```cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
#include <bitset>
#include <math.h>
using namespace std;

// Author: Glenn Storm
// Text adventure game Zombie Cruise
//
// Object: survive on a cruise ship with a collection of people faced with a mysterious virus that has been released
//
// Locations: [8] Bridge, Fore Deck, Aft Deck, Ballroom, Lounge, Kitchen, Store Room, Engine Room
// Items: [8] Flare Gun, Fire Extinguisher, Alcohol Bottle, Diving Knife, Spear Gun, Wrench, Cleaver, Fuel Can
// People: [8] Captain, 1st Mate, Chef, Bartender, Mr. Rich, Mrs. Rich, Prof. Smart, Ms. Sass
//
// (see Revisions below)
// Game data: Time to Release (0-7 turns), Is Released, Radio Used, S.O.S. Called, Rescue Arrived, Game Over
// Time / Scoring data: Time To Rescue (since call for help, 0-31 turns), Score (0-31)
// Player data: Current Location, Left Equip, Right Equip, Health
// Location data: Location I.D., Exit A, Exit B, Exit C
// Item data: Item I.D., Current Location, Is Equipped, Is Used, Damage
// Character data: People I.D., Current Location, Is Infected, Is Zombie, Target Location
//
// Working Memory Budget : 32 bytes (256 bits)
// Game: 1 byte
// Time / Score: 1 byte
// Player 1 byte
// Locations: 8 bytes
// Items: 13 bytes (1 of each item, plus extra one of 5 items)
// Characters: 8 bytes
//
// -- REVISIONS --
//
// [oops! 0-7 uses three bits, not two]
// Game data : (ok)
// Time / Score data : (ok)
// Player data : only one equip item? (not ok)
//    [currentLocation(3)-equipItem(3)-healthRemaining(2)]
//    borrow 1 byte from items, use for 2 equip plus inventory? (ok)
// [currentLocation(3)-equipItemA(3)-equipItemB(3)-inventoryHeld(3)-healthRemaining(2)-extra?(2)]
//    use extra 2 bits for your appearance when 'look at me' or 'look at mirror'? ("spiffy", "stressed", "worn out", "like hell") (not ok)
//    (refactor...)
//    use 4 bit number to index items equipped (0-11), use value 15 to equal 'none equipped' (ok)
//    use 3 bit number to track how many items carried at once (7 max) (ok)
// [currentLocation(3)-healthRemaining(2)-equipItemA(4)-equipItemB(4)-inventoryCarried(3)]
// Location data : no location i.d. needed? only two exits per location? (not ok)
//    [exitA(3)-exitB(3)-...]
//    borrow 1 byte from items, use 1 bit for each of 8 rooms? (ok - must assemble 3 bits from 2 different bytes, get/set num)
//    [exitA(3)-exitB(3)-exitC(2+1 bit from aux byte)]
// Item data : no damage rating needed? (ok) 11 bytes, 1 of each plus extra 3 items total (ok)
//    [itemID(3)-currentLocation(3)-isHeld(1)-isUsed(1)]
// Character data : no char i.d. needed? (ok)
//    [currentLocation(3)-targetLocation(3)-isInfected(1)-isZombie(1)]
//
// [wait. data (like location data) should only include that which cannot be static: exits are static]
// Location data: no exit data needed? (ok)
//    [fireStatus(2), fireDuration(3), locationDamage(2), lightsOn(1)]
// (this means no splitting location data, and +1 item available = total of 12 items, 12 bytes of item data)
//
// Revised Working Memory Budget : 32 bytes (256 bits)
// Game: 1 byte
// Time / Score: 1 byte
// Player 2 bytes
// Locations: 8 bytes
// Items: 12 bytes (1 of each item, plus extra one of 4 items)
// Characters: 8 bytes
//
// [wait. no way to kill characters/zombies with no health. need wait during infection plus zombie plus death]
// REFACTOR NEEDED
// [need target location? can use a single 'wait here' bit to slow zombies and use other two bits for health?]
// Character data: nix target room (3 bits) as unnecessary, use single 'wait 1 turn' bit, use other two for health 0-3 (ok)
// [currentLocation(3)-waitHere(1)-health(2)-isInfected(1)-isZombie(1)]
//
// [wait. location timer needs 4 bits, not 3]
// [lose location damage bits. transfer one bit to fire timer (4), call other bit 'flicker' to effect light on] (ok)
// [lightsOn(1)-flickerLights(1)-fireState(2)-fireTimer(4)]
//
// [wait. initial location description > flickering lights]
// [lose light flicker bit and use it to indicate if location has been visited]
// [lightsOn(1)-visited(1)-fireState(2)-fireTimer(4)]
//
// -- END REVISIONS --
//
// Boat Map:
//
//       +-----+
//       |  0  |
// +-----+-----+-----+
// |  2 | 4 | 3 | 1 /
//  \---+---+---+--/
//  x\  7 | 5 | 6 /
//    +---+---+--+
//
// 0= Bridge, 1= Fore Deck, 2= Aft Deck, 3= Ballroom, 4= Lounge, 5= Kitchen, 6= Store Room, 7= Engine Room
// Exits: 0=1-2-4, 1=0-3-6, 2=0-4-7, 3=1-4-5, 4=0-2-3, 5=3-6-7, 6=1-5-7, 7=2-5-6
//
// Player Commands:
// Quit, Help, Wait, Look, Look At [item/character], Take [item], Drop [item], Inventory,
// Equip [item], Use [item/location feature], Attack [character] With [item],
// Move To [location], Talk To [character]
//
// Player can have up to 2 items at once, Player can move, Player can use items, Items can be moved or used, Characters can move
// Characters can be infected, Infected characters turn zombie, Zombies can move, Zombies can attack player or characters
// Characters infected previous turn 'rest' at location one turn, Infected characters turn zombie next turn
// Characters not infected move from locations with zombies, characters stay in
```

```
locations with other characters or player
// Zombies target location and take another turn to move, Zombies target exits to follow characters or player
// Attacking Zombies with items does damage, Attacking Characters with items kills them (no points)
// Items Spear Gun, Flare Gun, Fuel Can, Alcohol Bottle are one use only, destroyed after use, auto-dropped at location
//
// Locations have a light switch, and if player uses, lights toggle on or off, unless they are damaged
// While lights are off, room descriptions are unavailable with 'Look' command, only sounds and smells are provided
// Flickering lights provide a more brief and vague description of the room and its contents (characters/zombies)
// Item descriptions are only randomly available (50% chance) while lights are off or flickering
// Locations on fire always have light. Flare gun used provides light that turn
//
// Fuel can and alcohol bottle used makes location flammable, Flare gun used in flammable location makes fire (while items exist)
// Location fire status (0-3) can be none, flammable, onFire, burnt. Fire damage (0-7) occurs in location over time (0-15 turns)
// If fire in location has damaged more than 2, can spread to flammable adjacent locations
// If damaged more than 4, can spread to non-flammable locations, but at 7, fire goes out on its own
// If location damaged more than 2, lights flicker if on, if more than 4, lights cannot turn on
// Characters move from locations with fire (will continually exit if entire ship on fire)
// If Bridge is on fire, player cannot use radio to call S.O.S. without taking 4 damage and dying
// Fire can spread to flammable locations connected with exit (chance per turn)
// Using Fire Extinguisher in location removes items that make fire
//
// Character health = 1
// Zombie health = 3
// Player health = 3
// If Player loses all health, - Player loses -
//
// Item Damage: Flare Gun=3, Fire Extinguisher=2, Alcohol Bottle=1, Diving Knife=2, Spear Gun=3, Wrench=2, Cleaver=2, Fuel Can=1
// = ITEM DAMAGE TABLE =
//  Spear Gun, Flare Gun        : 3
//  Fire Extinguisher, Wrench,
//    Cleaver, Diving Knife     : 2
//  Alcohol Bottle, Fuel Can    : 1
// Fire Damage: 2 per turn stayed at location on fire (characters not infected will always survive fire, by exiting)
//
// Player at Bridge can use radio (takes 2 turns)
// If Player uses radio at Bridge, S.O.S. called and timer started
// If Player survives time to rescue, coast guard arrives
// If <4 Zombies left, rescue successful, - Player wins -
// If >=4 Zombies left, coast guard overcome, - Player loses -
//
// Scoring:
// Character survives to successful rescue: 2 points
// Kill a Zombie: 3 points
// Player survives until rescue: 2 points
// Rescue successful (<4 Zombies at rescue): 5 points
// HIGHEST SCORE POSSIBLE : 24+2+5 = 31
// Score Ranks: <11 "Coward", 11-20 "Survivor", 21-30 "Hero", 31 "Zombie Killer"
//
// [REVISION SCORING 0-7, not 0-31]
// Player survives until rescue: 2 points
// Survivors saved (>3 passengers unturned): 2 points
// Rescue successful (<4 Zombies at rescue): 3 points
// HIGHEST SCORE POSSIBLE : 2+2+3 = 7
// Score Ranks: <3 "Zombie Meat", 3-4 "Survivor", 5-6 "Hero", 7 "Zombie Killer"
//
// Dialog:
// Characters say unique lines to player when talked to
// Characters say unique exchanges when together at location
// Infected Characters say subtle clue to excuse 'rest'
// Zombies say unique mumbled versions of character lines
// Player is silent
//
// Game Flow:
// 1. Introduction - Bon Voyage (8 turns)
// 2. Begin - Outbreak (food or drink?, which character 1st?)
// 3. Middle A - Havok (characters and items)
// 4. Plateau - S.O.S. (timer start)
// 5. Middle B - Survive (combat and hazards)
// 6. Crisis - Rescue? (timer end 32 turns)
// 7. End - Prologue and Scoring
//
// Dialog:
// Character dialog is presented in response to "Talk To" player commands or as a result of two characters in the same location
// Attacked character exclaims as injury or dying words
// Infected character dialog is a subtle reference to needing to stop and rest
// Zombie dialog is a mumbled version of the character's normal dialog to player
// Dialog Format: [player response A] [player response B] [char to char] [affirmative] [negative] [injury] [rest] [zombie speak]
// Captain: "Ahoy there" "If you need anything, just ask" "Is everyone all right?" "Certainly" "I'm afraid not" "Oh!" "I need to rest" "Ahhuugh thuuugh"
// 1st Mate: "Here to help" "At your service" "Can I get anyone anything?" "Yes" "Not really" "Ah!" "Let me sit down" "Huungh tuugh huuughp"
// Chef: "Stay out of my kitchen" "Bon appetit" "Is anyone hungry?" "Oui oui" "No" "Sacre bleu!" "Excuse moi" "Buughn appuughtuuught"
// Bartender: "It's always happy hour" "I'm here to listen" "Another round?" "Yeah" "Nope" "Hey!" "I'm going down" "Uuhts uuhlwuuhs huuuhpugh huugh"
// Mr. Rich: "Bully!" "I thought this was a private cruise" "Do you know how much money I have?" "Of course" "Not at all" "I say!" "I'm okay" "Buuulluughe"
// Mrs. Rich: "Well, I never!" "The service here is awful" "Can you help me?" "Yes indeed" "Certainly not" "My stars!" "I'm feeling faint" ""
// Prof. Smart: "This is fascinating" "I have a theory" "Don't you see what this means?" "I think so" "It's unknowable" "Stay back!" "I need a minute" "Thuughs ughs fuuusuunughtugh"
// Ms. Sass: "Seriously?" "Worst cruise ever" "Could this be any more tragic?" "Whatever" "Like I care" "Ew!" "Just leave me alone" "Surunghsluughee"

class   ZCPlayer {
    public:
    bitset<16>   pBits;
    // [ (3)currentLocation, (2)healthRemaining, (4)equipR, (4)equipL, (3)inventoryStored ]
};

class   ZCLocation {
    public:
    bitset<8>   locBits;
    // [ (1)lightsOn, (1)visited, (2)fireStatus, (4)fireDuration ]
};

class   ZCChar {
    public:
    bitset<8>   charBits;
    // [ (3)currentLocation, (1)waitHere, (2)healthRemaining, (1)isInfected, (1)isZombie ]
};
```

```cpp

class   ZCItem {
    public:
    bitset<8>   itemBits;
    // [ (3)itemID, (3)currentLocation, (1)isHeld, (1)isUsed ]
};

class   ZCGame {
    public:
    bitset<8>   gameBits;       // 1 byte
    bitset<8>   scoreBits;      // 1 byte
    ZCPlayer    player;         // 2 bytes
    ZCLocation* locations;      // 1 bytes
    ZCChar*     characters;     // 8 bytes
    ZCItem*     items;          // 12 bytes
                                // 32 bytes total working memory
    void        Initialize();
    int         DisambiguateLocation( string* words );
    int         DisambiguateItem( string* words );
    int         DisambiguateCharacter( string* words );
    int         FindItemInLocation( int itemType );
    int         FindItemInInventory( int itemType );
    void        ParseMove( string pMove );
    void        IncrementTurn();
    void        IncrementScore( int scoreAdd );
    bool        IsDarkArea();
    void        LocationNotice();
    void        FireNotice();
    void        ItemNotice();
    void        CharacterNotice();
    bool        CharacterAlive( int charIndex );
    void        ZombieChatter();
    void        CharacterChatter();
    string      InventoryFormat();
    bool        TakeItem( int itemIndex );
    bool        DropItem( int itemIndex );
    void        EquipItem( int itemIndex );
    void        EquipAny();
    bool        UseItem( int itemIndex );
    void        InfectCharacter( int charIndex );
    void        IncrementStory();
    void        IncrementInfection();
    void        IncrementFire();
    void        FireDamage();
    void        DoZombieMoves();
    void        DoZombieAttacks();
    void        DoCharacterMoves();
    void        DoPlayerAttack();
};

bitset<8>   SetNum( bitset<8> bits, int num, int startPos, int range ) {
    bits &= ~( (( 1 << range )-1) << startPos ); // clear
    bits |= ( num << startPos ); // set
    return bits;
}

bitset<16>   SetNum( bitset<16> bits, int num, int startPos, int range ) {
    bits &= ~( (( 1 << range )-1) << startPos );
    bits |= ( num << startPos );
    return bits;
}

int         GetNum( bitset<8> bits, int startPos, int range ) {
    unsigned short tmp = bits.to_ulong(); // required pre-process
    // static_cast<int>() required to convert double from pow()
    return ( ( tmp >>= startPos ) & ( static_cast<int>( pow( 2, range ) )-1 ) ); //
```

```
"(2^range)-1" = mask!
}

int         GetNum( bitset<16> bits, int startPos, int range ) {
    unsigned short tmp = bits.to_ulong();
    return ( ( tmp >>= startPos ) & ( static_cast<int>( pow( 2, range ) )-1 ) );
}

bitset<8>   SetBit( bitset<8> bits, bool b, bitset<8> mask ) {
    return ( ( b ) ? ( bits |= mask ) : ( bits &= ~mask ) );
}

bitset<16>  SetBit( bitset<16> bits, bool b, bitset<16> mask ) {
    return ( ( b ) ? ( bits |= mask ) : ( bits &= ~mask ) );
}

bool        GetBit( bitset<8> bits, bitset<8> mask ) {
    return ( ( bits & mask ) == mask );
}

bool        GetBit( bitset<16> bits, bitset<16> mask ) {
    return ( ( bits & mask ) == mask );
}

string      DebugBits( string name, bitset<8> bits ) {
    string retString = ". Debug Bits ";
    retString += name + " [";
    for (int i=0; i<8; i++) {
        if ( GetBit( bits, (1 << (7-i)) ) == 0 ) // reverse order
            retString += "0";
        else
            retString += "1";
    }
    retString += "]\n";
    return retString;
}

string      DebugBits( string name, bitset<16> bits ) {
    string retString = ". Debug Bits ";
    retString += name + " [";
    for (int i=0; i<16; i++) {
        if ( GetBit( bits, (1 << (15-i)) ) == 0 )
            retString += "0";
        else
            retString += "1";
        if ( i == 7 )
            retString += " ";
    }
    retString += "]\n";
    return retString;
}

string InsertNewlines( string input, int charWidth ) {
    string retString = input;
    int pos = 0;
    int nlPos = charWidth;
    bool done = ( nlPos > retString.length() );
    // start at first char position, step ahead charWidth, work backwards to find a
space and replace it with a newline, repeat
    while ( !done ) {
        for ( int i=(nlPos-1); i>pos; i-- ) {
            if ( retString.substr( i, 1 ) == " " ) {
                // space found
                string pre = retString.substr( 0, i );
                string post = retString.substr( (i+1), retString.npos );
                // replaced with newline
```

```cpp
                retString = pre + "\n" + post;
                // step forward charWidth from there
                pos = (i+2);
                nlPos = (pos+charWidth);
                done = ( post.length() < charWidth );
                break;
            }
        }
    }
    return retString;
}

string HelpFormat() {
    string retString = "";

    retString = "[HELP]";
    retString += "\n\n";
    retString += "Zombie Cruise is a text-based adventure, where player commands take the form of\nkeywords.";
    retString += "\n\n";
    retString += "To win, you will have to consider what is happening in the game, based on the\ntext presented. You will be able to perform actions and react to your\nsurroundings if you read carefully.";
    retString += "\n\n";
    retString += "Here are the commands available to you:";
    retString += "\n\n";
    retString += " help";
    retString += "\n";
    retString += " wait";
    retString += "\n";
    retString += " look / around / at <location> / <item> / <character> / myself";
    retString += "\n";
    retString += " go/move/run/walk to <location>";
    retString += "\n";
    retString += " inventory";
    retString += "\n";
    retString += " take <item> / all";
    retString += "\n";
    retString += " drop <item> / all";
    retString += "\n";
    retString += " equip <item> / any (allows attack, also protects from attack)";
    retString += "\n";
    retString += " attack *NOTE: must have item equipped to attack";
    retString += "\n";
    retString += " use <item> / <location-specific feature>";
    retString += "\n";
    retString += " quit";
    retString += "\n\n";
    retString += "Good luck.";
    retString += "\n";

    return retString;
}

void    ZCGame::Initialize() {
    // gameBits
    gameBits = 0 << 0;
    // scoreBits
    scoreBits = 0 << 0;
    // playerBits
    player.pBits = 0 << 0;
    locations = new ZCLocation[8];
    characters = new ZCChar[8];
    items = new ZCItem[12];
    for (int i=0; i<8; i++) {
        // locations
        locations[i].locBits = 1 << 7; // all lights on
        // characters
        characters[i].charBits = 1 << 5; // all start on foredeck
        characters[i].charBits = SetBit( characters[i].charBits, true, (1 << 4) ); // all wait at start
        characters[i].charBits = SetNum( characters[i].charBits, 3, 2, 2 ); // all have 3 health remaining
        // items
        items[i].itemBits = i << 5; // one of each item
        switch(i) {
        case 0:
            items[i].itemBits |= 0 << 2; // flare gun at bridge
            break;
        case 1:
            items[i].itemBits |= 3 << 2; // fire extinguisher at ballroom
            break;
        case 2:
            items[i].itemBits |= 4 << 2; // alcohol bottle at lounge
            break;
        case 3:
            items[i].itemBits |= 1 << 2; // diving knife at fore deck
            break;
        case 4:
            items[i].itemBits |= 2 << 2; // spear gun at aft deck
            break;
        case 5:
            items[i].itemBits |= 7 << 2; // wrench at engine room
            break;
        case 6:
            items[i].itemBits |= 5 << 2; // cleaver at kitchen
            break;
        case 7:
            items[i].itemBits |= 6 << 2; // fuel can at store room
            break;
        }
    }
    for (int i=8; i<12; i++) {
        items[i].itemBits = (rand() % 8) << 5; // random item
        items[i].itemBits |= (rand() % 8) << 2; // random location at start
    }
};

string StoryFormat( int storyStage ) {
    string retString = "";
    switch (storyStage) {
    case 0:
        retString = "It is a warm summer afternoon, and this evening's event is a private moonlight cruise hosted by Doctor Zilch in honor of a breakthrough drug discovery he plans to announce to selected guests and the press. You have been invited to cover the story for the local newspaper. Here at the marina, the cruise ship 'Hot Irony' is ready to recieve the small number of guests. A crew is aboard already to prepare the dinner service for this evening. You board the ship, and make your way to the lounge with the other guests. As the ship casts off, you note how beautiful the scene is as the setting sun glistens on the ocean's horizon.\n";
        break;
    case 1:
        retString = "The dinner bell is rung by Chef Rotisserie, and guests are welcomed to a long table set up in the ballroom. The Captain takes the seat at the head of the table, as a fine wine is poured for each guest by Dr. Zilch. 'This is a momentous occassion, my friends. You are truly in for a treat! Heee hee heee!' Professor Smart says, 'Well, we're all very excited to hear your announcement doctor.' 'Yes, yes. But first, a toast! To all our lessons of evolution, the wonders of nature, our more primal selves! To our success!' The bewildered guests pause before drinking the toast. 'Now, let me prepare for the surprise of the evening!' Dr. Zilch says as he suddenly exits the ballroom.\n";
        break;
    case 2:
        retString = "First mate Pole bursts into the ballroom. 'Captain Swell! The lifeboats! Someone has cast them all off!' 'What?! Well, for now, I suggest all our guests return to the lounge and wait, if you please.' And so ...\n";
        break;
    case 32:
        retString = "A bright light shines over the water; a spotlight from a coast guard cruiser. 'Attention! Be prepared to be boarded! This is the coast guard and we're here to help! Anyone holding weapons will be considered hostile! Stay calm and cooperate with us!' The cruiser comes alongside and holds long enough for a small team of armed men to board the ship.\n";
        break;
    case 33:
        retString = "A small uniformed coast guard team boards the ship ... And you hope that there are enough of them to handle any remaining zombies. Because if you missed some, and this infection spreads further, the whole world could be at risk. Fingers crossed ...\n";
        break;
    default:
        // no story
        break;
    }
    retString = InsertNewlines(retString, 79);
    return retString;
}

string LocationFormat( int locIndex ) {
        string retString = "";
    switch (locIndex) {
    case 0:
        retString = "\n[BRIDGE]";
        break;
    case 1:
        retString = "\n[FORE DECK]";
        break;
    case 2:
        retString = "\n[AFT DECK]";
        break;
    case 3:
        retString = "\n[BALLROOM]";
        break;
    case 4:
        retString = "\n[LOUNGE]";
        break;
    case 5:
        retString = "\n[KITCHEN]";
        break;
    case 6:
        retString = "\n[STORE ROOM]";
        break;
    case 7:
        retString = "\n[ENGINE ROOM]";
        break;
    }
    return retString;
}

string LocationDescriptionFormat( int locIndex ) {
    string retString = "";
    switch (locIndex) {
    case 0:
        retString = "\n[BRIDGE]";
        retString += "\nAt the highest point on the ship, the bridge is the command center with all\nship controls.";
        retString += "\nThere is a radio here, with emergency instructions for contacting the coast\nguard.";
        retString += "\nThere is a first aid kit here, with simple instructions to heal a variety of\nwounds.";
        retString += "\nFrom here, there is an exit to the Fore deck, the Aft deck and the Lounge.";
        break;
    case 1:
        retString = "\n[FORE DECK]";
        retString += "\nThis is a large open deck in the front of the ship and above the waves\nbelow.";
        retString += "\nFrom here, there is an exit to the Bridge, the Ballroom and the Store room.";
        break;
    case 2:
        retString = "\n[AFT DECK]";
        retString += "\nThis is an open air deck at the back of the ship with a view of the moon\nshining on the ocean.";
        retString += "\nFrom here, there is an exit to the Bridge, the Lounge and the Engine room.";
        break;
    case 3:
        retString = "\n[BALLROOM]";
        retString += "\nThis is an elaborate dancing and dining ballroom the serves as the primary\ngathering place on the ship.";
        retString += "\nFrom here, there is an exit to the Fore deck, the Lounge and the Kitchen.";
        break;
    case 4:
        retString = "\n[LOUNGE]";
        retString += "\nThis small dimly lit lounge is fit for mixing and mingling with other guests\non board.";
        retString += "\nFrom here, there is an exit to the Bridge, the Aft deck and the Ballroom.";
        break;
    case 5:
        retString = "\n[KITCHEN]";
        retString += "\nThis is a large galley kitchen capable of serving a few dozen people at a time.";
        retString += "\nThere is a first aid kit here, with simple instructions to heal a variety of\nwounds.";
        retString += "\nFrom here, there is an exit to the Ballroom, the Store room and the Engine\nroom.";
        break;
    case 6:
        retString = "\n[STORE ROOM]";
        retString += "\nThis is a utility storage room used for supplies and maintenance.";
        retString += "\nFrom here, there is an exit to the Kitchen, the Engine room and the Fore deck.";
        break;
    case 7:
        retString = "\n[ENGINE ROOM]";
        retString += "\nThis is the main engine room of the ship with a large diesel engine\ndominating the room.";
        retString += "\nFrom here, there is an exit to the Kitchen, the Store room and the Aft deck.";
        break;
    }
    return retString;
}

string ItemName( int itemIndex ) {
    string retString = "";
    switch (itemIndex) {
    case 0:
        retString = "a flare gun";
        break;
    case 1:
        retString = "a fire extinguisher";
        break;
    case 2:
        retString = "an alcohol bottle";
        break;
    case 3:
        retString = "a diving knife";
        break;
    case 4:
        retString = "a spear gun";
        break;
    case 5:
        retString = "a wrench";
        break;
    case 6:
        retString = "a cleaver";
        break;
    case 7:
        retString = "a fuel can";
        break;
    }
    return retString;
}

string ItemFormat( int itemIndex ) {
    string retString = "There is ";
    retString += ItemName( itemIndex );
    retString += " here.";
    return retString;
}

string ItemDescriptionFormat( int itemIndex ) {
    string retString = "";
    switch (itemIndex) {
    case 0:
        retString = "This hand-held flare gun is used to signal other ships with a magnesium flare.";
        break;
    case 1:
        retString = "This full-sized metal fire extinguisher is used to put out medium and small fires.";
        break;
    case 2:
        retString = "This is a full bottle of premium spirit alcohol.";
        break;
    case 3:
        retString = "This is a large hand-held blade used during diving for utility and safety.";
        break;
    case 4:
        retString = "This is a rifle-sized compressed air spear gun used in emergencies while diving.";
        break;
    case 5:
        retString = "This is a very large monkey wrench used during maintenance of heavy machinery.";
        break;
    case 6:
        retString = "This large meat cleaver is very sharp and used to de-bone and section large cuts of meat.";
        break;
    case 7:
        retString = "This two gallon fuel can is used to hold diesel fuel in reserve.";
        break;
    }
    retString = InsertNewlines(retString, 79);
    return retString;
}

string CharacterDescriptionFormat( int charIndex ) {
    string retString = "";
    switch (charIndex) {
    case 0:
        retString = "Captain Swell is a heavy set man with a thick salt and pepper beard. He wears the uniform of a ship's captain.";
        break;
    case 1:
        retString = "First mate Pole is a tall thin man with particularly good posture. He wears the uniform of a crewman.";
        break;
    case 2:
        retString = "Chef Rotisserie is a stout barrel-chested man with a long mustache and dark hair arranged in a comb-over.";
        break;
    case 3:
        retString = "Phil is the ship's bartender. He has tied his hair back in a short ponytail, and wears a red vest.";
        break;
    case 4:
        retString = "Mr. Rich is an elderly gentleman with short white hair, and wearing a fine blue suit with a red tie.";
        break;
    case 5:
        retString = "Mrs. Rich is a sophisticated woman with a long yellow evening gown and a red flower in her hair.";
        break;
    case 6:
        retString = "Prof. Smart is a tall man in wire-rimmed glasses, wearing a simple brown wool suit.";
        break;
    case 7:
        retString = "Ms. Sass is a short thin young woman with long black hair and glasses, wearing a dark purple dress.";
        break;
    }
    retString = InsertNewlines(retString, 79);
    return retString;
}

string CharacterName( int charIndex, bool zombie ) {
    string retString = "";
    if ( zombie )
        retString = "Zombie ";
    switch (charIndex) {
    case 0:
        retString += "Captain Swell";
        break;
    case 1:
        retString += "First Mate Pole";
        break;
    case 2:
        retString += "Chef Rotisserie";
        break;
    case 3:
        retString += "Phil";
        break;
    case 4:
        retString += "Mr. Rich";
        break;
    case 5:
        retString += "Mrs. Rich";
        break;
    case 6:
        retString += "Prof. Smart";
        break;
    case 7:
        retString += "Ms. Sass";
    }
    return retString;
}

string CharacterDialogResponse( int charIndex ) {
    string retString = "";
    int r = (rand() % 2);
    switch (charIndex) {
    case 0:
        if ( r == 0 )
            retString = "Ahoy there";
        else
            retString = "If you need anything, just ask";
        break;
    case 1:
        if ( r == 0 )
            retString = "Here to help";
        else
            retString = "At your service";
        break;
    case 2:
        if ( r == 0 )
            retString = "Stay out of my kitchen";
        else
            retString = "Bon appetit";
        break;
    case 3:
        if ( r == 0 )
            retString = "It's always happy hour";
        else
            retString = "I'm here to listen";
        break;
    case 4:
        if ( r == 0 )
            retString = "Bully!";
        else
            retString = "I thought this was a private cruise";
        break;
    case 5:
        if ( r == 0 )
            retString = "Well, I never!";
        else
            retString = "The service here is awful";
        break;
    case 6:
        if ( r == 0 )
            retString = "This is fascinating";
        else
            retString = "I have a theory";
        break;
    case 7:
        if ( r == 0 )
            retString = "Seriously?";
        else
            retString = "Worst cruise ever";
        break;
    }
    return retString;
}
```

```

string CharacterDialogQuestion( int charIndex ) {
    string retString = "";
    switch (charIndex) {
    case 0:
        retString = "Is everyone all right?";
        break;
    case 1:
        retString = "Can I get anyone anything?";
        break;
    case 2:
        retString = "Is anyone hungry?";
        break;
    case 3:
        retString = "Another round?";
        break;
    case 4:
        retString = "Do you know how much money I have?";
        break;
    case 5:
        retString = "Can you help me?";
        break;
    case 6:
        retString = "Don't you see what this means?";
        break;
    case 7:
        retString = "Could this be any more tragic?";
        break;
    }
    return retString;
}

string CharacterDialogAnswer( int charIndex ) {
    string retString = "";
    int r = (rand() % 2);
    switch (charIndex) {
    case 0:
        if ( r == 0 )
            retString = "Certainly";
        else
            retString = "I'm afraid not";
        break;
    case 1:
        if ( r == 0 )
            retString = "Yes";
        else
            retString = "Not really";
        break;
    case 2:
        if ( r == 0 )
            retString = "Oui oui";
        else
            retString = "No";
        break;
    case 3:
        if ( r == 0 )
            retString = "Yeah";
        else
            retString = "Nope";
        break;
    case 4:
        if ( r == 0 )
            retString = "Of course";
        else
            retString = "Not at all";
        break;
    case 5:
        if ( r == 0 )
            retString = "Yes indeed";
        else
            retString = "Certainly not";
        break;
    case 6:
        if ( r == 0 )
            retString = "I think so";
        else
            retString = "It's unknowable";
        break;
    case 7:
        if ( r == 0 )
            retString = "Whatever";
        else
            retString = "Like I care";
        break;
    }
    return retString;
}

string CharacterDialogExclamation( int charIndex ) {
    string retString = "";
    switch (charIndex) {
    case 0:
        retString = "Oh!";
        break;
    case 1:
        retString = "Ah!";
        break;
    case 2:
        retString = "Sacre bleu!";
        break;
    case 3:
        retString = "Hey!";
        break;
    case 4:
        retString = "I say!";
        break;
    case 5:
        retString = "My stars!";
        break;
    case 6:
        retString = "Stay back!";
        break;
    case 7:
        retString = "Eww!";
        break;
    }
    return retString;
}

string CharacterDialogResting( int charIndex ) {
    string retString = "";
    switch (charIndex) {
    case 0:
        retString = "I need to rest";
        break;
    case 1:
        retString = "Let me sit down";
        break;
    case 2:
        retString = "Excuse moi";
        break;
    case 3:
        retString = "I'm going down";
        break;
    case 4:
        retString = "I'm all right";
        break;
    case 5:
        retString = "I feel faint";
        break;
    case 6:
        retString = "I need a minute";
        break;
    case 7:
        retString = "Just leave me alone";
        break;
    }
    return retString;
}

string CharacterDialogZombie( int charIndex ) {
    string retString = "";
    switch (charIndex) {
    case 0:
        retString = "Ahhuugh thuuugh";
        break;
    case 1:
        retString = "Huungh tuugh huuughp";
        break;
    case 2:
        retString = "Buughn appuughtuuught";
        break;
    case 3:
        retString = "Uuhts uuhlwuuhs huuuhpugh huugh";
        break;
    case 4:
        retString = "Buuulluughe";
        break;
    case 5:
        retString = "Wuuulh uuh nuuvvuugh";
        break;
    case 6:
        retString = "Thuughs ughs fuuusuunughtugh";
        break;
    case 7:
        retString = "Surunghsluughee";
        break;
    }
    return retString;
}

string ZCGame::InventoryFormat() {
    string retString = "\n You have ";
    int itemCount = 0;
    for ( int i=0; i<12; i++ ) {
        if ( GetBit(items[i].itemBits, (1<<1)) ) {
            itemCount++;
            retString += "\n";
            switch ( GetNum(items[i].itemBits,5,3) ) {
            case 0:
                retString += "  a flare gun";
                break;
            case 1:
                retString += "  a fire extinguisher";
                break;
            case 2:
                retString += "  an alcohol bottle";
                break;
            case 3:
                retString += "  a diving knife";
                break;
            case 4:
                retString += "  a spear gun";
                break;
            case 5:
                retString += "  a wrench";
                break;
            case 6:
                retString += "  a cleaver";
                break;
            case 7:
                retString += "  a fuel can";
                break;
            }
        }
    }
    if ( itemCount == 0 ) {
        retString += "nothing of use.";
    }
    // equipped items
    int equipItemA = GetNum( player.pBits, 7, 4 );
    int equipItemB = GetNum( player.pBits, 3, 4 );
    if ( equipItemA < 12 ) {
        retString += "\n You hold " + ItemName( GetNum( items[equipItemA].itemBits, 5, 3 ) ) + " in your right hand";
    }
    if ( equipItemB < 12 ) {
        retString += "\n You hold " + ItemName( GetNum( items[equipItemB].itemBits, 5, 3 ) ) + " in your left hand";
    }
    return retString;
}

bool ZCGame::TakeItem( int itemIndex ) {
    bool retBool = false;
    int inventoryCount = GetNum( player.pBits, 0, 3 );
    // if item is not taken and item location is player location, take
    if ( !GetBit( items[itemIndex].itemBits, (1<<1) ) && ( GetNum(items[itemIndex].itemBits, 2, 3) == GetNum(player.pBits, 13, 3) ) ) {
        if ( GetBit( items[itemIndex].itemBits, (1<<0) ) ) {
            // used items cannot be taken
        }
        else if ( inventoryCount < 7 ) {
            // taken if ( inventory count < 7 )
            items[itemIndex].itemBits = SetBit( items[itemIndex].itemBits, true, (1<<1) );
            retBool = true;
            // increment inventory count if taken
            inventoryCount++;
            player.pBits = SetNum( player.pBits, inventoryCount, 0, 3 );
        }
        else {
            cout << "\nYou hold too much, so " << ItemName( GetNum(items[itemIndex].itemBits, 5, 3) ) << " is not taken.";
        }
    }
    return retBool;
}

bool ZCGame::DropItem( int itemIndex ) {
    bool retBool = false;
    int itemType = GetNum( items[itemIndex].itemBits, 5, 3 );
    if ( GetBit( items[itemIndex].itemBits, (1<<1) ) ) {
        // if equipped, un-equip (set to value 15)
```

```cpp
        if ( itemIndex == GetNum( player.pBits, 7, 4 ) ) {
            player.pBits = SetNum( player.pBits, 15, 7, 4 );
            cout << "\nNow " << ItemName( itemType ) << " is no longer equipped in your right hand.";
        }
        else if ( itemIndex == GetNum( player.pBits, 3, 4 ) ) {
            player.pBits = SetNum( player.pBits, 15, 3, 4 );
            cout << "\nNow " << ItemName( itemType ) << " is no longer equipped in your left hand.";
        }
        // drop here
        items[itemIndex].itemBits = SetBit( items[itemIndex].itemBits, false, (1<<1) );
        items[itemIndex].itemBits = SetNum( items[itemIndex].itemBits, GetNum( player.pBits, 13, 3 ), 2, 3 );
        retBool = true;
        // decrement inventory count
        int inventoryCount = GetNum( player.pBits, 0, 3 );
        inventoryCount--;
        player.pBits = SetNum( player.pBits, inventoryCount, 0, 3 );
    }
    return retBool;
}

void ZCGame::EquipItem( int itemType ) {
    int itemIndexA = GetNum( player.pBits, 7, 4 );
    int itemIndexB = GetNum( player.pBits, 3, 4 );
    // check for items matching type in inventory
    int itemIndex = FindItemInInventory( itemType );
    if ( itemIndex > -1 ) {
        if ( GetBit( items[itemIndex].itemBits, (1<<1) ) && itemIndexA != itemIndex && itemIndexB != itemIndex ) {
            // items held and not already equipped
            if ( itemIndexA == 15 ) {
                itemIndexA = itemIndex;
                items[itemIndex].itemBits = SetBit( items[itemIndex].itemBits, true, (1<<1) );
                player.pBits = SetNum( player.pBits, itemIndex, 7, 4 );
                cout << "\nNow " << ItemName(GetNum( items[itemIndexA].itemBits, 5, 3 )) << " is equipped in your right hand.";
            }
            else if ( itemIndexB == 15 ) {
                itemIndexB = itemIndex;
                items[itemIndex].itemBits = SetBit( items[itemIndex].itemBits, true, (1<<1) );
                player.pBits = SetNum( player.pBits, itemIndex, 3, 4 );
                cout << "\nNow " << ItemName(GetNum( items[itemIndexB].itemBits, 5, 3 )) << " is equipped in your left hand.";
            }
            else {
                // un-equip left hand item
                items[itemIndexB].itemBits = SetBit( items[itemIndexB].itemBits, false, (1<<1) );
                cout << "\nYou keep " << ItemName(GetNum( items[itemIndexB].itemBits, 5, 3 )) << ", but ...";
                // switch right hand item to left
                itemIndexB = itemIndexA;
                player.pBits = SetNum( player.pBits, itemIndexB, 3, 4 );
                // equip new item right hand
                itemIndexA = itemIndex;
                player.pBits = SetNum( player.pBits, itemIndexA, 7, 4 );
                cout << "\nIn your right hand, you now hold " << ItemName(GetNum( items[itemIndexA].itemBits, 5, 3 )) << " ...";
                cout << "\nand " << ItemName(GetNum( items[itemIndexB].itemBits, 5, 3 )) << " is equipped in your left hand.";
            }
        }
        else if ( itemIndexA == itemIndex ) {
            cout << "\nYou already have that item equipped in your right hand.";
        }
        else if ( itemIndexB == itemIndex ) {
            cout << "\nYou already have that item equipped in your left hand.";
        }
    }
    else {
        cout << "\nYou do not have that item.";
    }
}

void ZCGame::EquipAny() {
    int itemIndexA = GetNum( player.pBits, 7, 4 );
    int itemIndexB = GetNum( player.pBits, 3, 4 );
    for ( int i=0; i<12; i++ ) {
        if ( GetBit( items[i].itemBits, (1<<1) ) && itemIndexA != i && itemIndexB != i ) {
            // items held and not already equipped
            if ( itemIndexA == 15 ) {
                itemIndexA = i;
                items[i].itemBits = SetBit( items[i].itemBits, true, (1<<1) );
                player.pBits = SetNum( player.pBits, i, 7, 4 );
                cout << "\nNow " << ItemName(GetNum( items[itemIndexA].itemBits, 5, 3 )) << " is equipped in your right hand.";
            }
            else if ( itemIndexB == 15 ) {
                itemIndexB = i;
                items[i].itemBits = SetBit( items[i].itemBits, true, (1<<1) );
                player.pBits = SetNum( player.pBits, i, 3, 4 );
                cout << "\nNow " << ItemName(GetNum( items[itemIndexB].itemBits, 5, 3 )) << " is equipped in your left hand.";
            }
        }
    }
}

bool ZCGame::UseItem( int itemIndex ) {
    bool retBool = false;
    // if one-use item, use and return true
    int itemType = GetNum( items[itemIndex].itemBits, 5, 3 );
    if ( itemType == 0 || itemType == 1 || itemType == 2 || itemType == 4 || itemType == 7 ) {
        if ( itemType != 1 ) {
            // all except fire extinguisher can only be used once
            items[itemIndex].itemBits = SetBit( items[itemIndex].itemBits, true, (1<<0) );
        }
        // if alcohol bottle or fuel can, make current location flammable
        int currRoom = GetNum( player.pBits, 13, 3 ); // current room is where player is
        items[itemIndex].itemBits = SetNum( items[itemIndex].itemBits, currRoom, 2, 3 ); // set item location here
        if ( itemType == 2 || itemType == 7 ) {
            if ( GetNum( locations[currRoom].locBits, 4, 2 ) == 2 ) {
                // on fire area resets fire timer
                cout << "\n ... and fuel is added to the fire.";
                locations[ currRoom ].locBits = SetNum( locations[ currRoom ].locBits, 1, 0, 4 ); // fire timer reset
            }
            else {
                cout << "\n ... flammable liquid spills everywhere.";
                locations[ currRoom ].locBits = SetNum( locations[ currRoom ].locBits, 1, 4, 2 ); // room flammable
            }
        }
        else if ( itemType == 1 ) {
            // fire extinguisher puts out fire
            if ( GetNum( locations[currRoom].locBits, 4, 2 ) == 2 ) {
                // on fire area is extinguished
                cout << "\nWith some work, the fire is extinguished.";
                locations[ currRoom ].locBits = SetNum( locations[ currRoom ].locBits, 3, 4, 2 ); // room burnt
            }
            else {
                cout << "\nWhite fire retardant coats the area, and slowly disappears.";
            }
        }
        else if ( itemType == 0 ) {
            // flare gun ignites flammable areas
            if ( GetNum( locations[currRoom].locBits, 4, 2 ) == 1 ) {
                cout << "\n ... and the area bursts into flames.";
                locations[ currRoom ].locBits = SetNum( locations[ currRoom ].locBits, 2, 4, 2 ); // room flammable
                locations[ currRoom ].locBits = SetNum( locations[ currRoom ].locBits, 1, 0, 4 ); // fire timer reset
            }
        }
        retBool = true;
    }
    return retBool;
}

void ZCGame::InfectCharacter( int charIndex ) {
    characters[charIndex].charBits = SetBit( characters[charIndex].charBits, true, (1<<1) );
    // force immobile
    characters[charIndex].charBits = SetBit( characters[charIndex].charBits, true, (1<<4) );
}

string PlayerPrompt() {
    cout << "[Your Turn] > ";
    string playerMove;
    getline(cin, playerMove );
    return playerMove;
}

string* WordBreak( string s ) {
    string* retStrings = new string[5];
    int wordCount = 0;
    string workS = s;
    string currWord = "";
    int wordLen = 0;
    while ( wordLen != workS.npos ) {
        wordLen = workS.find( " ", 0 );
        currWord = workS.substr( 0, wordLen );
        workS = workS.substr( (wordLen+1), workS.npos );
        retStrings[ wordCount++ ] = currWord;
    }
    return retStrings;
}

int ZCGame::DisambiguateLocation( string* w ) {
    int retInt = -1;
    // Locations: [8] Bridge, Fore Deck, Aft Deck, Ballroom, Lounge, Kitchen, Store Room, Engine Room
    if ( w[0] == "Bridge" || w[0] == "bridge" || w[1] == "Bridge" || w[1] == "bridge" ) {
        retInt = 0;
    }
    else if ( w[0] == "Foredeck" || w[0] == "foredeck" || w[1] == "Foredeck" || w[1] == "foredeck" || w[0] == "Fore" || w[0] == "fore" || w[1] == "Fore" || w[1] == "fore" ) {
        retInt = 1;
    }
    else if ( w[0] == "Aftdeck" || w[0] == "aftdeck" || w[1] == "Aftdeck" || w[1] == "aftdeck" || w[0] == "Aft" || w[0] == "aft" || w[1] == "Aft" || w[1] == "aft" ) {
        retInt = 2;
    }
    else if ( w[0] == "Ballroom" || w[0] == "ballroom" || w[1] == "Ballroom" || w[1] == "ballroom" || w[0] == "Ball" || w[0] == "ball" || w[1] == "Ball" || w[1] == "ball" ) {
        retInt = 3;
    }
    else if ( w[0] == "Lounge" || w[0] == "lounge" || w[1] == "Lounge" || w[1] == "lounge" ) {
        retInt = 4;
    }
    else if ( w[0] == "Kitchen" || w[0] == "kitchen" || w[1] == "Kitchen" || w[1] == "kitchen" ) {
        retInt = 5;
    }
    else if ( w[0] == "Storeroom" || w[0] == "storeroom" || w[1] == "Storeroom" || w[1] == "storeroom" || w[0] == "Store" || w[0] == "store" || w[1] == "Store" || w[1] == "store" ) {
        retInt = 6;
    }
    else if ( w[0] == "Engineroom" || w[0] == "engineroom" || w[1] == "Engineroom" || w[1] == "engineroom" || w[0] == "Engine" || w[0] == "engine" || w[1] == "Engine" || w[1] == "engine" ) {
        retInt = 7;
    }
    return retInt;
}

int ZCGame::DisambiguateItem( string* w ) {
    int retInt = -1;
    // Items: [8] Flare Gun, Fire Extinguisher, Alcohol Bottle, Diving Knife, Spear Gun, Wrench, Cleaver, Fuel Can
    if ( w[0] == "Flare" || w[0] == "flare" || w[1] == "Flare" || w[1] == "flare" || w[0] == "Flaregun" || w[0] == "flaregun" || w[1] == "Flaregun" || w[1] == "flaregun" ) {
        retInt = 0;
    }
    else if ( w[0] == "Fire" || w[0] == "fire" || w[1] == "Fire" || w[1] == "fire" || w[0] == "Extinguisher" || w[0] == "extinguisher" || w[1] == "Extinguisher" || w[1] == "extinguisher" ) {
        retInt = 1;
    }
    else if ( w[0] == "Alcohol" || w[0] == "alcohol" || w[1] == "Alcohol" || w[1] == "alcohol" || w[0] == "Bottle" || w[0] == "bottle" || w[1] == "Bottle" || w[1] == "bottle" ) {
        retInt = 2;
    }
    else if ( w[0] == "Diving" || w[0] == "diving" || w[1] == "Diving" || w[1] == "diving" || w[0] == "Knife" || w[0] == "knife" || w[1] == "Knife" || w[1] == "knife" ) {
        retInt = 3;
    }
    else if ( w[0] == "Spear" || w[0] == "spear" || w[1] == "Spear" || w[1] == "spear" || w[0] == "Speargun" || w[0] == "speargun" || w[1] == "Speargun" || w[1] == "speargun" ) {
        retInt = 4;
    }
    else if ( w[0] == "Wrench" || w[0] == "wrench" || w[1] == "Wrench" || w[1] == "wrench" ) {
        retInt = 5;
    }
    else if ( w[0] == "Cleaver" || w[0] == "cleaver" || w[1] == "Cleaver" || w[1] == "cleaver" ) {
        retInt = 6;
    }
    else if ( w[0] == "Fuel" || w[0] == "fuel" || w[1] == "Fuel" || w[1] == "fuel" || w[0] == "Can" || w[0] == "can" || w[1] == "Fuelcan" || w[1] == "fuelcan" ) {
        retInt = 7;
    }
    return retInt;
}

int ZCGame::DisambiguateCharacter( string* w ) {
    int retInt = -1;
    // People: [8] Captain, 1st Mate, Chef, Bartender, Mr. Rich, Mrs. Rich, Prof. Smart, Ms. Sass
    if ( w[0] == "Swell" || w[0] == "swell" || w[1] == "Swell" || w[1] == "swell" ) {
        retInt = 0;
    }
    else if ( w[0] == "Pole" || w[1] == "Pole" || w[2] == "Pole" || w[0] == "pole" || w[1] == "pole" || w[2] == "pole" ) {
        retInt = 1;
    }
    else if ( w[0] == "Chef" || w[1] == "chef" || w[1] == "Rotisserie" || w[0] == "chef" || w[1] == "chef" || w[1] == "rotisserie" || w[2] == "rotisserie" ) {
        retInt = 2;
    }
    else if ( w[0] == "Phil" || w[0] == "phil" || w[1] == "Phil" || w[1] == "phil" ) {
        retInt = 3;
    }
    else if ( ( w[0] == "Mr." || w[0] == "mr." || w[0] == "mr" ) && ( w[1] == "Rich" || w[1] == "rich" ) ) {
        retInt = 4;
    }
    else if ( ( w[0] == "Mrs." || w[0] == "mrs." || w[0] == "mrs" ) && ( w[1] == "Rich" || w[1] == "rich" ) ) {
        retInt = 5;
    }
    else if ( w[0] == "Smart" || w[0] == "smart" || w[1] == "Smart" || w[1] == "smart" ) {
        retInt = 6;
    }
    else if ( w[0] == "Sass" || w[0] == "sass" || w[1] == "Sass" || w[1] == "sass" ) {
        retInt = 7;
    }
    return retInt;
}

int ZCGame::FindItemInLocation( int itemType ) {
    // return index of item among all including additional random items (8-11)
    int retInt = -1; // -1 = not found
    for (int i=0; i<12; i++) {
        if ( GetNum( items[i].itemBits, 5, 3 ) == itemType ) {
            if ( GetNum( player.pBits, 13, 3 ) == GetNum( items[i].itemBits, 2, 3 ) ) {
                // exclude used items
                if ( !GetBit( items[i].itemBits, (1<<0) ) ) {
                    retInt = i;
                    break;
                }
                else {
                    retInt = -2; // -2 = found in location but used (feedback to
player)
                    }
                }
            }
        }
        return retInt;
    }

    int ZCGame::FindItemInInventory( int itemType ) {
        // return index of item among all including additional random items (8-11)
        int retInt = -1;
        for (int i=0; i<12; i++) {
            if ( GetNum( items[i].itemBits, 5, 3 ) == itemType ) {
                if ( GetBit( items[i].itemBits, (1<<1) ) ) {
                    retInt = i;
                    break;
                }
            }
        }
        return retInt;
    }

    int FindRoomFromExit( int currentRoom, int exitIndex ) {
        // return room index from current room and exit index
        int retInt = currentRoom;
        // Exits: 0=1-2-4, 1=0-3-6, 2=0-4-7, 3=1-4-5, 4=0-2-3, 5=3-6-7, 6=1-5-7,
7=2-5-6
        switch (currentRoom) {
        case 0:
            if ( exitIndex == 0 )
                retInt = 1;
            else if ( exitIndex == 1 )
                retInt = 2;
            else
                retInt = 4;
            break;
        case 1:
            if ( exitIndex == 0 )
                retInt = 0;
            else if ( exitIndex == 1 )
                retInt = 3;
            else
                retInt = 6;
            break;
        case 2:
            if ( exitIndex == 0 )
                retInt = 0;
            else if ( exitIndex == 1 )
                retInt = 4;
            else
                retInt = 7;
            break;
        case 3:
            if ( exitIndex == 0 )
                retInt = 1;
            else if ( exitIndex == 1 )
                retInt = 4;
            else
                retInt = 5;
            break;
        case 4:
            if ( exitIndex == 0 )
                retInt = 0;
            else if ( exitIndex == 1 )
                retInt = 2;
            else
```

```cpp
            retInt = 3;
        break;
    case 5:
        if ( exitIndex == 0 )
            retInt = 3;
        else if ( exitIndex == 1 )
            retInt = 6;
        else
            retInt = 7;
        break;
    case 6:
        if ( exitIndex == 0 )
            retInt = 1;
        else if ( exitIndex == 1 )
            retInt = 5;
        else
            retInt = 7;
        break;
    case 7:
        if ( exitIndex == 0 )
            retInt = 2;
        else if ( exitIndex == 1 )
            retInt = 5;
        else
            retInt = 6;
        break;
    }
    return retInt;
}

void ZCGame::ParseMove( string pMove ) {
    // break move into words (5 words max)
    string* word = new string[5];
    word = WordBreak( pMove );
    // match to commands
    // identify command targets
    if ( word[0] == "debug" ) {
        if ( word[1] == "game" ) {
            cout << DebugBits( "gameBits", gameBits );
        }
        else if ( word[1] == "score" ) {
            cout << DebugBits( "scoreBits", scoreBits );
        }
        else if ( word[1] == "player" ) {
            cout << DebugBits( "playerBits", player.pBits );
        }
        else if ( word[1] == "locations" || word[1] == "loc" || word[1] == "locs" ) {
            for (int i=0;i<8;i++) {
                cout << DebugBits( "locBits", locations[i].locBits );
            }
        }
        else if ( word[1] == "characters" || word[1] == "char" || word[1] == "chars" ) {
            for (int i=0;i<8;i++) {
                cout << DebugBits( "charBits", characters[i].charBits );
            }
        }
        else if ( word[1] == "items" || word[1] == "item" ) {
            for (int i=0;i<8;i++) {
                cout << DebugBits( "itemBits", items[i].itemBits );
            }
        }
        else {
            // debug all
            cout << DebugBits( "playerBits", player.pBits );
```

```
                cout << DebugBits( "gameBits", gameBits );
                cout << DebugBits( "scoreBits", scoreBits );
                for (int i=0;i<8;i++) {
                    cout << DebugBits( "locBits", locations[i].locBits );
                }
                for (int i=0;i<8;i++) {
                    cout << DebugBits( "charBits", characters[i].charBits );
                }
                for (int i=0;i<8;i++) {
                    cout << DebugBits( "itemBits", items[i].itemBits );
                }
            }
            cout << "\n";
        }
        else if ( word[0] == "quit" ) {
            gameBits = SetBit( gameBits, true, (1<<0) );
        }
        else if ( word[0] == "cheat" ) {
            if ( word[1] == "health" || word[1] == "heal" ) {
                if ( GetNum( player.pBits, 11, 2 ) < 3 ) {
                    player.pBits = SetNum( player.pBits, 3, 11, 2 );
                    cout << "\n-cheat- You feel much better.";
                }
            }
            else if ( word[1] == "teleport" || word[1] == "tport" ) {
                string* checkWords = new string[3];
                for (int i=0; i<3; i++) {
                    checkWords[i] = word[(i+2)];
                }
                int tLoc = DisambiguateLocation( checkWords );
                if ( tLoc > -1 && tLoc != GetNum( player.pBits, 13, 3 ) ) {
                    player.pBits = SetNum( player.pBits, tLoc, 13, 3 );
                    cout << "\n-cheat- You have been transported to a new location.";
                }
            }
        }
        else if ( word[0] == "help" || word[0] == "h" || word[0] == "?" ) {
            cout << HelpFormat();
        }
        else if ( word[0] == "wait" || word[0] == "..." ) {
            cout << "\n...";
            IncrementTurn();
        }
        else if ( word[0] == "look" ) {
            if ( word[1] == "at" ) {
                if ( word[2] == "me" || word[2] == "myself" ) {
                    // damage assessment based on player health
                    switch( GetNum( player.pBits, 11, 2 ) ) {
                    case 0:
                        cout << "\nYou are dying.";
                        break;
                    case 1:
                        cout << "\nYou are in very rough shape.";
                        break;
                    case 2:
                        cout << "\nYou are hurt, but still okay.";
                        break;
                    case 3:
                        cout << "\nYou are looking good.";
                        break;
                    }
                    IncrementTurn();
                }
                else if ( word[2] == "room" || word[2] == "area" ) {
                    // area look, if lit
                    if ( !IsDarkArea() ) {
```

```cpp
                    cout << LocationDescriptionFormat( GetNum( player.pBits, 13, 3 ) );
                }
                else {
                    cout << "\nIt's too dark here to see.";
                }
                IncrementTurn();
            }
            else {
                // 'look at' check
                int playerLoc = GetNum( player.pBits, 13, 3 );
                string* checkWords = new string[3];
                for (int i=0; i<3; i++) {
                    checkWords[i] = word[(i+2)];
                }
                // char match (must be in area, alive, non-infected and non-zombie)
                int charIdx = -1;
                charIdx = DisambiguateCharacter( checkWords );
                if ( charIdx > -1 ) {
                    if ( playerLoc == GetNum( characters[charIdx].charBits, 5, 3 ) ) {
                        if ( CharacterAlive(charIdx) && !GetBit( characters[charIdx].charBits, (1<<1) ) && !GetBit( characters[charIdx].charBits, (1<<0) ) ) {
                            cout << "\n" << CharacterDescriptionFormat(charIdx);
                        }
                        else {
                            cout << "\n" << CharacterName( charIdx, GetBit( characters[charIdx].charBits, (1<<0) ) ) << " is not looking well.";
                        }
                        IncrementTurn();
                    }
                    else {
                        cout << "\nThat person is not here.";
                    }
                }
                // check item (must be in inventory or in location)
                int itemIdx = -1;
                itemIdx = DisambiguateItem( checkWords );
                if ( itemIdx > -1 ) {
                    // check aux items
                    itemIdx = FindItemInInventory( itemIdx );
                    if ( itemIdx == -1 ) {
                        itemIdx = FindItemInLocation( DisambiguateItem( checkWords ) );
                    }
                    if ( itemIdx > -1 ) {
                        cout << "\n" << ItemDescriptionFormat( itemIdx );
                        IncrementTurn();
                    }
                    else {
                        cout << "\nYou do not see that item.";
                    }
                }
                // check location (must be player location -> area look)
                int locIdx = -1;
                locIdx = DisambiguateLocation( checkWords );
                if ( locIdx > -1 ) {
                    if ( playerLoc == locIdx ) {
                        cout << LocationDescriptionFormat( GetNum( player.pBits, 13, 3 ) );
                        IncrementTurn();
                    }
                }
            }
        }
        else if ( word[1] == "around" || word[1] == "here" ) {
            // area look, if lit
            if ( !IsDarkArea() ) {
                cout << LocationDescriptionFormat( GetNum( player.pBits, 13, 3 ) );
            }
            else {
                cout << "\nIt's too dark here to see.";
            }
            IncrementTurn();
        }
        else {
            // area look, if lit
            if ( !IsDarkArea() ) {
                cout << LocationDescriptionFormat( GetNum( player.pBits, 13, 3 ) );
            }
            else {
                cout << "\nIt's too dark here to see.";
            }
            IncrementTurn();
        }
    }
    else if ( word[0] == "inventory" || word[0] == "inv" || word[0] == "i" ) {
        cout << InventoryFormat();
        IncrementTurn();
    }
    else if ( word[0] == "take" ) {
        // 'take' check
        int playerLoc = GetNum( player.pBits, 13, 3 );
        string* checkWords = new string[4];
        for (int i=0; i<4; i++) {
            checkWords[i] = word[(i+1)];
        }
        // item match (must be in area, not used)
        int itemIdx = -1;
        itemIdx = DisambiguateItem( checkWords );
        if ( itemIdx > -1 ) {
            // check aux items in location
            itemIdx = FindItemInLocation( itemIdx );
            if ( itemIdx > -1 ) {
                if ( !GetBit( items[itemIdx].itemBits, (1<<1) ) ) {
                    if ( TakeItem( itemIdx ) ) {
                        cout << "\nYou take " << ItemName( GetNum( items[itemIdx].itemBits, 5, 3 ) );
                        IncrementTurn();
                    }
                }
            }
            else if ( itemIdx == -2 ) {
                cout << "\nThat item is used and cannot be taken.";
            }
            else {
                cout << "\nThat item is not here.";
            }
        }
        else if ( word[1] == "all" || word[1] == "everything" ) {
            for (int i=0; i<12; i++) {
                if ( TakeItem( i ) ) {
                    cout << "\nYou take " << ItemName( GetNum( items[i].itemBits, 5, 3 ) );
                }
            }
            IncrementTurn();
        }
        else {
            cout << "\nIt is not clear what item you want to take from here.\n(Name the item you see here or try 'take all' to take everything)";
        }
    }
    else if ( word[0] == "drop" ) {
        // 'drop' check
        string* checkWords = new string[4];
        for (int i=0; i<4; i++) {
            checkWords[i] = word[(i+1)];
        }
        // item match (must be in inventory)
        int itemIdx = -1;
        itemIdx = DisambiguateItem( checkWords );
        if ( itemIdx > -1 ) {
            // check aux items in inventory
            itemIdx = FindItemInInventory( itemIdx );
            if ( itemIdx > -1 ) {
                if ( DropItem(itemIdx) ) {
                    cout << "\nYou drop " << ItemName( GetNum( items[itemIdx].itemBits, 5, 3 ) );
                    IncrementTurn();
                }
            }
            else  {
                cout << "\nYou do not have that item.";
            }
        }
        else if ( word[1] == "all" || word[1] == "everything" ) {
            for (int i=0; i<12; i++) {
                if ( DropItem(i) ) {
                    cout << "\nYou drop " << ItemName( GetNum( items[i].itemBits, 5, 3 ) );
                }
            }
            IncrementTurn();
        }
        else {
            cout << "\nIt is not clear what item in your inventory you want to drop.\n(Name the item you have taken or try 'drop all' to drop everything)";
        }
    }
    else if ( word[0] == "equip" ) {
        // item match (must be in inventory)
        int itemType = -1;
        string* checkWords = new string[4];
        for (int i=0; i<4; i++) {
            checkWords[i] = word[(i+1)];
        }
        itemType = DisambiguateItem( checkWords );
        if ( itemType > -1 ) {
            EquipItem( itemType ); // use item type here, EquipItem sorts out if in inventory
            IncrementTurn();
        }
        else if ( word[1] == "any" || word[1] == "anything" || word[1] == "everything" ) {
            EquipAny();
            IncrementTurn();
        }
        else {
            cout << "\nIt is not clear what item you want to equip in your hand.\n(Name the item you have taken to equip or try 'equip any' to grab anything)";
        }
    }
    else if ( word[0] == "attack" ) {
        DoPlayerAttack();
        IncrementTurn();
    }
    else if ( word[0] == "use" ) {
```

```cpp
        int playerLoc = GetNum( player.pBits, 13, 3 );
        // item match (must be in inventory, usable type)
        string* checkWords = new string[4];
        for ( int i=0; i<4; i++ ) {
            checkWords[i] = word[(i+1)];
        }
        int itemType = DisambiguateItem(checkWords);
        int itemIdx = FindItemInInventory( itemType );
        if ( itemType > -1 && itemIdx > -1 ) {
            // use item
            if ( !UseItem( itemIdx ) ) {
                cout << "\nThere is nothing this item can do.";
            }
            else if ( itemType != 1 ) {
                DropItem( itemIdx ); // single use items are dropped, but not fire extinguisher
            }
        }
        // area match
        else if ( word[1] == "radio" ) {
            if ( playerLoc == 0 ) {
                if ( GetBit( gameBits, (1<<3) ) ) {
                    cout << "\nYou have already called for help and the coast guard is on the way.";
                }
                else {
                    // make sure infection has already been released (infection timer gets reset as a game balance feature)
                    int numDeadChars = 0;
                    int numZombiesOnboard = 0;
                    int numCharsAlive = 0;
                    for ( int i=0; i<8; i++ ) {
                        if ( !CharacterAlive(i) ) {
                            numDeadChars++;
                        }
                        else if ( GetBit( characters[i].charBits, (1<<0) ) ) {
                            numZombiesOnboard++;
                        }
                        else {
                            numCharsAlive++;
                        }
                    }
                    if ( numDeadChars > 0 || numZombiesOnboard > 0 ) {
                        // radio not used, infection released
                        gameBits = SetBit( gameBits, true, (1<<3) );
                        if ( numCharsAlive > 0 ) {
                            cout << "\n YOU [On Radio]: Mayday! Mayday! It's ... Send armed response! We need help!";
                        }
                        else {
                            cout << "\n YOU [On Radio]: Mayday! Mayday! The ship ... There's ... Just send help!";
                        }
                        // skip increment turn (cheat to add a turn before response)
                    }
                    else {
                        // using radio, but infection not released yet
                        cout << "\n Coast Guard [Radio]: This channel is reserved for emergencies only. Over.";
                        IncrementTurn();
                    }
                }
            }
            else {
                cout << "\nThere is no radio here. (Do you think there might be one
on the bridge?)";
                }
            }
            else if ( word[1] == "first" || word[2] == "first" || word[2] == "aid" || word[3] == "aid" || word[3] == "kit" || word[4] == "kit" ) {
                // first aid kits on the bridge and in the kitchen
                if ( playerLoc == 0 || playerLoc == 5 ) {
                    // first aid kit heals if hurt
                    if ( GetNum( player.pBits, 11, 2 ) < 3 ) {
                        cout << "\nYou're able to heal your wounds somewhat, and you feel better.";
                        int health = GetNum( player.pBits, 11, 2 );
                        player.pBits = SetNum( player.pBits, (health+1), 11, 2 );
                        IncrementTurn();
                    }
                    else {
                        cout << "\nAfter rummaging through the first aid kit, you decide you're feeling fine.";
                        IncrementTurn();
                    }
                }
            }
            else {
                cout << "\nIt is not clear what you want to use.\n(Name the item you have taken or the usable feature you see here)";
            }
        }
        else if ( word[0] == "talk" ) {
            if ( word[1] == "to" ) {
                // 'talk to' check
                string* checkWords = new string[3];
                for (int i=0; i<3; i++) {
                    checkWords[i] = word[(i+2)];
                }
                int playerLoc = GetNum( player.pBits, 13, 3 );
                int charIdx = -1;
                charIdx = DisambiguateCharacter( checkWords );
                if ( charIdx > -1 ) {
                    // char match (must be in area, alive, non-infected and non-zombie)
                    if ( playerLoc == GetNum( characters[charIdx].charBits, 5, 3 ) && CharacterAlive(charIdx) && !GetBit( characters[charIdx].charBits, (1<<1) ) && !GetBit( characters[charIdx].charBits, (1<<0) ) ) {
                        cout << "\n " << CharacterName( charIdx, false ) << ": " << CharacterDialogResponse( charIdx );
                        IncrementTurn();
                    }
                }
                else if ( word[2] == "me" || word[2] == "myself" ) {
                    cout << "\nNo one notices you are talking to yourself.";
                    IncrementTurn();
                }
                else {
                    cout << "\nThere is no one here by that name.";
                }
            }
            else {
                cout << "\nNo one notices you are talking to yourself.";
                IncrementTurn();
            }
        }
        else if ( word[0] == "go" || word[0] == "move" || word[0] == "walk" || word[0] == "run" ) {
            if ( word[1] == "to" ) {
                string* checkWords = new string[3];
                for ( int i=0; i<3; i++ ) {
                    checkWords[i] = word[(i+2)];
```

```cpp
            }
            int locIndex = DisambiguateLocation( checkWords );
            if ( locIndex > -1 ) {
                // location match (must have available exit)
                int playerLoc = GetNum( player.pBits, 13, 3 );
                int exitA = FindRoomFromExit( playerLoc, 0 );
                int exitB = FindRoomFromExit( playerLoc, 1 );
                int exitC = FindRoomFromExit( playerLoc, 2 );
                if ( exitA == locIndex || exitB == locIndex || exitC == locIndex ) {
                    player.pBits = SetNum( player.pBits, locIndex, 13, 3 );
                    IncrementTurn();
                }
                else {
                    cout << "\nYou cannot get there from here.\n";
                    cout << LocationDescriptionFormat( playerLoc );
                    IncrementTurn();
                }
            }
            else {
                cout << "\nThat is not a place you can go to. (Try 'look' to see the exits from here)";
            }
        }
        else {
            cout << "\nTry the command words 'go to' followed by an area, to move around.\n(or try 'help' for a list of commands)\n";
            cout << LocationDescriptionFormat( GetNum( player.pBits, 13, 3 ) );
        }
    }
    else if ( word[0] == "leave" || word[0] == "exit" ) {
        cout << "\nTry the command words 'go to' followed by an area, to move around.\n(or try 'help' for a list of commands)\n";
        cout << LocationDescriptionFormat( GetNum( player.pBits, 13, 3 ) );
    }
    else {
        // no match
        cout << "\nIt's not clear what you mean to do. (Try 'help' for a list of commands)";
        IncrementTurn();
    }
}

void ZCGame::IncrementTurn() {
    // (skip if game over)
    // if not released increment time to release
    // check time to release
    // if radio used and not S.O.S. call radio respond, flip S.O.S.
    // if S.O.S. called, increment turns
    // check for rescue time
    // if rescue time, call rescue arrived, flip rescue arrived
    // check if game over
    // Game data: Time to Release (0-7 turns), Is Released, Radio Used, S.O.S. Called, Rescue Arrived, Game Over
    // Time / Scoring data: Time To Rescue (since call for help, 0-31 turns), Score (0-31)
    if ( GetBit( gameBits, (1<<0) ) )
        return; // skip if game over
    if ( !GetBit( gameBits, (1<<4) ) ) {
        int timeToRelease = GetNum( gameBits, 5, 3 );
        if ( !GetBit( gameBits, (1<<4) ) ) {
            // not yet released
            if ( timeToRelease < 7 )
                gameBits = SetNum( gameBits, (timeToRelease+1), 5, 3 );
            else {
                // story signal release
                gameBits = SetBit( gameBits, true, (1<<4) );
                gameBits = SetNum( gameBits, 2, 5, 3 ); // infection timer reset to turn #2

                // GAME BALANCE NEED (re-start release timer if zombie is killed)
                int randChar = ( rand() % 8 );
                int safety = 0;
                // only alive non-infected and non-zombie character choice
                while ( !CharacterAlive(randChar) && safety < 8 && ( GetBit( characters[randChar].charBits, (1<<1) ) || GetBit( characters[randChar].charBits, (1<<0) ) ) ) {
                    safety++;
                    randChar = ( rand() % 8 );
                }
                if ( safety == 8 ) {
                    // without valid character selection, reset infection release timer
                    gameBits = SetBit( gameBits, false, (1<<4) );
                }
                else {
                    // random victim infected
                    InfectCharacter( randChar );
                }
            }
        }
    }
    else if ( GetBit( gameBits, (1<<3) ) ) {
        if ( !GetBit( gameBits, (1<<2) ) ) {
            // story radio respond
            gameBits = SetBit( gameBits, true, (1<<2) );
            if ( GetNum( player.pBits, 13, 3 ) == 0 ) {
                cout << "\n\n Coast Guard [Radio]: We're on our way, just hang tight. Over.\n";
            }
        }
        else {
            // increment time to rescue
            int timeToRescue = GetNum( scoreBits, 3, 5 );
            if ( timeToRescue < 31 )
                scoreBits = SetNum( scoreBits, (timeToRescue+1), 3, 5 );
            else {
                if ( !GetBit( gameBits, (1<<1) ) ) {
                    // story signal rescue arrived
                    gameBits = SetBit( gameBits, true, (1<<1) );
                    cout << "\n\n" << StoryFormat(32);
                }
                else {
                    // story signal game over
                    gameBits = SetBit( gameBits, true, (1<<0) );
                    cout << "\n\n" << StoryFormat(33);
                }
            }
        }
    }
}

void ZCGame::IncrementScore( int scoreAdd ) {
    int oldScore = GetNum( scoreBits, 0, 3 );
    scoreBits = SetNum( scoreBits, (oldScore+scoreAdd), 0, 3 );
}

bool ZCGame::IsDarkArea() {
    // is location inside, not on fire, and lights off?
    int playerLoc = GetNum( player.pBits, 13, 3 );
    int fireState = GetNum( locations[playerLoc].locBits, 4, 2 );
    return ( !GetBit( locations[playerLoc].locBits, (1<<7) ) && ( fireState != 2 )
&& ( playerLoc != 1 && playerLoc != 2 ) );
}

void ZCGame::LocationNotice() {
    int playerLoc = GetNum( player.pBits, 13, 3 );
    if ( IsDarkArea() ) {
        cout << LocationFormat( playerLoc ) << "\nThe lights are off and it is dark here.";
    }
    else if ( !GetBit( locations[playerLoc].locBits, (1<<6) ) ) {
        cout << LocationDescriptionFormat( playerLoc );
        locations[playerLoc].locBits = SetBit( locations[playerLoc].locBits, true, (1<<6) ); // now visited
    }
    else {
        cout << LocationFormat( playerLoc );
    }
}

void ZCGame::FireNotice() {
    int playerLoc = GetNum( player.pBits, 13, 3 );
    int fireState = GetNum( locations[playerLoc].locBits, 4, 2 );
    if ( fireState == 1 )
        cout << "\nThe area smells of flammable fumes.";
    else if ( fireState == 2 ) {
        int fireTime = GetNum( locations[playerLoc].locBits, 0, 4 );
        if ( fireTime < 6 )
            cout << "\nA large fire has started here.";
        else if ( fireTime > 5 && fireTime < 13 )
            cout << "\nThis area is engulfed in flames and smoke.";
        else
            cout << "\nThe fire in this area is now more smoke than fire.";
    }
    else if ( fireState == 3 && !IsDarkArea() )
        cout << "\nThis area has been ravaged by fire, and is thoroughly burnt.";
}

void ZCGame::ItemNotice() {
    if ( IsDarkArea() )
        return;
    int playerLoc = GetNum( player.pBits, 13, 3 );
    for (int i=0; i<12; i++) {
        int itemLoc = GetNum( items[i].itemBits, 2, 3 );
        int itemType = GetNum( items[i].itemBits, 5, 3 );
        if ( itemLoc == playerLoc && !GetBit( items[i].itemBits, (1<<1) ) ) {
            if ( !GetBit( items[i].itemBits, (1<<0) ) )
                cout << "\n" << ItemFormat( itemType );
            else {
                // used item
                if ( itemType == 0 )
                    cout << "\nThere is a spent flare gun is here.";
                else if ( itemType == 2 )
                    cout << "\nThere is a shattered glass bottle of alcohol is here.";
                else if ( itemType == 4 )
                    cout << "\nThere is the empty firing mechanism of a spear gun is here.";
                else if ( itemType == 7 )
                    cout << "\nThere is an empty fuel can is here.";
                else
                    cout << "\nThere is " << ItemFormat( itemType ) << " here ... used."; // should never be seen
            }
        }
    }
}
```

```cpp

void ZCGame::CharacterNotice() {
    if ( IsDarkArea() )
        return;
    int playerLoc = GetNum( player.pBits, 13, 3 );
    for (int i=0; i<8; i++) {
        int charLoc = GetNum( characters[i].charBits, 5, 3 );
        if ( charLoc == playerLoc ) {
            if ( CharacterAlive(i) ) {
                // alive characters
                cout << "\n" << CharacterName( i, ( GetBit( characters[i].charBits, (1<<0) ) && !GetBit( characters[i].charBits, (1<<1) ) ) ) << " is here.";
            }
            else {
                // corpses
                cout << "\nThe corpse of " << CharacterName( i, ( GetBit( characters[i].charBits, (1<<0) ) && !GetBit( characters[i].charBits, (1<<1) ) ) ) << " lies here.";
            }
        }
    }
}

void ZCGame::ZombieChatter() {
    int playerLoc = GetNum( player.pBits, 13, 3 );
    int zombieChatterCount = 0;
    for (int i=0; i<8; i++) {
        // only alive characters
        if ( !CharacterAlive(i) )
            continue;
        int charLoc = GetNum( characters[i].charBits, 5, 3 );
        // zombie, but not infected
        if ( charLoc == playerLoc && GetBit(characters[i].charBits, (1<<0)) && !GetBit(characters[i].charBits, (1<<1)) ) {
            // chance of zombie chatter
            if ( rand() % 2 == 0 ) {
                // additional newline before first zombie chatter
                if ( zombieChatterCount == 0 )
                    cout << "\n";
                cout << "\n " << CharacterName(i, true ) << ": " << CharacterDialogZombie(i);
                zombieChatterCount++;
            }
        }
    }
}

bool ZCGame::CharacterAlive( int charIndex ) {
    return ( GetNum( characters[charIndex].charBits, 2, 2 ) > 0 );
}

void ZCGame::CharacterChatter() {
    int playerLoc = GetNum( player.pBits, 13, 3 );
    int charCount = 0;
    int* charactersHere = new int[8];
    for (int i=0; i<8; i++) {
        charactersHere[i] = 0;
        // only alive characters
        if ( !CharacterAlive(i) )
            continue;
        int charLoc = GetNum( characters[i].charBits, 5, 3 );
        if ( charLoc == playerLoc && !GetBit(characters[i].charBits, (1<<1)) && !GetBit(characters[i].charBits, (1<<0)) ) {
            // only present alive non-infected non-zombie characters chatter
            charactersHere[charCount] = i;
            charCount++;
        }
    }
    if ( charCount > 1 ) {
        // chatter pick one initiate
        int charInitiate = charactersHere[( rand() % charCount )];
        cout << "\n\n " << CharacterName(charInitiate, false) << ": " << CharacterDialogQuestion( charInitiate );
        // pick a replier (not initiate)
        int charReplier = charInitiate;
        while (charReplier == charInitiate) {
            charReplier = charactersHere[( rand() % charCount )];
        }
        cout << "\n " << CharacterName(charReplier, false) << ": "  << CharacterDialogAnswer( charReplier );
    }
}

void ZCGame::IncrementStory() {
    // Present story based on timing and events
    bool isReleased = GetBit( gameBits, (1<<4) );
    bool gameOver = GetBit( gameBits, (1<<0) );
    int releaseCount = GetNum( gameBits, 5, 3 );
    int rescueCount = GetNum( scoreBits, 3, 5 );

    if ( gameOver ) {
        // story end
    }
    else if ( !isReleased ) {
        if ( releaseCount < 7 ) {
            // pre-release
            cout << "\n" << StoryFormat(releaseCount);
            if ( releaseCount == 0 ) {
                player.pBits = SetNum( player.pBits, 1, 13, 3 ); // on foredeck
                for ( int i=0; i<8; i++ ) {
                    characters[i].charBits = SetNum( characters[i].charBits, 1, 5, 3 ); // on foredeck
                }
            }
            else if ( releaseCount == 1 ) {
                player.pBits = SetNum( player.pBits, 3, 13, 3 ); // in ballroom
                for ( int i=0; i<8; i++ ) {
                    if ( i == 0 )
                        characters[i].charBits = SetNum( characters[i].charBits, 3, 5, 3 ); // capt in ballroom
                    else if ( i == 1 )
                        characters[i].charBits = SetNum( characters[i].charBits, 0, 5, 3 ); // 1st mate on bridge
                    else if ( i == 2 )
                        characters[i].charBits = SetNum( characters[i].charBits, 5, 5, 3 ); // chef in kitchen
                    else if ( i == 3 )
                        characters[i].charBits = SetNum( characters[i].charBits, 4, 5, 3 ); // phil in lounge
                    else
                        characters[i].charBits = SetNum( characters[i].charBits, 3, 5, 3 ); // rest in ballroom
                }
            }
            else if ( releaseCount == 2 ) {
                player.pBits = SetNum( player.pBits, 4, 13, 3 ); // in lounge
                for ( int i=0; i<8; i++ ) {
                    if ( i < 2 )
                        characters[i].charBits = SetNum( characters[i].charBits, 0, 5, 3 ); // capt and 1st mate on bridge
                    else if ( i == 2 )
                        characters[i].charBits = SetNum( characters[i].charBits, 5,
5, 3 ); // chef in kitchen
                    else
                        characters[i].charBits = SetNum( characters[i].charBits, 4, 5, 3 ); // rest in lounge
                }
            }
        }
    }
    else {
        if ( rescueCount == 0 ) {
            // story response
        }
        else if ( rescueCount < 31 ) {
            // waiting for rescue
        }
        else {
            if ( rescueCount == 31 ) {
                if ( !GetBit( gameBits, (1<<0) ) ) {
                    // story rescue arrived
                    cout << "\n" << StoryFormat(31);
                }
                else {
                    cout << "\n" << StoryFormat(32);
                }
            }
        }
    }
}

void ZCGame::IncrementInfection() {
    // Three-stages : infection but not zombie, both infection and zombie, then zombie but not infected
    for ( int i=0; i<8; i++ ) {
        if ( !CharacterAlive(i) )
            continue; // only alive characters
        if ( !GetBit( characters[i].charBits, (1<<0) ) && GetBit( characters[i].charBits, (1<<1) ) ) {
            // any character not zombie but infected becomes zombie
            if ( GetNum( player.pBits, 13, 3 ) == GetNum( characters[i].charBits, 5, 3 ) ) {
                // if in same room as player, perform resting dialog
                cout << "\n " << CharacterName( i, false ) << ": " << CharacterDialogResting( i );
            }
            characters[i].charBits = SetBit( characters[i].charBits, true, (1<<0) ); // zombie, starting next turn
            characters[i].charBits = SetBit( characters[i].charBits, true, (1<<4) ); // stay put
        }
        else if ( GetBit( characters[i].charBits, (1<<0) ) && GetBit( characters[i].charBits, (1<<1) ) ) {
            // character turns
            if ( GetNum( player.pBits, 13, 3 ) == GetNum( characters[i].charBits, 5, 3 ) ) {
                // if in same room as player, announce character turn
                cout << "\n" << CharacterName( i, false ) << " has turned zombie.";
            }
            characters[i].charBits = SetBit( characters[i].charBits, false, (1<<1) ); // no longer 'infected', but full zombie
            characters[i].charBits = SetBit( characters[i].charBits, true, (1<<4) ); // stay put
        }
    }
}

void ZCGame::IncrementFire() {
    // any area on fire, increment fire timer
    // if timer at end, fire out and room set to 'burnt'
    // if timer more than 7, chance of spread through exit to another room, if flammable
    for ( int i=0; i<8; i++ ) {
        if ( GetNum( locations[i].locBits, 4, 2 ) == 2 ) {
            // room on fire
            int fireTimer = GetNum( locations[i].locBits, 0, 4 );
            fireTimer++;
            if ( fireTimer >= 15 ) {
                fireTimer = 15;
                locations[i].locBits = SetNum( locations[i].locBits, 3, 4, 2 ); // area 'burnt'
            }
            locations[i].locBits = SetNum( locations[i].locBits, fireTimer, 0, 4 );
            int chanceOfSpread = ( rand() % 4 );
            if ( fireTimer < 15 && fireTimer > 6 && chanceOfSpread == 0 ) {
                int spreadFire = ( rand() % 3 );
                int fireExit = FindRoomFromExit( i, spreadFire );
                if ( GetNum( locations[fireExit].locBits, 4, 2 ) == 1 ) {
                    // fire spread to adjacent flammable room
                    locations[fireExit].locBits = SetNum( locations[fireExit].locBits, 2, 4, 2 ); // on fire
                    locations[fireExit].locBits = SetNum( locations[fireExit].locBits, 1, 0, 4 ); // fire timer reset
                    // if player in room, announce
                    if ( GetNum( player.pBits, 13, 3 ) == fireExit )
                        cout << "\nFire has spread into this room\n";
                }
            }
        }
    }
}

void ZCGame::FireDamage() {
    for (int i=0; i<8; i++) {
        int fireTime = GetNum( locations[i].locBits, 0, 4 );
        if ( GetNum( locations[i].locBits, 4, 2 ) == 2 && ( fireTime > 5 && fireTime < 13 ) ) {
            // fire damage knocks lights out
            if ( fireTime > 8 ) {
                if ( (rand() % 4) == 0 ) {
                    locations[i].locBits = SetBit( locations[i].locBits, false, (1<<7) );
                }
            }
            // player hurt by fire
            if ( GetNum( player.pBits, 13, 3 ) == i ) {
                int playerHealth = GetNum( player.pBits, 11, 2 );
                playerHealth--;
                if ( playerHealth <= 0 ) {
                    playerHealth = 0;
                    cout << "\n\n - You have died in the flames -";
                    gameBits = SetBit( gameBits, true, (1<<0) ); // game over
                }
                else {
                    cout << "\n\nYou have been burned by the flames.";
                }
                player.pBits = SetNum( player.pBits, playerHealth, 11, 2 );
            }
            // characters and zombies hurt by fire
            for (int n=0; n<8; n++) {
                if ( GetNum( characters[n].charBits, 5, 3 ) == i ) {
                    if ( CharacterAlive(n) ) {
                        // character is here and not dead
                        int charHealth = GetNum( characters[n].charBits, 2, 2 );
```

```cpp
                        charHealth--;
                        if ( charHealth <= 0 ) {
                            charHealth = 0;
                            // if player in room, announce char death
                            if ( GetNum(player.pBits, 13, 3) == i )
                                cout << "\n" << CharacterName( n, ( !GetBit( characters[n].charBits, (1<<1)) ) ) << " is killed by fire.";
                        }
                        else {
                            // if non-zombie and if player in room, char pain dialog
                            if ( GetNum(player.pBits, 13, 3) == i && ( GetBit( characters[n].charBits, (1<<1) ) || !GetBit( characters[n].charBits, (1<<0) ) ) )
                                cout << "\n " << CharacterName( n, ( !GetBit( characters[n].charBits, (1<<1) ) && GetBit(characters[n].charBits, (1<<0) ) ) ) << ": " << CharacterDialogExclamation( n );
                        }
                        characters[n].charBits = SetNum( characters[n].charBits, charHealth, 2, 2 );
                    }
                }
            }
        }
    }
}

void ZCGame::DoZombieMoves() {
    for (int i=0; i<8; i++) {
        // only alive characters
        if ( !CharacterAlive(i) )
            continue;
        // only zombies
        // zombie, but not infected
        if ( GetBit( characters[i].charBits, (1<<0) ) && !GetBit( characters[i].charBits, (1<<1) ) ) {
            int currRoom = GetNum( characters[i].charBits, 5, 3 );
            int newRoom = FindRoomFromExit( currRoom, ( rand() % 3 ) );
            // interrupt move if player is in same room
            // do move now if target room is not same as current (zombies move slow)
            //  set current room to target and return
            if ( !GetBit( characters[i].charBits, (1<<1) ) && currRoom != GetNum( player.pBits, 13, 3 ) ) {
                // notify player in same room that zombie is entering
                if ( newRoom == GetNum( player.pBits, 13, 3 ) && !IsDarkArea() )
                    cout << "\n\n" << CharacterName(i, true) << " arrives here.\n";
                characters[i].charBits = SetNum( characters[i].charBits, newRoom, 5, 3 );

                return;
            }
            bool charsInRoom = false;
            for (int n=0; n<8; n++) {
                if ( n != i && CharacterAlive(n) && currRoom == GetNum( characters[n].charBits, 5, 3 ) && !GetBit( characters[n].charBits, (1<<0) ) && !GetBit( characters[n].charBits, (1<<1) ) ) {
                    charsInRoom == true;
                    break;
                }
            }
            // if non-zombie / non-infected in room, stay (target room is same)
            // if player in room, stay
            // if no non-zombie in room, chance of move (un-set 'waiting' bit)
            if ( charsInRoom || currRoom == GetNum( player.pBits, 13, 3 ) ) {
                characters[i].charBits = SetBit( characters[i].charBits, true, (1<<4) );
            }
            else {
                characters[i].charBits = SetBit( characters[i].charBits, false, (1<<4) );
            }
        }
    }
}

void ZCGame::DoZombieAttacks() {
    for (int i=0; i<8; i++) {
        // only alive characters
        if ( !CharacterAlive(i) )
            continue;
        // only zombies
        // zombie, but not infected
        if ( GetBit( characters[i].charBits, (1<<0) ) && !GetBit( characters[i].charBits, (1<<1) ) ) {
            int currRoom = GetNum( characters[i].charBits, 5, 3 );
            int* allChars = new int[8];
            int charsInRoom = 0;
            for (int n=0; n<8; n++) {
                allChars[charsInRoom] = 0;
                // only alive characters
                if ( !CharacterAlive(n) )
                    continue;
                if ( n != i && currRoom == GetNum( characters[n].charBits, 5, 3 ) && !GetBit( characters[n].charBits, (1<<0) ) && !GetBit( characters[n].charBits, (1<<1) ) ) {
                    allChars[charsInRoom] = n;
                    charsInRoom++;
                }
            }
            // if non-zombie / non-infected in room, attack
            // if player in room, and not dead, attack
            if ( charsInRoom > 0 || ( currRoom == GetNum( player.pBits, 13, 3 ) && GetNum(player.pBits, 11, 2 ) > 0 ) ) {
                if ( currRoom == GetNum( player.pBits, 13, 3 ) && GetNum(player.pBits, 11, 2 ) > 0 ) {
                    // zombie attack player
                    cout << "\n" << CharacterName( i, true ) << " attacks you";
                    // if nothing equipped, 75% chance lose health
                    // if one item equipped, 50% chance lose health
                    // if two items equipped, 25% chance lose health
                    bool loseHealth = true;
                    int combatRoll = ( rand() % 4 );
                    if ( combatRoll == 0 )
                        loseHealth = false;
                    if ( GetNum(player.pBits, 7, 4 ) != 15 && combatRoll < 2 )
                        loseHealth = false;
                    if ( GetNum(player.pBits, 3, 4 ) != 15 && combatRoll < 3 )
                        loseHealth = false;
                    if ( loseHealth ) {
                        int pHealth = GetNum( player.pBits, 11, 2 );
                        pHealth -= 1;
                        player.pBits = SetNum( player.pBits, pHealth, 11, 2 );
                        if ( pHealth < 1 ) {
                            // player dies
                            cout << "\n\n - YOU HAVE TURNED ZOMBIE -";
                            gameBits = SetBit( gameBits, true, (1<<0) );
                        }
                        else {
                            cout << "\n - YOU ARE HURT -";
                        }
                    }
                    else {
                        cout << " and misses.";
```

```cpp
                    }
                }
                else {
                    // infect random character
                    int infectedChar = allChars[( rand() % charsInRoom )];
                    characters[infectedChar].charBits = SetBit( characters[infectedChar].charBits, true, (1<<1) );
                    // reduce character health by one (set to 2)
                    characters[infectedChar].charBits = SetNum( characters[infectedChar].charBits, 2, 2, 3 );
                    // character performs exclamation
                    if ( currRoom == GetNum( player.pBits, 13, 3 ) )
                        cout << "\n " << CharacterName( infectedChar, false ) << ": " << CharacterDialogExclamation( infectedChar );
                }
            }
        }
    }
}

void ZCGame::DoCharacterMoves() {
    bool choseToExit = ( rand() % 6 == 0 );
    int chosenExitIndex = ( rand() % 3 );
    for (int i=0; i<8; i++) {
        // only alive characters
        if ( !CharacterAlive(i) )
            continue;
        // only non-zombies, set target room
        if ( !GetBit( characters[i].charBits, (1<<0) ) ) {
            // character moves
            int currRoom = GetNum( characters[i].charBits, 5, 3 );
            int targetRoom = currRoom;
            int chosenExit = FindRoomFromExit( currRoom, chosenExitIndex );
            // if infected, stay
            if ( GetBit( characters[i].charBits, (1<<1) ) ) {
                characters[i].charBits = SetBit( characters[i].charBits, true, (1<<4) );
            }
            else {
                bool zombieInRoom = false;
                bool otherInRoom = false;
                for (int n=0; n<8; n++) {
                    // only alive, and not self
                    if ( n != i && CharacterAlive(n) && GetNum( characters[n].charBits, 5, 3 ) == currRoom ) {
                        // zombie, but not infected
                        if ( GetBit( characters[n].charBits, (1<<0) ) && !GetBit( characters[n].charBits, (1<<1) ) ) {
                            zombieInRoom = true;
                        }
                        else
                            otherInRoom = true;
                    }
                }
                if ( GetNum( locations[currRoom].locBits, 4, 2 ) == 2 ) {
                    // if current room on fire, move
                    int panicExit = FindRoomFromExit( currRoom, ( rand() % 3 ) );
                    targetRoom = panicExit;
                }
                else if ( zombieInRoom ) {
                    // if zombie in room, move
                    int panicExit = FindRoomFromExit( currRoom, ( rand() % 3 ) );
                    targetRoom = panicExit;
                }
                else if ( GetNum( player.pBits, 13, 3 ) == currRoom ) {
                    // if player in room, stay
                    characters[i].charBits = SetBit( characters[i].charBits, true, (1<<4) );
                }
                else if ( !otherInRoom ) {
                    // if alone, chance of move through any exit
                    if ( rand() % 4 == 0 ) {
                        int panicExit = FindRoomFromExit( currRoom, ( rand() % 3 ) );
                        targetRoom = panicExit;
                    }
                }
                else {
                    if ( ( rand() % 8 ) == 0 ) {
                        // if not alone, very rare chance of split up
                        int panicExit = FindRoomFromExit( currRoom, ( rand() %3 ) );
                        targetRoom = panicExit;
                    }
                    else if ( choseToExit ) {
                        // if not alone, rare chance of move through same exit
                        targetRoom = chosenExit;
                    }
                }
            }
            // do move now (characters move fast)
            if ( targetRoom != currRoom ) {
                if ( currRoom == GetNum( player.pBits, 13, 3 ) && !IsDarkArea() )
                    cout << "\n" << CharacterName( i, false ) << " leaves.";
                // if target room != current room, set current to target
                characters[i].charBits = SetNum( characters[i].charBits, targetRoom, 5, 3 );
                if ( targetRoom == GetNum( player.pBits, 13, 3 ) && !IsDarkArea() )
                    cout << "\n" << CharacterName( i, false ) << " arrives here.";
            }
        }
    }
}

void ZCGame::DoPlayerAttack() {
    // find all zombies in same room, pick one
    int numZombies = 0;
    int* zombiesInRoom = new int[8];
    for (int i=0; i<8; i++) {
        zombiesInRoom[numZombies] = 0;
        // only alive characters that are in room
        if ( !CharacterAlive(i) || GetNum( characters[i].charBits, 5, 3 ) != GetNum( player.pBits, 13, 3 ) )
            continue;
        // zombie, but not infected
        if ( GetBit( characters[i].charBits, (1<<0) ) && !GetBit( characters[i].charBits, (1<<1) ) ) {
            zombiesInRoom[numZombies] = i;
            numZombies++;
        }
    }
    if ( numZombies == 0 ) {
        cout << "\nThere is no threat to attack here.";
    }
    else {
        int targetZombie = zombiesInRoom[( rand() % numZombies )];
        int combatRoll = ( rand() % 4 );
        int damageDone = 0;
        // if equipped, use either equipped item as weapon
        int equipA = GetNum( player.pBits, 7, 4 );
        int equipB = GetNum( player.pBits, 3, 4 );
        int weaponIndex = 15;
```

```cpp
            if ( equipA != 15 && equipB != 15 ) {
                if ( rand() % 2 == 0 )
                    weaponIndex = equipA;
                else
                    weaponIndex = equipB;
            }
            else if ( equipA != 15 )
                weaponIndex = equipA;
            else if ( equipB != 15 )
                weaponIndex = equipB;
            // otherwise, no damage
            if ( weaponIndex != 15 ) {
                // find item weapon type
                int weaponType = GetNum( items[weaponIndex].itemBits, 5, 3 );
                switch (weaponType) {
                case 0:
                    damageDone = 3;
                    break;
                case 1:
                    damageDone = 2;
                    break;
                case 2:
                    damageDone = 1;
                    break;
                case 3:
                    damageDone = 2;
                    break;
                case 4:
                    damageDone = 3;
                    break;
                case 5:
                    damageDone = 2;
                    break;
                case 6:
                    damageDone = 2;
                    break;
                case 7:
                    damageDone = 1;
                    break;
                }
                if ( combatRoll < 3 && damageDone > 0 ) {
                    // hit, reduce zombie health
                    int zombieHealth = GetNum( characters[targetZombie].charBits, 2, 2 );
                    zombieHealth -= damageDone;
                    if ( zombieHealth <= 0 ) {
                        zombieHealth = 0;
                        // zombie killed
                        cout << "\n - " << CharacterName( targetZombie, true ) << " is killed with " << ItemName(weaponType) << " -";
                        // GAME BALANCE NEED
                        int numZombiesOnboard = 0;
                        int numCharsAlive = 0;
                        for ( int i=0; i<8; i++ ) {
                            if ( GetBit( characters[i].charBits, (1<<0) ) ) {
                                numZombiesOnboard++;
                            }
                            else {
                                numCharsAlive++;
                            }
                            if ( numCharsAlive > 0 && numZombiesOnboard == 0 ) {
                                gameBits = SetBit( gameBits, false, (1<<4) ); // reset infection release timer
                            }
                            else {
                                // no chars left to infect, player is alone onboard
                            }
                        }
                    }
                    else
                        cout << "\nYou hit " << CharacterName( targetZombie, true ) << " with " << ItemName(weaponType) << " for " << damageDone << " damage.";
                    characters[targetZombie].charBits = SetNum( characters[targetZombie].charBits, zombieHealth, 2, 2 );
                    // use item
                    if ( UseItem( weaponIndex ) ) {
                        if ( weaponIndex != 1 ) {
                            // one-use, so drop item
                            DropItem( weaponIndex );
                            cout << "\n";
                        }
                    }
                }
                else {
                    cout << "\nYour attempt to hit " << CharacterName( targetZombie, true ) << " missed.";
                    if ( weaponIndex == 15 )
                        cout << "\n[HINT] Equip yourself with items you take to be more effective in combat.\n";
                    else
                        cout << "\n";
                }
            }
            else {
                cout << "\nYour unarmed attack is ineffective against the undead.";
                cout << "\n[HINT] Equip yourself with items you take to be more effective in combat.\n";
            }
        }
    }

    void Pause( int seconds ) {
        int pTime = time(NULL);
        while (time(NULL)<(pTime+seconds)) {
            // ... waiting
        }
    }

    int main() {

        // PROTOTYPE TESTING
        ZCGame thisGame;
        // SEED RANDOM
        srand(time(NULL));
        // INITIALIZE GAME
        thisGame.Initialize();

        thisGame.player.pBits = SetNum( thisGame.player.pBits, 4, 13, 3 );
        thisGame.player.pBits = SetNum( thisGame.player.pBits, 3, 11, 2 ); // location=4, health=3
        thisGame.player.pBits = SetNum( thisGame.player.pBits, 15, 7, 4 ); // equip Item A = none
        thisGame.player.pBits = SetNum( thisGame.player.pBits, 15, 3, 4 ); // equip Item B = none

        cout << "\n [ ZOMBIE CRUISE . a text adventure by Glenn Storm ]\n\n";

        Pause(3);

        cout << HelpFormat();

        Pause(4);

        string testStr = "debug";

        while ( testStr != "quit" && !GetBit( thisGame.gameBits, (1<<0) ) ) {
            // story
            thisGame.IncrementStory();
            // player view
            thisGame.LocationNotice();
            thisGame.FireNotice();
            thisGame.IncrementFire();
            thisGame.ItemNotice();
            thisGame.CharacterNotice();
            // chatter
            thisGame.ZombieChatter();
            thisGame.CharacterChatter();
            // zombie attacks
            thisGame.DoZombieAttacks();
            // zombie moves
            thisGame.DoZombieMoves();
            // infection progress
            thisGame.IncrementInfection();
            // character moves
            thisGame.DoCharacterMoves();
            // fire damage
            thisGame.FireDamage();
            cout << "\n\n";
            Pause(1);
            if ( !GetBit( thisGame.gameBits, (1<<0) ) ) { // game not over
                // player input
                testStr = PlayerPrompt();
                thisGame.ParseMove(testStr);
                cout << "\n";
            }
            else {
                cout << " . GAME OVER .\n";
                Pause(3);
            }
        }

        // GAME LOOP

        // Game update
        // Story update
        // Fire update
        // Character update
        // Zombie update
        // Player update
        // Score update

        // Story format
        // Location format
        // Dynamics (lights/fire) format
        // Item format
        // Character format
        // Zombie format

        // Dialog format

        // Present story
        // Present location
        // Present dynamics
        // Present items
        // Present characters
        // Present zombies
        // Present dialog

```

```cpp
    // Player prompt

    // Game reaction (quit)
    // Help reaction
    // Move reaction
    // Talk reaction
    // Look reaction
    // Use reaction
    // Equip reaction

    // Inventory management

    // Combat response
    // Item response
    // Character response
    // Zombie response
    // Fire response
    // Lighting response

    // Radio response
    // Location-specific response

    // END GAME LOOP

    // TEMP scoring
    int numChars = 0;
    int numZombies = 0;
    for (int i=0; i<8; i++) {
        // only alive characters
        if ( !thisGame.CharacterAlive(i) )
            continue;
        if ( GetBit( thisGame.characters[i].charBits, (1<<0) ) )
            numZombies++;
        else
            numChars++;
    }
    if ( GetNum( thisGame.player.pBits, 11, 2 ) < 1 )
        numZombies++; // player is zombie if dead
    else
        numChars++; // player is surviving character if alive
    cout << "\n\n TOTAL ZOMBIES ONBOARD: " << numZombies;
    cout << "\n TOTAL SURVIVORS ONBOARD: " << numChars;
    int score = 0;
    if ( GetBit( thisGame.gameBits, (1<<2) ) ) {
        score += 1; // SOS called +1
    }
    if ( GetBit( thisGame.gameBits, (1<<1) ) ) {
        score += 1; // Rescue arrived +1
        if ( GetNum( thisGame.player.pBits, 11, 2 ) > 0 ) {
            score += 1; // Player alive
        }
        if ( numChars > 3 ) {
            score += 1; // Characters outnumber zombies
        }
        if ( numZombies < 4 ) {
            score += 3; // Zombies suppressed
        }
    }
    cout << "\n\n  FINAL SCORE: " << score << "/7\n" << "  RANK: ";
    // Score Ranks: <3 "Zombie Meat", 3-4 "Survivor", 5-6 "Hero", 7 "Zombie Killer"
    if ( score < 3 )
        cout << "Zombie Meat";
    else if ( score < 5 )
        cout << "Survivor";
    else if ( score < 7 )
        cout << "Hero";
    else
        cout << "Zombie Killer";
    cout << "\n";

    Pause(5);

    string e;
    cout << "\n [PRESS ENTER KEY TO END]\n";
    getline(cin, e);

    return 0;
};
```