Twisty

Twisty is a minimalist interactive fiction adventure game somewhat like Zork. You are in a maze of twisty little passages, all alike.

This program is a rather nostalgic little labyrinth game. Start out by typing "get lamp", then use the direction words "left", "right", "forward", and "back" to explore the caves. Note that these words can refer to different passages depending on which way you're facing. "back" always takes you back the way you came. Type "quit" to quit.

The maze is randomly generated each time you play. The algorithm ensures that you start at least 6 hops away from the exit. All passages are two-way. (There is a trick, well-known among adventure game fans, to keeping one's bearings in games like this. Unless you know the trick, it can be very difficult to find your way around. Try it out first; I'll give away the secret at the bottom of this document.)

The goal is to collect 8 treasures (plus the lamp) and escape into the sunlight - or moonlight, as the case may be. After a few minutes, the lamp runs out of oil, and things become perilous. If you manage to escape, you score 50 points for your hide, 50 points per item, and bonus points for speed.

Twisty was written by Jason Orendorff.

TWISTY.C	You will need: a C/C++ complier .

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#define NITEMS 9
#define LAMP 0
#define CARRIED 0  /* used in itemLoc to indicate carried items */

const char *itemNames[NITEMS] = {
      "lamp",
      "diadem",
      "harp",
      "kris",
      "chime",
      "ruby",
      "scepter",
      "talisman",
      "zither"
};

#define NDIRS 4
#define FORWARD_DIR 1
#define OPPOSITE_DIR(i)  ((i) ^ 2)
#define OPPOSITE_WALL(w)  ((w) ^ 2)

const char *dirNames[NDIRS] = {
      "left",
      "forward",
      "right",
      "back"
};

/* The cave locations are numbered [1...NCAVES].  If you change
   NCAVES, you must also change the addTangle() calls in
   drawMap().  The other "locations" are special end-of-game
   indicators. */
#define NCAVES 15

#define START_LOC  1
#define EXIT_THRESHOLD_LOC  NCAVES
```

Listing continued on next page...

Listing Continued from previous page….

```
#define WIN_LOC  0
#define QUIT_LOC (NCAVES + 1)
#define EATEN_LOC (NCAVES + 2)

/* The map.  The value at map[loc * NDIRS + dir] is 0 if there is no
   tunnel in that wall.  Otherwise it is the wall number of the other
   end of the tunnel, (destination * NDIRS + destinationWallDir).
   drawMap counts on this being initially zeroed out. */
int map[(1 + NCAVES) * NDIRS];

int loc;  /* player location */
int heading;  /* player heading */
int itemLoc[NITEMS];  /* item locations */

/* Time when the lamp will run out; or 0 if it already has. */
time_t lampOil;

int needDesc;  /* describe the surroundings before the next prompt */

char verb;  /* First letter of the current command.  The game only
    recognizes the first letter of each input word. */

char noun;  /* First letter of the second word of the current command,
    or '\0' if the command is only one word long. */

/* random whole number from 1 to i */
int rollDie(int i)
{
      return rand() % i + 1;
}

/* === The map === */
int lastTunnel;

/* Convert a relative direction (d is 0 for left, 1 for forward,
   and so on; see dirNames) to a cardinal direction (an integer
   in the half-open range [0..NDIRS], usable as an index into a
   room's map[] entries). */
int realDir(int d)
{
      return (d + heading + NDIRS - FORWARD_DIR) % NDIRS;
}

/* Make a tunnel connecting walls i and j, if neither is already
   connected to someplace else. */
int addPath(int i, int j)
{
      lastTunnel = j;
      if (map[i] == 0 && map[j] == 0) {
            map[i] = j;
            map[j] = i;
            return 1;
      }
      return 0;
}

/* Add some random passages to the maze.  start and stop are wall
   numbers; the new passages connect walls in the half-open range
   [start, stop). */
void addTangle(int start, int stop, int tries)
```

Listing continued on next page...

```
{
      /* I think the winning submission has a bug here.  At least
           the behavior seems mystifying to me.  The intended
         behavior, I think, was like this. */

      int i, j;

      for (i = start; i < stop - 1; i++)
            for (j = 0; j < tries; j++)
                  if (addPath(i, i + rollDie(stop - 1 - i)))
                        break;
}

void drawMap()
{
      int i;

      /* First, make a path from START_LOC to EXIT_THRESHOLD_LOC
           that visits every room.  With this, at worst there's one
           really long solution. */
      for (i = START_LOC; i < EXIT_THRESHOLD_LOC; i++)
            addPath(i * NDIRS, (i + 1) * NDIRS + rollDie(NDIRS - 1));

      /* Add the exit.  The exit is always in the exit threshold
           cave, and always directly opposite the only passage into
           that room, so the command to leave the maze is always
           "forward". (Otherwise the message "sunlight streams in
         ahead!" would be nonsensical.) */
      map[OPPOSITE_WALL(lastTunnel)] = WIN_LOC * NDIRS + 1;

      /* Add more passages to the maze. */
      addTangle(4, 16, 2);  /* first 3 rooms */
      addTangle(16, 36, 3);  /* next 5 rooms */
      addTangle(36, 60, 6);  /* last 6 rooms.
            60 == EXIT_THRESHOLD_LOC * NDIRS; this stops just short of
            adding any more passages into the exit-threshold cave. */
}

/* === Input === */

void readCommand()
{
      char L[99];
      char *p;

      /* The obfuscated version has something more like
           `if (*fgets(L, 98, stdin) == '\0')` which can crash. */
      if (fgets(L, 99, stdin) == NULL || L[0] == '\0')
            exit(0);

      p = L;
      while (*p != '\0' && *p <= ' ')
            p++;
      verb = *p;
      while (*p != '\0' && *p != ' ')
            p++;
      while (*p != '\0' && *p <= ' ')
            p++;
      noun = *p;
}
```

Listing continued on next page...

Listing Continued from previous page….

```c
/* === inventory and get/drop === */

/* Darkness affects several things in the game.  It's dark if the lamp
   has run out or is not present.  Bug:  It should never be dark in
   EXIT_THRESHOLD_LOC. */
int isDark()
{
	if (loc == EXIT_THRESHOLD_LOC)
		return 0;
	if (lampOil == 0)
		return 1;
	return itemLoc[LAMP] != loc && itemLoc[LAMP] != CARRIED;
}

int inv()
{
	int haveAny = 0;
	int i;

	printf("you have\n");
	for (i = 0; i < NITEMS; i++) {
		if (itemLoc[i] == CARRIED) {
			printf("   a %s\n", itemNames[i]);
			haveAny = 1;
		}
	}
	if (!haveAny)
		printf("   nothing\n");
	return haveAny;
}

/* Get or drop a specific item. "name" is the first character of the
   item name.  If there is no item with that name, you get a generic
   "you do not see/have that" error message. */
int specificGetOrDrop(int isGet, char name)
{
	int i;

	if (!(isGet && isDark())) {
		for (i = 0; i < NITEMS; i++) {
			if (name == itemNames[i][0]
					&& itemLoc[i] == (isGet ? loc : CARRIED)) {
				printf("done\n");
				return itemLoc[i] = (isGet ? CARRIED : loc);
			}
		}
	}
	printf(" you do not %s that\n", isGet ? "see" : "have");
	return 0;
}

/* Get/drop without a specific item. */
void vagueGetOrDrop(int isGet)
{
	printf("%s what? ", isGet ? "get" : "drop");
	readCommand();
	specificGetOrDrop(isGet, verb);
}
```

Listing continued on next page...

Listing Continued from previous page….

```
void getOrDrop(int isGet)
{
      if (noun != '\0')
            specificGetOrDrop(isGet, noun);
      else if (isGet || inv())
            vagueGetOrDrop(isGet);
}

/* === Movement === */
/* Try moving; i is 0 if the way is blocked, otherwise the number of
   the destination wall. */
void tryMovingTo(int i)
{
      if (i != 0) {
            printf("you climb...\n");
            loc = i / NDIRS;  /* The original adds "& 63" here,
               but it has no effect. */
            heading = OPPOSITE_DIR(i % NDIRS);
            needDesc = 1;
      } else {
            printf("eh?\n");
      }

      if (isDark() && rollDie(6) == 6) {
            printf("chomp crunch grind slurp\n"
                   "oops.  you fed the grue\n");
            loc = EATEN_LOC;
      }
}

/* === Cave description === */
/* What kind of light do we have at this time of day? */
const char *light()
{
      time_t x = time(0);
      int j = localtime(&x)->tm_hour;

      if (j < 6 || j > 19)
            return "moonlight";
      else
            return "sunlight";
}

/* Print a description of the player's current location. */
void lookAround()
{
      int i;
      int seen;

      if (isDark()) {
            printf("it is pitch black here\n");
            return;
      }

      printf("you are in a maze of twisty little passages\ncaves lead: ");

      for (i = 0; i < NDIRS; i++)
            if (map[loc * NDIRS + realDir(i)])
```

Listing continued on next page...

```
                  printf(" %s", dirNames[i]);
      putchar('\n');

      seen = 0;
      for (i = 0; i < NITEMS; i++) {
            if (itemLoc[i] == loc) {
                  if (seen++ == 0)
                        printf("you see\n");
                  printf("   a %s\n", itemNames[i]);
            }
      }

      if (loc == EXIT_THRESHOLD_LOC)
            printf("%s streams in ahead!\n", light());
}

/* === Main === */
int isDirectionVerb(char verb, int *pDir)
{
      int i;

      for (i = 0; i < NDIRS; i++) {
            if (verb == dirNames[i][0]) {
                  *pDir = i;
                  return 1;
            }
      }
      return 0;
}

int main()
{
      int score;
      int i;
      int d;

      lampOil = time(0) + 5 * 60;
      srand(lampOil);

      drawMap();

      /* Scatter the treasures, but put the player and the lamp in
           the starting cave. */
      for (i = 0; i < NITEMS; i++)
            itemLoc[i] = rollDie(NCAVES);
      loc = itemLoc[LAMP] = START_LOC;
      heading = 1;
      needDesc = 1;

      /* "while the player is in the caves..." */
      while (loc >= START_LOC && loc < START_LOC + NCAVES) {
            /* check lamp lifetime */
            if (lampOil != 0 && lampOil < time(0)) {
                  if (!isDark()) {
                        printf("your lamp dies!\n");
                        needDesc = 1;
                  }
                  lampOil = 0;
            }
```

Listing continued on next page...

TWISTY.C	Listing Continued from previous page….

```
            /* describe surroundings if necessary */
            if (needDesc) {
                  lookAround();
                  needDesc = 0;
            }

            /* get command */
            printf("> ");
            readCommand();

            /* interpret command */
            if (verb == 'i')
                  inv();
            else if (verb == 'q')
                  loc = QUIT_LOC;
            else if (verb == 'g' || verb == 'd')
                  getOrDrop(verb == 'g');
            else if (isDirectionVerb(verb, &d))
                  tryMovingTo(map[loc * NDIRS + realDir(d)]);
            else
                  printf("eh?\n");
      }

      if (loc == WIN_LOC) {
            printf("into bright %s!\n", light());

            score = 50;
            for (i = 0; i < NITEMS; i++)
                  if (itemLoc[i] == CARRIED)
                        score += 50;
            if (lampOil != 0)
                  score += lampOil - time(0);

            printf("score: %d\n", score);
      }
      return 0;
}
```

GAME STRATEGY SPOILER: The trick, of course, is to drop items in empty rooms to mark them. That way you usually know if you're in a room you've seen before. It greatly aids mapping.