      $a1,200         #ms duration but not accurate
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
li      $a0,200         #ms duration but not accurate
syscall

li      $v0,    31      #syscall for midi
li      $a0,    78      #pitch
li      $a1,100
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
li      $a0,100
syscall

li      $v0,    31      #syscall for midi
li      $a0,    80      #pitch
li      $a1,700         #ms duration but not accurate
li      $a2,1
li      $a3,    110     #volume
syscall

li      $v0,32 #Since sleep thread is not accurate we must compensate
```

```
        li      $a0,700         #ms duration but not accurate
        syscall

        li      $v0,    31      #syscall for midi
        li      $a0,    77      #pitch
        li      $a1,100
        li      $a2,1
        li      $a3,    110     #volume
        syscall

        li      $v0,32 #Since sleep thread is not accurate we must compensate
        li      $a0,100
        syscall

        li      $v0,    31      #syscall for midi
        li      $a0,    77      #pitch
        li      $a1,100
        li      $a2,1
        li      $a3,    110     #volume
        syscall

        li      $v0,32 #Since sleep thread is not accurate we must compensate
        li      $a0,200         #ms duration but not accurate
        syscall

        li      $v0,    33
        li      $a0,    80      #pitch
        li      $a1,1000
        li      $a2,1
        li      $a3,    110     #volume
        syscall


        lw      $ra,0($sp)      #pop stack and return
        addi    $sp,$sp,4
        jr      $ra

practiceKeyboard:
        addi    $sp,$sp,-4      #save return adress to stack
        sw      $ra,0($sp)
```

```
practiceLoop:
        li      $v0,4
        la      $a0,practiceDialogue  #print out user dialogue
        syscall

        jal     checkReadyBit         #so this function continously checks that ready bit is ready.
Once its ready it stores the value and compares it to the correpsonding key

        beq     $t0,'0', exitPractice
        beq     $t0,'1', key1
        beq     $t0,'2', key2   #corresponding keys to check values
        beq     $t0,'3', key3
        beq     $t0,'4', key4

        j       practiceLoop  #continously loops

exitPractice:
        lw      $ra,    0($sp) #once exit pop stack and return
        addi    $sp,$sp,4
        jr      $ra

key1:
        li      $v0,    31      #syscall for midi
        li      $a0,    61      #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,    110     #volume
        syscall
        j       practiceLoop

key2:
        li      $v0,    31      #syscall for midi
        li      $a0,    77      #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,    110     #volume
        syscall
        j       practiceLoop
```

```
key3:
        li      $v0,    31      #syscall for midi
        li      $a0,    78      #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,    110     #volume
        syscall
        j       practiceLoop

key4:
        li      $v0,    31      #syscall for midi
        li      $a0,    80      #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,    110     #volume
        syscall
        j       practiceLoop

checkReadyBit:
        lui     $t2,0xffff      #this is the memory adress

checkReadyBitLoop:
        lw      $t3,0($t2)      #grab the mmemory adress
        andi    $t3,$t3,1       #and it so we only have the last bit
        beq     $t3,0,checkReadyBitLoop     #check that the bit is ready
        lb      $t0,4($t2)      #if ready we grab the input
        jr      $ra     #return

realKeyboard:

        addi    $sp,$sp,-4      #save to stack
        sw      $ra,0($sp)


        la      $t8,userInput   #to save user input we use malloc space
        li      $t1,0   #initialize to 0

        li      $v0,4
        la      $a0,realDialogue        #let player know that practice is over
        syscall
```

```
keyLoop:
        jal     checkReadyBit1          #this function reads ands saves the first 15 notes


        beq     $t1,15, exitKeyLoop
        beq     $t0,'1',key11   #once 15 or greater is reached exit loop and compare scores
        beq     $t0,'2',key21
        beq     $t0,'3',key31
        beq     $t0,'4',key41   #compares notes

        j       keyLoop



key11:

        mul     $t3,$t1,4       #makes sure t1 is size of word each element in array is size of
word (4 bytes)
        add     $t4,$t8,$t3     #add the adress of user input and size to get current pointer
        li      $t7,1           #since key 1 is pressed store that value to compare score. (Key 1 is
value 1 key 2 is value 2 etc.. we then compare these values)
        sw      $t7,0($t4)      #store the value to the adress calculate
        addi    $t1,$t1,1       #increment

                                #play the pressed note
        li      $v0,31 #syscall for midi
        li      $a0,61 #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,110         #volume
        syscall
        j       keyLoop

key21:
        mul     $t3,$t1,4       #makes sure t1 is size of word each element in array is size of
word (4 bytes)
        add     $t4,$t8,$t3     #add the adress of user input and size to get current pointer
        li      $t7,2   #since key 2 is pressed store that value to compare score. (Key 1 is value 1
key 2 is value 2 etc.. we then compare these values)
```

```
        sw      $t7,0($t4)      #store value at calc address
        addi    $t1,$t1,1       #increement


        li      $v0,31 #syscall for midi
        li      $a0,77 #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,110         #volume
        syscall
        j       keyLoop

key31:
        mul     $t3,$t1,4       #makes sure t1 is size of word each element in array is size of
word (4 bytes)
        add     $t4,$t8,$t3     #add the adress of user input and size to get current pointer
        li      $t7,3   #since key 3 is pressed store that value to compare score. (Key 1 is value 1
key 2 is value 2 etc.. we then compare these values)
        sw      $t7,0($t4)      #store value at calc address
        addi    $t1,$t1,1       #increement


        li      $v0,31 #syscall for midi
        li      $a0,78 #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,110         #volume
        syscall
        j       keyLoop

key41:

        mul     $t3,$t1,4       #makes sure t1 is size of word each element in array is size of
word (4 bytes)
        add     $t4,$t8,$t3     #add the adress of user input and size to get current pointer
        li      $t7,4   #since key 4 is pressed store that value to compare score. (Key 1 is value 1
key 2 is value 2 etc.. we then compare these values)
        sw      $t7,0($t4)      #store value at calc address
        addi    $t1,$t1,1       #incre,emt
```

```
        li      $v0,31 #syscall for midi
        li      $a0,80 #pitch
        li      $a1,300         #ms duration but not accurate
        li      $a2,1
        li      $a3,110         #volume
        syscall
        j       keyLoop

checkReadyBit1:
        lui     $t2,0xffff      #checks ready nit same as before

checkReadyBitLoop1:
        lw      $t3,0($t2)      #grab the mmemory adress
        andi    $t3,$t3,1       #and it so we only have the last bit
        beq     $t3,0,checkReadyBitLoop1    #check that the bit is ready
        lb      $t0,4($t2)      #if ready we grab the input
        jr      $ra     #return
exitKeyLoop:
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4     #used to return
        jr      $ra
scoring:

        addi    $sp,$sp,-4
        sw      $ra,0($sp)      #save return to stack

        li      $t0,0   #i=0
        li      $t1,0   #score count, initialize at 0
scoreLoopRow:
        li      $t6,0   #j=0
                                #t0, t1, t6  are being used in score loop
                                #t3-5 are being used in arry adres

                                #t2 and t7 are being used to compare
                #use a nested for loop in order to calculate each array value incrementally
                #i=0, j=0
                #array adress takes in below arguements
                # a0 is userRow
                # a1 is userCol
```

```
                        #a2 is num of col
                        #a3 is base adress
scoreLoopCol:
        move   $a0,$t0#current row
        move   $a1,$t6#current col
        li     $a2,5   #num of col in array
        la     $a3,notePat     #base adress (NOTEPAT)
        jal    arrayAddress   #calculate current array dress store retunr in vo
        lw     $t2,0($v0)#store return value in s7

        move   $a0,$t0#current row
        move   $a1,$t6#current col
        li     $a2,5   #num of col in array
        la     $a3,userInput  #base adress (userInput)
        jal    arrayAddress   #calculate current array dress store retunr in vo
        lw     $t7,0($v0)#store return value in s7

        bne    $t2,$t7,skipNote        #if correct note add score

        addi   $t1,$t1,1

skipNote:
        addi   $t6,$t6,1
        blt    $t6, 5, scoreLoopCol  #next col

        addi   $t0,$t0,1
        blt    $t0,3,scoreLoopRow   #next row

        ble    $t1,5,badJob
        ble    $t1,10, okJob  #commpare score


        j      goodJob
badJob:
        li     $v0,4
        la     $a0,scoreBad  #print that player did bad job
        syscall
        j      exitScoring
okJob:
        li     $v0,4
```

```
        la      $a0,scoreOk    #print player did okay job and give 100 dollars
        syscall
        addi    $s0,$s0,100
        j       exitScoring

goodJob:
        li      $v0,4
        la      $a0,scoreGood          #print player did a good job and give 150
        syscall
        addi    $s0,$s0,150
exitScoring:
        lw      $ra,0($sp)
        addi    $sp,$sp,4       #pop stack and return
        jr      $ra
arrayAddress:                   #can use t3-t7
        addi    $sp,$sp,-4
        sw      $ra,0($sp)      #save return to stack

        move    $t3, $a3  # base adress
        mul     $t4, $a0, $a2  # row * numColumns
        add     $t4, $t4, $a1  # (row * numColumns) + col
        sll     $t4, $t4, 2  # int size (multiply by 4)
        add     $t3, $t3, $t4    #base+row * numColumns) + col
        move    $v0, $t3         #return on t3

        lw      $ra, 0($sp)
        addi    $sp, $sp, 4      #exit by popping stack
        jr      $ra
callfamMenu:
        jal     famMenu      #nested within to make sure ra is being popped correctly
        j       mainMenu
famMenu:
        addi    $sp, $sp, -4     #Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)

        beq     $s5, 1, fam1    #depending on the family size we jump to the different functions
        beq     $s5, 2, fam2
        beq     $s5, 3, fam3
endFamMenu:
        lw      $ra, 0($sp)
```

```
        addi    $sp, $sp, 4     #exit by popping stack
        jr      $ra
fam1:
        addi    $sp, $sp, -4    #Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)
        li      $v0, 4  #so we print out the son gui
        la      $a0, sonName
        syscall
        la      $a0, sonGUI1
        syscall
        la      $a0, sonGUI2
        syscall
        la      $a0, sonGUI3
        syscall
        la      $a0, sonGUI4  #continue printing out sonGUI
        syscall
        la      $a0, sonGUI5
        syscall
        la      $a0, sonGUI6
        syscall
        la      $a0, sonGUI7
        syscall
        la      $a0, sonGUI8
        syscall

        move    $a0,$s1         #push the amount of hunger to be
        jal     printHunger     #when we get to hunger we call a special function

        la      $a0, sonGUI10           #finish printing
        syscall

        lw      $ra, 0($sp)
        addi    $sp, $sp, 4     #pop and return
        jr      $ra

fam2:
        addi    $sp, $sp, -4    #Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)

        li      $v0, 4  #so we print out the son gui
```

```
la      $a0, sonName
syscall
la      $a0, sonGUI1
syscall
la      $a0, sonGUI2
syscall
la      $a0, sonGUI3
syscall
la      $a0, sonGUI4  #since family size is 2 we print out son and wife
syscall
la      $a0, sonGUI5
syscall
la      $a0, sonGUI6
syscall
la      $a0, sonGUI7
syscall
la      $a0, sonGUI8
syscall

move    $a0,$s1
jal     printHunger     #call hunger function

la      $a0, sonGUI10
syscall

li      $v0, 4  #so now we print out the wife gui
la      $a0, wifeName
syscall
la      $a0, wifeGUI1
syscall
la      $a0, wifeGUI2
syscall
la      $a0, wifeGUI3
syscall
la      $a0, wifeGUI4
syscall
la      $a0, wifeGUI5
syscall
la      $a0, wifeGUI6
syscall
```

```
        la      $a0, wifeGUI7
        syscall
        la      $a0, wifeGUI8
        syscall

        move    $a0,$s2
        jal     printHunger     #calc wifes hunger and print

        la      $a0, wifeGUI10
        syscall



        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra       #pop and return

fam3:
        addi    $sp, $sp, -4      #Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)

        li      $v0, 4  #so we print out the son gui
        la      $a0, sonName
        syscall
        la      $a0, sonGUI1
        syscall
        la      $a0, sonGUI2
        syscall
        la      $a0, sonGUI3
        syscall
        la      $a0, sonGUI4  #we must now print son wife and daughter
        syscall
        la      $a0, sonGUI5
        syscall
        la      $a0, sonGUI6
        syscall
        la      $a0, sonGUI7
        syscall
        la      $a0, sonGUI8
        syscall
```

```
move   $a0,$s1          #calculate huunger bars and print
jal    printHunger

la     $a0, sonGUI10
syscall

li     $v0, 4  #so now we print out the wife gui
la     $a0, wifeName
syscall
la     $a0, wifeGUI1
syscall
la     $a0, wifeGUI2
syscall
la     $a0, wifeGUI3
syscall
la     $a0, wifeGUI4
syscall
la     $a0, wifeGUI5#finish printing
syscall
la     $a0, wifeGUI6
syscall
la     $a0, wifeGUI7
syscall
la     $a0, wifeGUI8
syscall

move   $a0,$s2          #push what hunger value to print
jal    printHunger    #calc hunger and print

la     $a0, wifeGUI10
syscall

li     $v0, 4  #so now we print out the daughter gui
la     $a0, daughterName
syscall
la     $a0, daughterGUI1
syscall
la     $a0, daughterGUI2
syscall
la     $a0, daughterGUI3
```

```
        syscall
        la      $a0, daughterGUI4
        syscall
        la      $a0, daughterGUI5
        syscall
        la      $a0, daughterGUI6
        syscall
        la      $a0, daughterGUI7
        syscall
        la      $a0, daughterGUI8
        syscall

        move    $a0,$s3
        jal     printHunger     #calculate hunger and rpint

        la      $a0, daughterGUI10   #fiinish printing
        syscall



        lw      $ra, 0($sp)
        addi    $sp, $sp, 4      #return with stack
        jr      $ra
printHunger:
        addi    $sp,$sp, -4      #save ra to stack
        sw      $ra, 0($sp)

        move    $t1,$a0#contains the hunger value that will be printed (for example moms hunger
value)



        li      $v0, 4  #print out start hunger label
        la      $a0, hunger1
        syscall

        move    $a0,$t1
        jal     forLoopHungerPrint   #we then call a function which prints out hashes for each
hunger and spaces for empty hunger

        li      $v0, 4
        la      $a0, hunger2   #finish printing
```

```
	syscall

	lw	$ra,0($sp)	#return with stack
	addi	$sp, $sp, 4
	jr	$ra
forLoopHungerPrint:  #function first stores how much hunger bars the NPC has, then prints out # accordingly.
			#afterwards function calc, the amount of blank spaces being printed by subtracting current hunger-10(the max hunger avalialbe)
	move	$t0,$a0#calculate what family member to pr
	li	$t7,10
	sub	$t1,$t7, $t0	#t1=hunger bar-10 which is the blank spaces
	li	$t3,0	#t3 is our I
	li	$t4,0

hashPrint:
	beq	$t3,$t0, spacePrint	#if t3(which is () is greater than or equal to t0(which is the hunger) we exit and

	li	 $v0,4  #print out the hunger
	la	$a0,hunger3
	syscall

	addi	$t3,$t3,1	#add to I
	j	hashPrint	#repeat for loop
spacePrint:
	li	$t4,0
spacePrintLoop:
	beq	$t4,$t1, endHungerBarPrint	#if t4 (which is i=0) is greater than or equal to t1(which is the blank space) we exit and return to main menu

	li	$v0,4
	la	$a0,hunger4	#print out the last part of hunger
	syscall

	addi	$t4,$t4,1
	j	spacePrintLoop		#incrememnt and loop
moneyMenu:
	li	$v0,4
	la	$a0, moneyStatus	#prints out dialogue string
```

```
        syscall

        li      $v0,1
        move    $a0,$s0         #prints current money
        syscall

        li      $v0,4
        la      $a0, moneyStatus1     #reminds players of the goal
        syscall

        j       mainMenu      #return
marketMenu:
        li      $v0,4
        la      $a0,marketMenuUI    #prints out market menu
        syscall

        la      $a0, marketMenuUI1
        syscall
        li      $s6,0   #checks for valid input using macro
        li      $s7,3
        ioCheck
        move    $t0, $v0        #stores return value

        beq     $t0, 0, mainMenu       #exit case

        li      $v0, 4
        la      $a0,marketMenuUI2   #prints out prompt for user
        syscall
        li      $s6,1
        li      $s7,3
        ioCheck         #validates that prompt is valid
        move    $t1, $v0        #t1 contains family member t0 contains meal

        beq     $t0,0,mainMenu
        blt     $t0,1,wrongMeal       #validates input or exit
        bgt     $t0,3,wrongMeal

        beq     $t0, 1, marketMenuSmall
        beq     $t0, 2, marketMenuMedium   #acess menu according to user selcted
        beq     $t0, 3, marketMenuFeast
```

```
	ble	$t1, 0, invalidFamilyMember #in case input is not valid print out error and
continue
	bgt	$t1, $s5, invalidFamilyMember
wrongMeal:
	li	$v0,4
	la	$a0,errorDiffSelect	#custom input validator
	syscall
	j	marketMenu
marketMenuSmall:


	blt	$s0, 10,notEnoughMoneyMarket	#if money is not enough print erro
	subi	$s0,$s0,10	#sub money cost
	li	$t3, 1	#t3 contains how much hunger to fill
	beq	$t1, 1, feedSon
	beq	$t1, 2, feedWife	#depending on selected family member jump to function
	beq	$t1, 3, feedDaughter

	j	familyFeed	#loop to feed
marketMenuMedium:

	blt	$s0, 25,notEnoughMoneyMarket	#not enough money, print error
	subi	$s0,$s0,25	#sub money
	li	$t3, 3	#t3 contains how much hunger to fill
	beq	$t1, 1, feedSon
	beq	$t1, 2, feedWife	#depending on family member selcted jump
	beq	$t1, 3, feedDaughter

	j	familyFeed	#Loop to feed
marketMenuFeast:

	blt	$s0, 35,notEnoughMoneyMarket	#jump if not enough money and print error
	subi	$s0,$s0,35	#sub money from global variable
	li	$t3, 10 #t3 contains how much hunger to fill
	beq	$t1, 1, feedSon
	beq	$t1, 2, feedWife	#beq based on
	beq	$t1, 3, feedDaughter

	j	familyFeed	#j to feed function
```

```
familyFeed:
	beq	$t1, 1,feedSon
	beq	$t1, 2,feedWife		#JUMP to family member being fed
	beq	$t1,3,feedDaughter

	lw	$ra, 0($sp)
	addi	$sp,$sp,4	#pops return register from stack and returns
	jr	$ra
feedSon:
	add	$s1,$s1,$t3	#add amount being fed to son global variable
	li	$v0, 4
	la	$a0, marketMenuUI4	#print after done
	syscall

	bgt	$s1,10,foodCapSon	#if more than 10 cap it
	j	marketMenu
feedWife:
	add	$s2,$s2,$t3	#add new hunger
	li	$v0, 4
	la	$a0, marketMenuUI5	#print out escape string
	syscall

	bgt	$s2,10,foodCapWife	#If more than 10 cap it
	j	marketMenu
feedDaughter:
	add	$s3,$s3,$t3
	li	$v0, 4	#see above comments, apply to this function but for daughter
	la	$a0, marketMenuUI6
	syscall

	bgt	$s3,10,foodCapDaughter
	j	marketMenu	#return
notEnoughMoneyMarket:
	li	$v0,4
	la	$a0,marketMenuUI3	#not enough money print errro
	syscall

	j	marketMenu	#return
invalidFamilyMember:
	li	$v0,4
```

```
        la      $a0, invalidFamilySizerError #if not 1-3 print invalid error
        syscall

        j       marketMenu
foodCapSon:
        li      $s1,10 #if greater than 10 cap at 10
        li      $v0,4
        la      $a0, marketMenuUI4  #print out menu message
        syscall

        j       marketMenu    #return
foodCapWife:
        li      $s2,10 #cap at 10
        li      $v0,4
        la      $a0, marketMenuUI4  #print message
        syscall

        j       marketMenu    #return
foodCapDaughter:
        li      $s3,10
        li      $v0,4   #cap at 10
        la      $a0, marketMenuUI4
        syscall

        j       marketMenu    #return
jobMenu:
        move    $t0,$s4#load the day into the register
        beq     $t0, 0, minigame1
        beq     $t0, 1, minigame2
        beq     $t0, 2, minigame3       #load different minigame based on day
        beq     $t0, 3, minigame4
        beq     $t0, 4,final

        j       mainMenu
RNGChildren:
        addi    $sp, $sp, -4     #Since its a nested function we must store our RA to the stack
        sw      $ra, 0($sp)

                        #generates random numbers from 0-1 and stores them in t7
        li      $v0,42 #you do not need to seed a value with time you can just do rng generator 0
```

```
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,2	#a1 is the upper range of values, mars does from 0-upperrange
	syscall
			#a0 will now contain a valu efrom 0-upperrange
	move	$t7,$a0#load into t7 which is the return function

	lw	$ra, 0($sp)
	addi	$sp, $sp, 4	#pops return register from stack and returns
	jr	$ra
RNGAdult:
	addi	$sp, $sp, -4	#Since its a nested function we must store our RA to the stack
	sw	$ra, 0($sp)
			#generates random numbers from 0-3 and stores them in t7
	li	$v0,42 #you do not need to seed a value with time you can just do rng generator 0
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,3	#a1 is the upper range of values, mars does from 0-upperrange
	syscall
			#a0 will now contain a valu efrom 0-upperrange
	move	$t7,$a0

	lw	$ra, 0($sp)
	addi	$sp, $sp, 4	#pops return register from stack and returns
	jr	$ra
dayLightCycle:
	addi	$s4,$s4,1	#increment day
	jal	hungerCycle	#do a hunger depletion on each family member


	j	mainMenu	#go to menu
hungerCycle:
	addi	$sp, $sp, -4	#Since its a nested function we must store our RA to the stack
	sw	$ra, 0($sp)

	blez	$s1,gameoverHunger
	blez	$s2,gameoverHunger	#If any of the family member run out of hunger print a
game over screen and end program
	blez	$s3,gameoverHunger

	beq	$s5,1, hungerCycleSize1
```

```
	beq	$s5,2, hungerCycleSize2		#depending on the number of family members decrement hunger
	beq	$s5,3, hungerCycleSize3

	j	hungerCycleEnd		#return
hungerCycleSize1:
	jal	RNGChildren
			#t7 now contians random value from 0-1
			#t0 contains amount of hunger to be decremented
	li	$t0,4	#since children go hungry faste they decrease 3 hunger bars per cycle
	beq	$t7,0,hungerDecrementSon	#jump to the decrement of hunger

	j	hungerCycleEnd		#pop and return
hungerCycleSize2:

	jal	RNGChildren
			#t7 now contians random value from 0-1
			#t0 contains amount of hunger to be decremented
	li	$t0,4	#amount of hunger to be deducted(children lose hunger faster)
	beq	$t7,0,hungerDecrementSon

	jal	RNGAdult	#calculate if hunger is depleted, if return value is 0 decrement hunger

	li	$t0,3
	beq	$t7,0,hungerDecrementWife	#decrement if 0

	j	hungerCycleEnd		#return
hungerCycleSize3:
	jal	RNGChildren
			#t7 now contians random value from 0-1
			#t0 contains amount of hunger to be decremented
	li	$t0,4	#since children go hungry faste they decrease 3 hunger bars per cycle
	beq	$t7,0,hungerDecrementSon

	jal	RNGChildren	#calculate if hunger is depleted, if return value is 0 decrement hunger

	li	$t0,4	#amount to be decremented
	beq	$t7,0,hungerDecrementDaughter
```

```
	jal	RNGAdult	#calculate if hunger is depleted, if return value is 0 decrement
hunger

	li	$t0,3	#amount to be decrement
	beq	$t7,0,hungerDecrementWife

	j	hungerCycleEnd		#return
hungerDecrementSon:
	sub	$s1,$s1,$t0	#t0 contains how much hunger is to be decremented
			#s1 is hunger of son\
	blez	$s1,gameoverHunger
	j	hungerCycleEnd
hungerDecrementWife:
	sub	$s2,$s2,$t0	#t0 contains how much hunger is to be decremented
			#s2 is hunger of wife
	blez	$s2,gameoverHunger
	j	hungerCycleEnd		#return
hungerDecrementDaughter:
	sub	$s3,$s3,$t0	#t0 contains how much hunger is to be decremented
			#s1 is hunger of daughter
	blez	$s3,gameoverHunger
	j	hungerCycleEnd		#return
gameOverHunger:
	li	$v0,4
	la	$a0,gameoverHunger	#print out game over
	syscall

	li	$v0,10 #end program
	syscall
hungerCycleEnd:
	lw	$ra, 0($sp)	#pops return register from stack and returns
	addi	$sp, $sp, 4
	jr	$ra
endHungerBarPrint:
	jr	$ra
minigame1:
	li	$t0,0	#score
	li	$t1,0	#customerAmount
	li	$t2,0	#productAmount
```

```
	li	$t3,0	#cashierChange
	li	$t4,0	#correctAmount
	li	$t5,0	# Set i=0

	li	$v0,4
	la	$a0, cashierGame0
	syscall #print out tutorial string
cashierGameLoop:
	li	$v0,42 #you do not need to seed a value with time you can just do rng generator 0
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,99 #a1 is the upper range of values, mars does from 0-upperrange
	syscall #generate value 0-99


	move	$t2,$a0
	addi	$t2,$t2,1		#make it so product amount is values from 1-100

	li	$v0,42 #you do not need to seed a value with time you can just do rng generator 0
	li	$a0,0	#this is the slected rng generator, generator 0 automitcally seeds for you
	li	$a1,49 #a1 is the upper range of values, mars does from 0-upperrange
	syscall

	move	$t1, $a0		#grab the random amount
	addi	$t1,$t1,1		#the customer will give you a number between 1-50 that has to be
greater than product amount
	add	$t1,$t1,$t2		#t1 now contains the total amount of the product and the customer
change
correctAmount:
	sub	$t4,$t1,$t2		#to get the correct amount for comparison undo sub and store it in
t4
instructionSections:
	li	$v0,4
	la	$a0, cashierGame1
	syscall #print out first string

	move	$a0, $t2
	li	$v0,1#print out product amount
	syscall

	li	$v0 11  #print $
```

```
li        $a0,36
syscall

li        $v0 11  #print newline
li        $a0,10
syscall

li        $v0,4
la        $a0,cashierGame2
syscall #print out second string

move   $a0,$t1#print out customer amount
li        $v0,1
syscall

li        $v0 11  #print $
li        $a0,36
syscall

li        $v0 11  #print newline
li        $a0,10
syscall

li        $v0,4
la        $a0, cashierGame3
syscall #print out third string

li        $v0,5   #read user input
syscall

move   $t3,$v0          #store it in t3

bne      $t3,$t4,incorrectAmount        #if not the correct amount take branch
addi     $t0,$t0,1         #if correct amount increment score by 1

addi     $t5,$t5,1         #increment i (loop counter)
beq      $t5,5,endCashierLoop#end if 5 customers reached

li        $v0,4
la        $a0, cashierGame4
```

```
        syscall #print out sucess string
        j       cashierGameLoop     #continue looping
incorrectAmount:
        li      $v0,4
        la      $a0, cashierGame5
        syscall #print out failure string

        addi    $t5,$t5,1
        beq     $t5,5,endCashierLoop#still increment i
        j       cashierGameLoop     #loop
endCashierLoop:
        blez    $t0,scoreCashier0       #depending on the score, give respective amount
        beq     $t0,1,scoreCashier1
        beq     $t0,2,scoreCashier2
        beq     $t0,3,scoreCashier3
        beq     $t0,4,scoreCashier4
        beq     $t0,5,scoreCashier5     #falls all the way through
scoreCashier0:
        li      $v0,4
        la      $a0,score0      #print out score string
        syscall

        li      $v0,4
        la      $a0,balance     #print out current balance
        syscall

        move    $a0,$s0
        li      $v0,1   #continue printing out balance
        syscall

        li      $v0,11
        li      $a0,10 #ascii newline
        syscall

        j       dayLightCycle#end minigame
scoreCashier1:
        addi    $s0,$s0,5       #increment money
        li      $v0,4
        la      $a0,score1      #print out score string
        syscall
```

```
	li	$v0,4
	la	$a0,balance	#print out current balance
	syscall

	li	$v0,11
	li	$a0,10 #ascii newline
	syscall

	move	$a0,$s0
	li	$v0,1	#continue printing out balance
	syscall

	li	$v0,11 #print newline
	li	$a0,10
	syscall

	j	dayLightCycle#end minigame
scoreCashier2:
	addi	$s0,$s0,10	#increment money
	li	$v0,4
	la	$a0,score2	#print out score string
	syscall

	li	$v0,4
	la	$a0,balance	#print out current balance
	syscall

	li	$v0,11
	li	$a0,10 #ascii newline
	syscall


	move	$a0,$s0
	li	$v0,1	#continue printing out balance
	syscall

	li	$v0,11 #print newline
	li	$a0,10
	syscall
```

```
        j       dayLightCycle#end minigame
scoreCashier3:
        addi    $s0,$s0,15
        li      $v0,4
        la      $a0,score3      #see above comments
        syscall

        li      $v0,11
        li      $a0,10 #ascii newline
        syscall



        li      $v0,4
        la      $a0,balance     #see comments for score2
        syscall

        move    $a0,$s0
        li      $v0,1
        syscall

        li      $v0,11 #print newline
        li      $a0,10
        syscall
        j       dayLightCycle#return
scoreCashier4:
        addi    $s0,$s0,20
        li      $v0,4
        la      $a0,score4      #see above comments
        syscall

        li      $v0,4
        la      $a0,balance     #see coments for score 2 and 1
        syscall



        move    $a0,$s0
        li      $v0,1
        syscall
```

```
        li      $v0,11 #print newline
        li      $a0,10
        syscall
        j       dayLightCycle#return
scoreCashier5:
        addi    $s0,$s0,30      #incrememnt with bonus
        li      $v0,4
        la      $a0,score5      #print string
        syscall

        li      $v0,11
        la      $a0,balance     #print balance
        syscall



        move    $a0,$s0
        li      $v0,1
        syscall

        li      $v0,11 #print newline
        li      $a0,10
        syscall

        j       dayLightCycle#return
final:
        li      $v0,4
        la      $a0,finalDialogue       #print bitmap specifications
        syscall
        li      $v0,4
        la      $a0,finalDialogue1
        syscall
        li      $v0,5   #prompts the user if hes ready
        syscall

        blt     $s0,200, fail    #fail if less than 200 dollars


                #P backbone
        li      $s1,1   #x coordinate
        li      $s2,1   #y coordinate initial
```

```
li      $s3,6   #y coordinate final
baseAdress      #macro that initializes base adress
jal     drawVerticalLine
                #store y coord iniitla and y coordinate final in s6 and s7
                #store x in stack for later
                #P top line
li      $s4,1   #y coordinate
li      $s5,1   #x coordinate initial
li      $s6,3   #x coordinate final
baseAdress
jal     drawHorizontalLine
                #P midline
li      $s4,3   #y coordinate
li      $s5,1   #x coordinate initial
li      $s6,2   #x coordinate final
baseAdress
jal     drawHorizontalLine
                        #P frontbone
li      $s1,3   #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,3   #y coordinate final
baseAdress
jal     drawVerticalLine
                #A leftbone
li      $s1,5   #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                #A top line
li      $s4,1   #y coordinate
li      $s5,6   #x coordinate initial
li      $s6,7   #x coordinate final
baseAdress
jal     drawHorizontalLine
                #A mid line
li      $s4,3   #y coordinate
li      $s5,5   #x coordinate initial
li      $s6,7   #x coordinate final
```

```
baseAdress
jal     drawHorizontalLine
                #S rightbone
li      $s1,7   #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                #S top line
li      $s4,1   #y coordinate
li      $s5,9   #x coordinate initial
li      $s6,11  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #S mid line
li      $s4,3   #y coordinate
li      $s5,10  #x coordinate initial
li      $s6,11  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #S leftbone
li      $s1,11  #x coordinate
li      $s2,3   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine
                #S bottom line
li      $s4,6   #y coordinate
li      $s5,9   #x coordinate initial
li      $s6,11  #x coordinate final
baseAdress
jal     drawHorizontalLine


                #P backbone
li      $s1,9   #x coordinate
li      $s2,1   #y coordinate initial
```

```
li      $s3,3   #y coordinate final
baseAdress
jal     drawVerticalLine

                #S top line
li      $s4,1   #y coordinate
li      $s5,13  #x coordinate initial
li      $s6,15  #x coordinate final
baseAdress
jal     drawHorizontalLine
                #S leftbone
li      $s1,13  #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,3   #y coordinate final
baseAdress
jal     drawVerticalLine

                #S mid line
li      $s4,3   #y coordinate
li      $s5,13  #x coordinate initial
li      $s6,15  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #S rightbone
li      $s1,15  #x coordinate
li      $s2,3   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine
                #S bottom line
li      $s4,6   #y coordinate
li      $s5,13  #x coordinate initial
li      $s6,15  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #! top
li      $s1,19  #x coordinate
li      $s2,1   #y coordinate initial
```

```
li      $s3,4   #y coordinate final
baseAdress
jal     drawVerticalLine

                #! bottom
li      $s1,19 #x coordinate
li      $s2,6   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                #! top
li      $s1,21 #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,4   #y coordinate final
baseAdress
jal     drawVerticalLine

                #! bottom
li      $s1,21 #x coordinate
li      $s2,6   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                #! top
li      $s1,23 #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,4   #y coordinate final
baseAdress
jal     drawVerticalLine

                #! bottom
li      $s1,23 #x coordinate
li      $s2,6   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine
```

```
	li	$v0,10
	syscall #finish

drawVerticalLine:
			#since we cant use t registers and we only have 2 s register we must store one value to stack
	addi	$sp,$sp,-4
	sw	$ra, 0($sp)	#save ra to stack

drawVerticalLineLoop:
	move	$a0,$s1		#load x coordinate
	move	$a1,$s2		#load current y coordinate

	pixelWidth	#load the vlaue of pixel width into a3

	jal	adressCalc	#calc the adress of the line

	move	$a0,$v0		#store the value of adress calculated
	colorPastelBlue		#initliaze a1 with color blue
	jal	loadColor	#paint the adress with color blue

	addi	$s2,$s2,1	#increment until end
	ble	$s2,$s3, drawVerticalLineLoop  #once end finish

	lw	$ra,0($sp)	#once finished pop stack and return
	addi	$sp,$sp,4
	jr	$ra
			#the function for calculating is Calculate address: base_addr + 4*(x + y*display width)
adressCalc:
	addi	$sp, $sp, -4
	sw	$ra, 0($sp)
			#a0 will be our temp holder v0 will be our return
			#a0=x a1=y

	mul	$a1,$a1,$a3	#mul our y coord with pixel width
	add	$a0,$a0,$a1	#add our x coord and our y coord
	mul	$a0,$a0,4	#mul by thje size of a word
	add	$a0,$a0,$a2	#add the x coord
	move	$v0,$a0			#return value
```

```
        lw      $ra, 0($sp)     #return by popping ra from stack
        addi    $sp,$sp,4
        jr      $ra
loadColor:
        addi    $sp, $sp, -4    #store our ra into stack
        sw      $ra, 0($sp)
                        #a0 is our adress and a1 is our colorPastelBlue
        sw      $a1,($a0)       #load the array with the color


        lw      $ra, 0($sp)
        addi    $sp,$sp,4       #pop our ra from stack
        jr      $ra
drawEnd:
        lw      $ra,0($sp)
        addi    $sp,$sp,4       #once finished pop stack and return
        jr      $ra

drawHorizontalLine:
                                #since the t registers arent saved
        addi    $sp,$sp,-4
        sw      $ra, 0($sp)     #stack

                        #S5 holds initial x coord, s6 holds final x coord, s4 is y coord

drawHorizontalLineLoop:
        move    $a0,$s5         #load current x coordinate
        move    $a1,$s4         #load y coordinate

        pixelWidth      #initilaize arguement with pizel width

        jal     horizontalAdressCalc #calcualte the adress of pizel

        move    $a0,$v0         #save returned adress
        colorPastelBlue         #initialize with color blue
        jal     horizontalLoadColor  #load color onto array

        addi    $s5,$s5,1       #increment loop
```

```
        ble     $s5,$s6, drawHorizontalLineLoop  #end loop

        lw      $ra,0($sp)      #pop stack and return
        addi    $sp,$sp,4
        jr      $ra
                #the function for calculating is Calculate address: base_addr + 4*(x +
y*display width)
horizontalAdressCalc:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)     #save ra onto stack
                #a0=x a1=y

        mul     $a1,$a1,$a3     #mul our x coord with pixel width
        add     $a0,$a0,$a1     #add our x coordinate onto the y
        mul     $a0,$a0,4       #mul by word size which is 4
        add     $a0,$a0,$a2     #add base
        move    $v0,$a0                 #store onto v0


        lw      $ra, 0($sp)
        addi    $sp,$sp,4       #push ra to stack
        jr      $ra
horizontalLoadColor:
        addi    $sp, $sp, -4    #save ra to stack
        sw      $ra, 0($sp)
                #a0 is our adress and a1 is our colorPastelBlue
        sw      $a1,($a0)       #store our color onto bitmap array


        lw      $ra, 0($sp)
        addi    $sp,$sp,4       #Pop ra to stack
        jr      $ra

fail:
                #F backbone
        li      $s1,1   #x coordinate
        li      $s2,1   #y coordinate initial
        li      $s3,6   #y coordinate final
        baseAdress
        jal     drawVerticalLine
```

```
                    #store y coord iniitla and y coordinate final in s6 and s7
                    #store x in stack for later
                    #F top line
li      $s4,1   #y coordinate
li      $s5,1   #x coordinate initial
li      $s6,3   #x coordinate final
baseAdress
jal     drawHorizontalLine
                    #F midline
li      $s4,3   #y coordinate
li      $s5,1   #x coordinate initial
li      $s6,2   #x coordinate final
baseAdress
jal     drawHorizontalLine
                    #A leftbone
li      $s1,5   #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                    #A top line
li      $s4,1   #y coordinate
li      $s5,6   #x coordinate initial
li      $s6,7   #x coordinate final
baseAdress
jal     drawHorizontalLine
                    #A mid line
li      $s4,3   #y coordinate
li      $s5,5   #x coordinate initial
li      $s6,7   #x coordinate final
baseAdress
jal     drawHorizontalLine
                    #A rightbone
li      $s1,7   #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine
```

```
                #I top line
li      $s4,1   #y coordinate
li      $s5,9   #x coordinate initial
li      $s6,11  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #I midbone
li      $s1,10  #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                #I bottom line
li      $s4,6   #y coordinate
li      $s5,9   #x coordinate initial
li      $s6,11  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #L leftbone
li      $s1,13  #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,6   #y coordinate final
baseAdress
jal     drawVerticalLine

                #L bottom line
li      $s4,6   #y coordinate
li      $s5,13  #x coordinate initial
li      $s6,15  #x coordinate final
baseAdress
jal     drawHorizontalLine

                #! top
li      $s1,17  #x coordinate
li      $s2,1   #y coordinate initial
li      $s3,4   #y coordinate final
baseAdress
```

```
jal	drawVerticalLine

		#! bottom
li	$s1,17 #x coordinate
li	$s2,6	#y coordinate initial
li	$s3,6	#y coordinate final
baseAdress
jal	drawVerticalLine

		#! top
li	$s1,19 #x coordinate
li	$s2,1	#y coordinate initial
li	$s3,4	#y coordinate final
baseAdress
jal	drawVerticalLine

		#! bottom
li	$s1,19 #x coordinate
li	$s2,6	#y coordinate initial
li	$s3,6	#y coordinate final
baseAdress
jal	drawVerticalLine

		#! top
li	$s1,21 #x coordinate
li	$s2,1	#y coordinate initial
li	$s3,4	#y coordinate final
baseAdress
jal	drawVerticalLine

		#! bottom
li	$s1,21 #x coordinate
li	$s2,6	#y coordinate initial
li	$s3,6	#y coordinate final
baseAdress
jal	drawVerticalLine
li	$v0,10
syscall

li	$v0,10 #end game
```

syscall

.txt File

Each day you will get a main menu, to navigate to the option press the number in the title (for example family menu is 1)
Your family has a decreasing hunger bar, if it reaches 0 they die.
To feed them you can buy them food from the market.
To check up on them enter the family menu.
To see how much money you have enter the money menu.
To make money you can do a job, if you do good you will be paid well.
Each time you pick a job a day will pass and your family will get hungry.
If they starve you lose.
MAKE 200$ BEFORE THE MONTH ENDS GOOD LUCK!