TAWS7. The 7th TAWS. This entry in the TAWS series, this one is fairly easy to jump in and play. TAWS7 is the first in the TAWS series to use the Curses library and is also the first to be action based.

TAWS7 is an arena shooter, a bit like the classic arcade game robotron if robotron were text based. The action quickly becomes frantic as the longer you play the more enemies appear. While you can fire in 8 directions they can fire at any angle and will constantly on aim. Your best strategy is to keep moving. Fortunately they only take 1 hit to kill, have a limited range, and are not immune to the shots of their team mates.

Move your character with the arrow keys and use the number pad to shoot in 8 directions.

TAWS7 is written by "Entar".

```c
/* game.c listing begins: */

// gamecode
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "curses.h"
#ifdef WIN32
//#include <conio.h>
#else
#include <sys/time.h>
#include <termios.h>
#include <unistd.h>
#endif
#include <math.h>

#define min(a,b) (a < b ? a : b)

typedef enum {t_bullet, t_player, t_enemy, t_rock} type_t;

typedef struct
{
  int active;
  float origin[2];
  float velocity[2];
  type_t type;
  chtype display;
  int health;
  unsigned int nextshot, nextmove;
} entity_t;

#define MAX_ENTITIES 64
entity_t entities[MAX_ENTITIES];
entity_t *player;
int angle[2];
int paused=0;

extern int width, height;

unsigned int timecounter, oldtime=0, timediff;

int max_enemies=1;
int killed=0;
int score=0;

void Game_Render (void)
{
    char message[128];
  int x, y;

    clear();
  // draw boundaries
  for (x=0; x < width; x++)
    mvaddch(0, x, '#'|COLOR_PAIR(COLOR_RED));
  for (y=1; y < height; y++)
  {
        mvaddch(y, 0, '#'|COLOR_PAIR(COLOR_RED));
        mvaddch(y, width-1, '#'|COLOR_PAIR(COLOR_RED));
  }
  for (x=0; x < width; x++)
      mvaddch(height-1, x, '#'|COLOR_PAIR(COLOR_RED));
```

```
/* Listing continued from previous page */
    for (y=1; y < height-1; y++)
        for (x=1; x < width-1; ++x)
            mvaddch(y, x, ' '|COLOR_PAIR(COLOR_RED));

  // put the entities on the screen
  for (x=1; x < MAX_ENTITIES; x++)
  {
    if (!entities[x].active)
      continue;
    mvaddch((int)(entities[x].origin[1]+0.5f), (int)(entities[x].origin[0]+0.5f),
entities[x].display);
  }
  // put the player on last, so he's always on top
  mvaddch((int)(player->origin[1]+0.5f), (int)(player->origin[0]+0.5f), player-
>display);

  if (paused)
      mvprintw(height/2, width/2-3, "PAUSED");

  if (player->health > 0)
      sprintf(message, "HP: %.2i   Score: %.4i    X=Exit"
      "    Numpad to control character.\n", player->health, score);
  else
    sprintf(message, "YOU DEAD Score: %i    X=Exit\n", score);
  mvprintw(height, 0, message);
  refresh();
}

entity_t *FindFreeEntity (void)
{
  int i;
  for (i=0; i < MAX_ENTITIES; i++)
  {
    if (entities[i].active < 1)
      return &entities[i];
  }
  return NULL;
}

int update=1;

void Game_SpawnEnemy (void)
{
  // pick a random spot on the edges of the map
  // and spawn an enemy in a free entity slot.
  entity_t *ent=FindFreeEntity();
  int side=rand()%4;
  if (!ent) return;
  ent->active = 1;
  ent->type = t_enemy;
  ent->display = 'i'|COLOR_PAIR(COLOR_YELLOW)|A_BOLD;
  ent->health = 2;
  ent->nextshot = timecounter + CLOCKS_PER_SEC*3/2-(rand()%(CLOCKS_PER_SEC/10));
  ent->nextmove = timecounter + CLOCKS_PER_SEC;
  if (side < 2) // left or right side
  {
    if (side == 0)
      ent->origin[0] = 1;
    else
      ent->origin[0] = width-2;
    ent->origin[1] = (rand()%(height-2))+1;
  }
  else // top or bottom
```
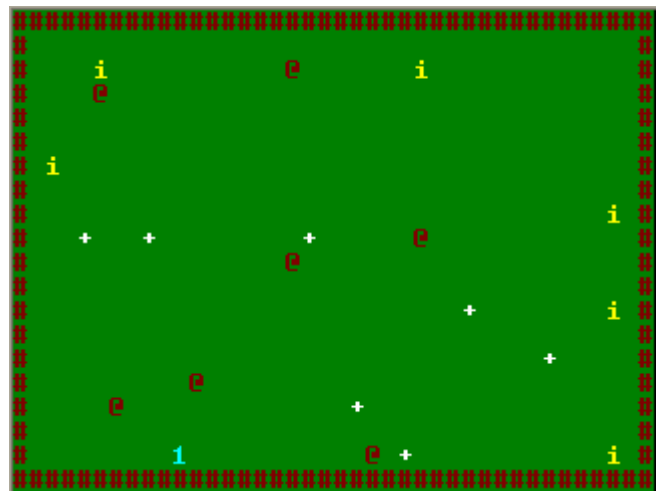
/* Listing continued from previous page */

```c
  {
    if (side == 2)
      ent->origin[1] = 1;
    else
      ent->origin[1] = height-2;
    ent->origin[0] = (rand()%(width-2))+1;
  }
  update = 1;
}

void Game_FireBullet (float org_x, float org_y, float vel_x, float vel_y)
{
  entity_t *ent;
  ent = FindFreeEntity();
  if (ent)
  {
    ent->active = 1;
    ent->type = t_bullet;
    ent->display = '+'|COLOR_PAIR(COLOR_WHITE)|A_BOLD;
    ent->origin[0] = org_x;
    ent->origin[1] = org_y;
    ent->velocity[0] = vel_x;
    ent->velocity[1] = vel_y;
    ent->nextshot = timecounter+CLOCKS_PER_SEC*5/2;
    update = 1;
  }
}

entity_t *Game_CheckCollision(int x, int y, entity_t *ignore)
{
  int i;
  for (i=0; i < MAX_ENTITIES; i++)
  {
      if (ignore == &entities[i])
            continue;
    if ((int)(entities[i].origin[0]+0.5f) == x && (int)(entities[i].origin[1]
+0.5f) == y)
      return &entities[i];
  }
  return NULL;
}

void Game_DoCollision (entity_t *a, entity_t *b)
{
    if (a->type == t_bullet)
    {
    // hit something
    if (b->type != t_bullet)
    {
      if (b->type != t_rock)
      {
        b->health -= 2;
        if (b->health < 1)
        {
          if (player == b)
            b->display = 'X'|COLOR_PAIR(COLOR_RED)|A_BOLD;
          else
            b->active = 0;
          if (player->health > 0 && b->type == t_enemy)
          {
            killed++;
            if (killed > (max_enemies*3))
              max_enemies++;
```

/* Listing continued on next page…*/

/* Listing continued from previous page */

```c
            score += 25;
          }
        }
      }
          a->active = 0;
      }
    }
  }
}

void Game_Update(void)
{
  int i;
    float old[2];
  entity_t *ent;
  int enemies=0;
  if (timediff < 1)
    return;
  for (i=0; i < MAX_ENTITIES; i++)
  {
    if(entities[i].active < 1)
      continue;
    if(entities[i].type == t_bullet)
    {
      if (timecounter >= entities[i].nextshot)
      {
        entities[i].active = 0;
        continue;
      }

      // check for collision
      ent = Game_CheckCollision((int)(entities[i].origin[0]+0.5f),
            (int)(entities[i].origin[1]+0.5f), &entities[i]);
      if (ent)
      {
        Game_DoCollision(&entities[i], ent);
        if (!entities[i].active)
          continue;
      }

      old[0] = (int)(entities[i].origin[0]+0.5f);
      old[1] = (int)(entities[i].origin[1]+0.5f);
      entities[i].origin[0]+=entities[i].velocity[0]
        *((float)timediff/(float)CLOCKS_PER_SEC);
      entities[i].origin[1]+=entities[i].velocity[1]
        *((float)timediff/(float)CLOCKS_PER_SEC);
      if ((int)(entities[i].origin[0]+0.5f) != old[0]
        || (int)(entities[i].origin[1]+0.5f) != old[1])
        update = 1;
      else
        continue;

      if ((int)(entities[i].origin[0]+0.5f) < 1 ||
        (int)(entities[i].origin[0]+0.5f) > width-2 ||
        (int)(entities[i].origin[1]+0.5f) < 1 ||
        (int)(entities[i].origin[1]+0.5f) > height-2)
        entities[i].active = 0; // kill if it goes off screen
      // check for collision
      ent = Game_CheckCollision((int)(entities[i].origin[0]+0.5f), (int)(entities
[i].origin[1]+0.5f), &entities[i]);
      if (ent)
            Game_DoCollision(&entities[i], ent);
    }
    else if (entities[i].type == t_enemy)
```

/* Listing continued from previous page */

```
    {
      enemies++;
      if (timecounter >= entities[i].nextmove)
      {
        if (entities[i].origin[0] < 2)
          entities[i].origin[0]++;
        else if (entities[i].origin[0] > width-3)
          entities[i].origin[0]--;
        else if (entities[i].origin[1] < 2)
          entities[i].origin[1]++;
        else if (entities[i].origin[1] > height-3)
          entities[i].origin[1]--;
        else
        {
          entities[i].origin[0]+=1-rand()%3;
          entities[i].origin[1]+=1-rand()%3;
        }
        entities[i].nextmove = timecounter + CLOCKS_PER_SEC;
        update = 1;
      }
      if (timecounter >= entities[i].nextshot && player->health > 0)
      {
        float org[2]={entities[i].origin[0], entities[i].origin[1]};
        float vel_x = player->origin[0]-entities[i].origin[0], vel_y=player-
>origin[1]-entities[i].origin[1];
        float size = sqrt(vel_x*vel_x+vel_y*vel_y);
        if (vel_x < 0)
          org[0] = entities[i].origin[0]-1;
        if (vel_x > 0)
          org[0] = entities[i].origin[0]+1;
        if (vel_y < 0)
          org[1] = entities[i].origin[1]-1;
        if (vel_y > 0)
          org[1] = entities[i].origin[1]+1;
        vel_x *= sqrt((height/3)*(height/3)*2)/size;
        vel_y *= sqrt((height/3)*(height/3)*2)/size;
        Game_FireBullet(org[0], org[1], vel_x, vel_y);
        entities[i].nextshot = timecounter + CLOCKS_PER_SEC*3/2
          - (rand()%(CLOCKS_PER_SEC/10));
        update = 1;
      }
    }
  }
  if (enemies < max_enemies && timecounter%2000 < 250)  {
    Game_SpawnEnemy();
  }
}

void Game_Loop (void)
{
  int i;
  int in;
  unsigned long totalDiff;

  // initialize the game.
  srand(time(NULL));
  // entities[0] is the player.
  player = &entities[0];
  player->active = 1;
  player->type = t_player;
  player->origin[0] = width/2;
  player->origin[1] = height/2;
  player->health = 20;
```

/* Listing continued from previous page */
```c
  player->display = '1'|COLOR_PAIR(COLOR_CYAN)|A_BOLD;

  // landscaping
  for (i=1; i <= 7; i++)
  {
    entities[i].active = 1;
    entities[i].type = t_rock;
    entities[i].origin[0] = (rand()%(width-2))+1;
    entities[i].origin[1] = (rand()%(height-2))+1;
    entities[i].health = 0;
    entities[i].display = '@'|COLOR_PAIR(COLOR_RED);
  }

  while (1)
  {
    timecounter = clock();
    totalDiff = timecounter - oldtime;
    oldtime = timecounter;

    if (!paused)
    {
            while (totalDiff > 0)
            {
                timediff = min(10, totalDiff);
        Game_Update();
        totalDiff -= timediff;
            }
    }
    in = getch();
    if (in != ERR)
    {
      if (in == 'p')
      {
        paused=!paused;
        update=1;
      }
      else if (in == 'x')
        break; // game over
      else if (paused)
      {
      }
      // movement
      else if (in == KEY_DOWN || in == 's')
      {
        if (player->origin[1] < height-2 && player->health > 0)
        {
          player->origin[1]+=1.0f;
          angle[0] = 0;
          angle[1] = 1;
          update = 1;
        }
      }
      else if (in == KEY_UP || in == 'w')
      {
        if (player->origin[1] > 1 && player->health > 0)
        {
          player->origin[1]-=1.0f;
          angle[0] = 0;
          angle[1] = -1;
          update = 1;
        }
      }
      else if (in == KEY_LEFT || in == 'a')
```

/* Listing continued from previous page */

```c
      {
        if (player->origin[0] > 1 && player->health > 0)
        {
          player->origin[0]-=1.0f;
          angle[0] = -1;
          angle[1] = 0;
          update = 1;
        }
      }
      else if (in == KEY_RIGHT || in == 'd')
      {
        if (player->origin[0] < width-2 && player->health > 0)
        {
          player->origin[0]+=1.0f;
          angle[0] = 1;
          angle[1] = 0;
          update = 1;
        }
      }
      else if (timecounter >= player->nextshot && player->health > 0)
      {
        if (in == '1')
          Game_FireBullet((int)player->origin[0]-1.0f,
            (int)player->origin[1]+1.0f, -height/2, height/2);
        else if (in == '2')
          Game_FireBullet((int)player->origin[0],
            (int)player->origin[1]+1.0f, 0, height/2);
        else if (in == '3')
          Game_FireBullet((int)player->origin[0]+1.0f,
            (int)player->origin[1]+1.0f, height/2, height/2);
        else if (in == '4')
          Game_FireBullet((int)player->origin[0]-1.0f,
            (int)player->origin[1], -height/2, 0);
        else if (in == '6')
          Game_FireBullet((int)player->origin[0]+1.0f,
            (int)player->origin[1], height/2, 0);
        else if (in == '7')
          Game_FireBullet((int)player->origin[0]-1.0f,
            (int)player->origin[1]-1.0f, -height/2, -height/2);
        else if (in == '8')
          Game_FireBullet((int)player->origin[0],
            (int)player->origin[1]-1.0f, 0, -height/2);
        else if (in == '9')
          Game_FireBullet((int)player->origin[0]+1.0f,
            (int)player->origin[1]-1.0f, height/2, -height/2);
        else if (in == '0')
        {
          Game_FireBullet((int)player->origin[0]+angle[0],
            (int)player->origin[1]+angle[1],
            angle[0]*height/2, angle[1]*height/2);
                }
        player->nextshot = timecounter + CLOCKS_PER_SEC/4;
      }
      flushinp();
    }
    if (update)
    {
      Game_Render();
      update = 0;
    }
  }
}
```

/* End of game.c but there's more on the next page…*/

```c
/* main.c listing begins... */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include "curses.h"

int width, height;

void ResizeScreen(int w)
{
  width = w;
  height = w/2;
}

extern void Game_Loop(void);

int main (int args, char *argc[])
{
  int i, pdc=0;
  char input[64];

  ResizeScreen(40); // default
  for (i=0; i < width; i++)
    printf("_");
  printf("\n");
  for (i=0; i < height; i++)
    printf("|\n");
  printf("Welcome to TAWS Volume 5!\n");
  printf("Resizing the window to the size of the window is recommended.\n");
  while (1)
  {
    printf("Is the screen size above acceptable? ");
    fgets(input, 63, stdin);
    if (input[0] == 'y' || input[0] == 'Y')
    {
        pdc = 1;
        initscr();
        cbreak();
        noecho();
        keypad(stdscr, 1);
        nodelay(stdscr, TRUE);
        start_color();
        for (i=0; i <= 8; ++i)
            init_pair(i, i, COLOR_GREEN);
      Game_Loop();
      break;
    }
    else if (input[0] == 'x' || input[0] == 'X' || !strcmp(input, "exit"))
      break;
    else
    {
      memset(input, 0, sizeof(input));
      printf("Enter width in characters:");
      fgets(input, 63, stdin);
      ResizeScreen(atoi(input));
      memset(input, 0, sizeof(input));
      for (i=0; i < width; i++)
        printf("_");
```

```
/* Listing continued from previous page */

      printf("\n");
      for (i=0; i < height; i++)
        printf("|\n");
    }
  }
  if (pdc)
      endwin();
  printf("Thanks for playing!\n");
  return 0;
}
```