NOTE: Backing up is your responsibility, but this would be a good time to do it.

III. Races - so the races that come with Suntzu just aren't enough, huh? Adding new races is fairly easy. Some of the races are already implemented and simply need turned on, others will have to be added from scratch. Either way, if you follow all of these instructions, your new race will work out perfectly. We're going to add a new race called Trog to the code. What's a Trog? Who knows, I made it up so we could add it to the code.

A. class.c

1. find: int class_ok_race[NUM_RACES][NUM_CLASSES] = { this section of the code resembles a chart. This chart is where the specific races are allowed or disallowed for the stock classes. The order is important, so we'll add our new race to the bottom of the chart. We also need to decide if our new race will be allowed to be mage, cleric, thief and/or warrior. You will also have to add a comma to the line before. So the last few lines of the code will look like this:

```
/* Warhost      */ { N, N, N, N },
/* Faerie       */ { N, N, N, N },  /* This is where you added the comma */
/* Trog         */ { N, Y, Y, Y }   /* This is the new race, notice that since it's last, it doesn't have a comma */
```

2. find: void do_start(struct char_data *ch) part way down this function, you'll see where each race is assigned the initial languages. We aren't adding any new languages with this document, so of the four already coded, we'll assume that trogs speak some common and that's it. So you will now have to add case RACE_TROG to the code immediately after case RACE_DWARF. It should look like this:

```
case RACE_DWARF:
  SET_SKILL(ch, SKILL_LANG_COMMON, 60);
  SET_SKILL(ch, SKILL_LANG_DWARVEN, 100);
  break;
 case RACE_TROG:
   SET_SKILL(ch, SKILL_LANG_COMMON, 90);   /* all trogs will now start with 90% common */
   break;
```

3. Since we already decided that Trogs can be thieves we now have to set the initial start values for all trogs that are thieves. Since I created Trogs, I have decided they are quite dextrous and would make good thieves. Continue past the newly assigned languages and find the listing for the initial stats for thieves, based on race. This time add a line for each possible thief skill, even if you choose to assign it a value of 0. After case RACE_GNOME add your code so it looks like this:

```
    case RACE_GNOME:
      SET_SKILL(ch, SKILL_SNEAK, 15);
      SET_SKILL(ch, SKILL_HIDE, 10);
      SET_SKILL(ch, SKILL_STEAL, 15);
      SET_SKILL(ch, SKILL_BACKSTAB, 10);
      SET_SKILL(ch, SKILL_PICK_LOCK, 15);
      break;
    case RACE_TROG:
      SET_SKILL(ch, SKILL_SNEAK, 15);
      SET_SKILL(ch, SKILL_HIDE, 15);
      SET_SKILL(ch, SKILL_STEAL, 20);
      SET_SKILL(ch, SKILL_BACKSTAB, 10);
      SET_SKILL(ch, SKILL_PICK_LOCK, 5);
      break;
```

4. Earlier we told the code that all trogs spoke common, but like any other skill it has to be assigned to the character at a certain level. This being a language, it should be assigned to the player at level 1, that way they can communicate when they first log on. Find: void

init_spell_race_levels(void) and then add in your language assignment like the rest. So now it will look like:

```
  /* Dwarf */
  spell_race_level(SKILL_LANG_COMMON, RACE_DWARF, 1);
  spell_race_level(SKILL_LANG_DWARVEN, RACE_DWARF, 1);
  /* Trog */
  spell_race_level(SKILL_LANG_COMMON, RACE_TROG, 1);
```

5. close and save class.c

B. constants.c

1. This first thing to define in constants is item extra bits. The only extra bits having to do with races is the anti-bits. So we have to list anti-trog with the rest of the bits. Find: const char *extra_bits[] = { following is a list of bits that can be assigned to items. It's extremely important to note where in the list you put your new bit. Just to keep things in order, we'll add the new bit after ANTI_DWARF. It should look like this:

```
  "ANTI_GNOME",
  "ANTI_DWARF",
  "ANTI_TROG",   /*  Trog race added by documentation  */
  "UNIQUE",
  "BROKEN",
```

NOTE: by adding a new bit, you are in fact pushing some already existing object flags down in the bitvector - so any existing items that have these bits set will need to be re-edited.

2. The only other thing to be assigned to our new race in this file is the level gains for thieves. At each level up, thieves gain some of their thieving abilities based on their race.
Find: cpp_extern const struct dex_skill_type race_app_skill[NUM_RACES] = after that is a listing of the races and the bonuses they get at each level toward their thief skills. We already decided that trogs make good thieves, so the nubmers will reflect. The new code should look something like this:

```
        { 0, 10, 15,  0,  0, }, /* DWARF     */
        { 5,  0,  0,  5, 10, }, /* TROG      */
};
```

3. close and save constants.c

C. interpreter.c

1. We have to add log on help support. When a new player logs on they may not know what a trog is (you still don't), and the code is already there to be able to look up the helpfile for races, now all we have to do is add our new race to the list.
Find: case CON_RACE_HELP: if you look through that section of code, you will notice that for each race in the code there is a case. We have to add our section to the code with a new case, but before the default is listed. So it should look something like this:

```
      case '5':
        if (race_ok_gender[(int)GET_SEX(d->character)][atoi(arg) - 1])
          show_help(d, "halfelf");
        else
          write_to_output(d, "\r\nThat's not a race.\r\nHelp on Race #: ");
        break;
      case '6':
        if (race_ok_gender[(int)GET_SEX(d->character)][atoi(arg) - 1])
          show_help(d, "trog");
        else
```

```
        write_to_output(d, "\r\nThat's not a race.\r\nHelp on Race #: ");
      break;
```

2. close and save interpreter.c

D. limits.c

1. This next part can get complicated, but if you follow the examples in the code, and what I'm about to give you, it can be done. As characters reach certain ages, their stats and attributes are affected. We now have to define at what years these changes will occur. Find: int graf(int race, int grafage, int p0, int p1, int p2, int p3, int p4, int p5, int p6) What follows is the ages things happen to player's stats. The first number is the lowest age of characters of that race. Not everyone in the mud world ages the same, traditionally elves age the slowest, and humans the quickest. Trogs will age slowly, but not as slow as elves. So the code will look something like this:

```
  case RACE_TROG:  /*  Now our trogs can age at a different rate than humans  */
    AGE1 = 35;
    AGE2 = 62;
    AGE3 = 105;
    AGE4 = 126;
    AGE5 = 157;
    break;
  default:      /*  humans and anyone else not previously listed  */
    AGE1 = 16;
    AGE2 = 31;
    AGE3 = 45;
    AGE4 = 53;
    AGE5 = 60;
    break;
```

2. close and save limits.c

E. races.c

1. The first step is to name the race to the code. Find: const char *race_names[] = { This is a case of just adding to the code that is already there. We'll add trog to the end of the list. So now the list looks like this:

```
 "warhost",
 "faerie",
 "trog",    /* put the new race before the \n but after everything else  */
 "\n"
```

2. The next step is to abbreviate our new race. Find: const char *race_abbrevs[] = { Again, we're just going to add on to the code that's already there. Like this:

```
 "War",
 "Fae",
 "Trg",
 "\n"
```

3. So far we have added the race name to the code and abbreviated it, so we have to add it to the list of races that are allowed for players. Find: const char *pc_race_types[] = { This is yet another case of adding to the existing code:

```
 "Warhost",
 "Faerie",
 "Trog",
 "\n"
```

4. The next step is to define what sexes are allowed with each race. The code is set up with

neuter, male and female. In the stock code, there are no neuters. Trogs, like the rest of the races can be either male or female. Find: int race_ok_gender[NUM_SEX][NUM_RACES] = { What follows is a chart of sorts. If you follow the letters in the remarked out line:
/* H, E, G, D, H, H, D, A, C, D, D, F, G, G, I, O, S, T, H, M, K, M, W, F */ Those are the first letters of the races that are precoded. They are in the same order as the pc_race_types that you defined earlier. Obviously, since we added Trogs to the end of that list, trogs will also show up at the end of this list, so now we have:
/* H, E, G, D, H, H, D, A, C, D, D, F, G, G, I, O, S, T, H, M, K, M, W, F, T */ And we have to add a N or Y (for No, or Yes) to the appropriate lines following. It gives us this:
/* N */ { N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N },
/* M */ { Y, Y, Y, Y, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, Y },
/* F */ { Y, Y, Y, Y, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, N, Y }

5. This file also has the part of the code that displays the race choice for character creation, so we have to add the new race there also. Find: const char *race_display[NUM_RACES] In order to keep the code flowing, we will again add it to the end of the list. You can adjust the numbers here later, as well as the color codes if you choose. For now, let's add it to the end:

```
  "23) Warhost\r\n",
  "24) Faerie\r\n",
  "25) Trog\r\n",
```

6. We told the code what to display on the menu, now we have to tell it what that choice means.
Find: int parse_race(struct char_data *ch, int arg) The cases that follow all directly correspond to the choices assigned above in the selection process. If you change a number here, make sure it is changed above. For example, we told the code to display 25) Trog, so a player who decides to play a trog will select 25 from the menu. So the code here says that 25 is trog. You can change it to be number 6 in both places, as long as they match. Here is what we should have to match the code already in place:

```
  case 23: race = RACE_WARHOST    ; break;
  case 24: race = RACE_FAERIE     ; break;
  case 25: race = RACE_TROG       ; break;  /* new race added by documentation  */
  default: race = RACE_UNDEFINED  ; break;
```

7. Since no two races are built the same, each one has a different set of modifiers for the attributes (str, int, wis, dex, con and cha). The next step is to define them. The default number is 18, so we have to tell the code how to modify the stat either above or below 18. We'll make trogs look like this:

```
    case RACE_DWARF:
      ch->real_abils.con += 1;
      ch->real_abils.cha -= 1;
      break;
    case RACE_TROG:
      ch->real_abils.str += 2;   /* max str 20 */
      ch->real_abils.dex += 1;   /* max dex 19 */
      ch->real_abils.con -= 1;   /* max con 17 */
      ch->real_abils.intel -= 2; /* max int 16 */
      ch->real_abils.wis -= 2;   /* max wis 16 */
      break;
```

8. How tall are trogs? Well, let's define that next.
Find: void set_height_by_race(struct char_data *ch) This part of the code can be tougher to read, since it's not very clear unless you are quite proficient with C. The male and female sections are seperate and seperated again by race. The height is defined in inches, along with a dice roll to modify the height by. The full section of code should look like this:

```
  if (GET_SEX(ch) == SEX_MALE) {
```

```
  if (IS_DWARF(ch))
    GET_HEIGHT(ch) = 43 + dice(1, 10);
  else if (IS_ELF(ch))
    GET_HEIGHT(ch) = 55 + dice(1, 10);
  else if (IS_GNOME(ch))
    GET_HEIGHT(ch) = 38 + dice(1, 6);
  else if (IS_TROG(ch))                    /*  Trog has been added here */
    GET_HEIGHT(ch) = 75 + dice(3, 8);  /*  Trog has been added here */
  else /* if (IS_HUMAN(ch)) */
    GET_HEIGHT(ch) = 60 + dice(2, 10);
 } else /* if (IS_FEMALE(ch)) */ {
  if (IS_DWARF(ch))
    GET_HEIGHT(ch) = 41 + dice(1, 10);
  else if (IS_ELF(ch))
    GET_HEIGHT(ch) = 50 + dice(1, 10);
  else if (IS_GNOME(ch))
    GET_HEIGHT(ch) = 36 + dice(1, 6);
  else if (IS_TROG(ch))                    /*  Trog has been added here */
    GET_HEIGHT(ch) = 75 + dice(3, 8);  /*  Trog has been added here */
  else /* if (IS_HUMAN(ch)) */
    GET_HEIGHT(ch) = 59 + dice(2, 10);
```

9. And now, how heavy are they? Same deal as before.
Find: void set_weight_by_race(struct char_data *ch) And we make the code look like this:

```
 if (GET_SEX(ch) == SEX_MALE) {
  if (IS_DWARF(ch))
    GET_WEIGHT(ch) = 130 + dice(4, 10);
  else if (IS_ELF(ch))
    GET_WEIGHT(ch) = 90 + dice(3, 10);
  else if (IS_GNOME(ch))
    GET_WEIGHT(ch) = 72 + dice(5, 4);
  else if (IS_TROG(ch))                /*  Trog has been added here */
    GET_WEIGHT(ch) = 190 + dice(3, 10);  /*  Trog has been added here */
  else /* if (IS_HUMAN(ch)) */
    GET_WEIGHT(ch) = 140 + dice(6, 10);
 } else /* if (IS_FEMALE(ch)) */ {
  if (IS_DWARF(ch))
    GET_WEIGHT(ch) = 105 + dice(4, 10);
  else if (IS_ELF(ch))
    GET_WEIGHT(ch) = 70 + dice(3, 10);
  else if (IS_GNOME(ch))
    GET_WEIGHT(ch) = 68 + dice(5, 4);
  else if (IS_TROG(ch))                /*  Trog has been added here */
    GET_WEIGHT(ch) = 190 + dice(3, 10);  /*  Trog has been added here */
  else /* if (IS_HUMAN(ch)) */
    GET_WEIGHT(ch) = 100 + dice(6, 10);
```

10. Remember making an item bit earlier for anti_trog? Here's where we do a little something with it.
Find: int invalid_race(struct char_data *ch, struct obj_data *obj) And we'll define the anti_trog just like everything else is defined in this section. It should look like this:

```
 if (OBJ_FLAGGED(obj, ITEM_ANTI_GNOME) && IS_GNOME(ch))
   return (TRUE);
 if (OBJ_FLAGGED(obj, ITEM_ANTI_TROG) && IS_TROG(ch))
   return (TRUE);
 return (FALSE);
```

11. close and save races.c

F. shop.c

1. Who do shops sell to according to the code? Find: const char *trade_letters[] = { Now we can add our new race to this list so that shops will sell to our race. The new list should look like this:

```
"Elf",
"Gnome",
"Trog",
"\n"
```

2. And who will shops not sell to? Find: if ((IS_HUMAN(ch) && NOTRADE_HUMAN(shop_nr)) || And this time we're going to do things differently, in order to make things easy on ourselves. We aren't going to add our new code to the end, but we also aren't going to put it at the beginning. All of the lines in the middle are identical, with the exception of the restrictions. We have to define in this section who the shops won't sell to. Let's add this:

```
if ((IS_HUMAN(ch) && NOTRADE_HUMAN(shop_nr)) ||
    (IS_DWARF(ch) && NOTRADE_DWARF(shop_nr)) ||
    (IS_ELF(ch) && NOTRADE_ELF(shop_nr)) ||
    (IS_TROG(ch) && NOTRADE_TROG(shop_nr)) ||
    (IS_GNOME(ch) && NOTRADE_GNOME(shop_nr))) {
```

3. close and save shop.c

G. utils.c

1. The only thing to change here is the starting age of all characters of the new race. Earlier we defined at what ages changes occur as well as the minimum age, now we have to set the starting age. Find: struct time_info_data *age(struct char_data *ch) Each case RACE is defined with the starting age, so now to add Trog. Earlier we said the minimum age is 35, so we'll make the starting age 38. It should look like this:

```
        case RACE_GNOME:
                player_age.year += 80;
                break;

        case RACE_TROG:
                player_age.year += 38;
                break;

        case RACE_HUMAN:
        default:
                player_age.year += 18;
                break;
```

2. close and save utils.c

H. shop.h

1. This is another place where we must define who shops won't trade with. Find: #define TRADE_NOGNOME (1 << 10) and quite simply add in another bitvector, incrememnting the count by 1. It should look like this:

```
#define TRADE_NOELF     (1 << 9)
#define TRADE_NOGNOME   (1 << 10)
#define TRADE_NOTROG    (1 << 11)
```

2. Further down, we have to tell the code what to do with that new bitvector. Find: #define NOTRADE_GNOME(i) (IS_SET(SHOP_TRADE_WITH((i)),

TRADE_NOGNOME))
and add a new one for TROG. Again, this is what it should look like:
```
#define NOTRADE_ELF(i)          (IS_SET(SHOP_TRADE_WITH((i)), TRADE_NOELF))
#define NOTRADE_GNOME(i)        (IS_SET(SHOP_TRADE_WITH((i)), TRADE_NOGNOME))
#define NOTRADE_TROG(i)         (IS_SET(SHOP_TRADE_WITH((i)), TRADE_NOTROG))
```

3. close and save shop.h

I. structs.h

1. At some point in the code we have to tell the code that there is actually a race to do all the stuff with that we have been doing. This is the place for that.
Find: #define RACE_UNDEFINED -1 What follows is a list of all the races that are coded in to the mud already. We have to add ours to the bottom of the list.
```
#define RACE_WARHOST      22
#define RACE_FAERIE       23
#define RACE_TROG         24
```

2. This next part is easy. The very next line of the code: #define NUM_RACES 24 must be changed. We now have 1 more than we did before, so increment NUM_RACES by 1. (if you are trying to count them to see, don't count the undefined, but do count RACE_HUMAN which is set to 0, not 1 as you might except). So now it should read: #define NUM_RACES 25

3. We also have to tell the code that there is a such thing as anti-trog equipment, since we have already told the code how to process it. Find: #define ITEM_ANTI_GNOME. If you remember earlier in this document I said to make note of where we put some information in relation to others in a list. This is where it becomes important (section B-1, constants.c). We have to put the ITEM_ANTI_TROG bit in the same place. It should come after ITEM_ANTI_GNOME. But that will also throw the following bit count off, so each of those will have to be altered as well. Here is what it should look like:
```
#define ITEM_ANTI_GNOME      25 /* Not usable by gnomes         */
#define ITEM_ANTI_TROG       26 /* Not usable by trogs          */
#define ITEM_UNIQUE_SAVE     27      /* unique object save           */
#define ITEM_BROKEN          28 /* Item is broken hands         */
#define ITEM_UNBREAKABLE     29 /* Item is unbreakable          */
#define ITEM_ANTI_MONK       30 /* Not usable by monks          */
#define ITEM_ANTI_BARBARIAN  31 /* Not usable by barbarians     */
#define ITEM_ANTI_SORCERER   32 /* Not usable by sorcerers      */
```

4. close and save structs.h

J. utils.h

1. Are trogs humanoid? If they are not then they must be added to the humanoid check.
Find: #define IS_HUMANOID(ch) if the added race is NOT humanoid in nature, then add them in to this define just as all the other are. Trogs are humanoid, so we won't add them here.

2. Lastly, we have to assign the character to the chosen race.
Find: #define IS_MINDFLAYER(ch) (GET_RACE(ch) == RACE_MINDFLAYER)
and add in the new race, just like all the others. It should look like this:
```
#define IS_KOBOLD(ch)          (GET_RACE(ch) == RACE_KOBOLD)
#define IS_MINDFLAYER(ch)      (GET_RACE(ch) == RACE_MINDFLAYER)
#define IS_TROG(ch)            (GET_RACE(ch) == RACE_TROG)
#define IS_UNDEAD(ch)          (IS_AFFECTED(ch, AFF_UNDEAD))
```

3. close and save utils.h

K. compile

1. considering the amount of change that has just taken place, I suggest that you make clean before you make.

2. Boot your mud and make a new character with the new race. I also suggest that you make a few items, and shops to test your new race out fully.