1. Hunt the Wumpus Intro. The purpose of this CWeb program is to play the oldie but a goldie game **Hunt the Wumpus.**

— **This is a program written in cpp and requires the g++ compiler.**

This game has 4 major factors:

- 2 x Bats - that can transport you
- 2 x Pits - that can kill you
- A Wumpus - that can eat you which means kill
- You and your arrows...good luck :)

All of the above mentioned are chucked into a game loop. That whilst there are still arrows left in your inventory you can play.

2. Program skeleton time.

```
  ⟨ Includes 3 ⟩
  ⟨ Functions Initialise 4 ⟩
  ⟨ Main 5 ⟩
  ⟨ Functions Definiton 12 ⟩
```

3. For c++, std library is a no no but we will use strings for output and rand numbers

⟨ Includes 3 ⟩ ≡

```
#include <iostream>    /* standard io */
#include <string>    /* strings */
#include <stdlib.h>    /* srand, rand */
#include <time.h>    /* time */
#include <iomanip>    /* output formatting */
  using namespace std;
```

This code is used in section 2.

4. Functions Initialise. will be used in this program for readability and reuseability.

⟨ Functions Initialise 4 ⟩ ≡

```
int withDanger(int &player, int &wumpus, int bats[2], int pits[2]);
    /* check if you're in the same cave as dangers */
int closeDanger(int wumpus, int bats[2], int pits[2], int closeRooms[3]);
    /* checks for dangers in nearby rooms */
int turn(int player, int closeRooms[3], int roomsOrder[20], int &wumpus, int &arrows, int
        roomsIndex[20][3], int roomsOrderBack[20]);    /* users turn for M or S */
void shoot(int player, int wumpus, int roomsIndex[20][3], int roomsOrderBack[20]);
    /* maps the path for the arrow to shoot */
int wumpusMove(int wumpus, int roomsIndex[20][3]);    /* returns the wumpus location */
void instructions();    /* shows the instructions */
void shuffle(int(&arrayX)[20], int(&arrayY)[20]);    /* shuffles the array sent to it */
void setDangers(int &wumpus, int(&arrayBats)[2], int(&arrayPits)[2], int(&save)[5]);
    /* sets the Dangers Index */
void reset(int &player, int &wumpus, int(&arrayBats)[2], int(&arrayPits)[2], int(&save)[5], int
        arrows);    /* reset the game if the user falls into a pit */
bool replay();    /* asks user if they want to replay */
```

This code is used in section 2.

5. Main Function. Now for the Main function.

```
⟨ Main 5 ⟩ ≡
  int main( )
  {
    ⟨ Varaibles of main 6 ⟩
    ⟨ set The Game 7 ⟩
    ⟨ display Instructions 10 ⟩
    ⟨ Game Play 11 ⟩
    return 0;
  }
```

This code is used in section 2.

6. Variables will include key game play features such as:

- The player who always starts at in postion 1
- The Wumpus who starts at anything but 1
- The bats who aren't with Wumpus or position 1
- The pit who aren't with Wumpus or position 1

```
⟨ Varaibles of main 6 ⟩ ≡
  int player = 0;
  int wumpus, bats[2], pits[2];
  int arrows = 5;    /* the number of arrows a player has in total */
  char reply;    /* the reply to yes or no's or M or S */
  int roomsOrder[20], roomsOrderBack[20];
  int closeRooms[3];
  int save[5];
  int roomsIndex[20][3] = {{4, 7, 1}, {0, 9, 2}, {1, 11, 3}, {2, 13, 4}, {3, 5, 0}, {14, 4, 6}, {5, 16, 7}, {6, 0, 8}, {7,
      17, 9}, {8, 1, 10}, {9, 18, 11}, {10, 2, 12}, {11, 19, 13}, {12, 3, 14}, {13, 15, 5}, {19, 14, 16}, {15, 6, 7}, {16,
      8, 18}, {17, 10, 19}, {18, 12, 15}};
```

This code is used in section 5.

7. Set the game. This section is about setting the values for the game play.

```
⟨ set The Game 7 ⟩ ≡
  ⟨ shuffle Arrays 8 ⟩
  ⟨ set Dangers 9 ⟩
```

This code is used in section 5.

8. Shuffle arrays. Shuffling the array *Rooms Order* allows for a new map to be created for every game play. The shuffle I use is fisher yates because it's quick and simple. The shuffled array *roomsOrder* should only be used for outputting to the user. Another array called *roomsOrderBack* is used as a index reference. This is done so no searching or fiddling has be done in order to find the rooms connections in the *roomsIndex* array.

```
⟨ shuffle Arrays 8 ⟩ ≡
  shuffle(roomsOrder, roomsOrderBack);
```

This code is used in section 7.

9. Set Dangers. This will set the dangers such as the *wumpus*, *bats* or *pits* around the map. Every time the game is refreshed the dangers will move to another position.

```
⟨ set Dangers 9 ⟩ ≡
  setDangers(wumpus, bats, pits, save);
```

This code is used in section 7.

10. Instructions. **Hunt the Wumpus** has this thing about instructions. Honestly I think games should just be played with trial and error :P. Following this the user is asked if they want to read the instructions first. How boring...

```
⟨ display Instructions 10 ⟩ ≡
  cout ≪ "Instructions␣(Y␣|␣N)␣:␣";
  cin ≫ reply;
  if (reply ≡ 'Y' ∨ reply ≡ 'y') {
    instructions();
  }
  else {
    cout ≪ "You're␣in␣for␣a␣mistake" ≪ endl;
  }
```

This code is used in section 5.

11. Game Play. This is the actual loop of the game.

```
⟨ Game Play 11 ⟩ ≡
  cout ≪ "\n␣---␣HUNT␣DAT␣WUMPUS␣---␣\n" ≪ endl;
  while (arrows > 0) {
    int wD = withDanger(player, wumpus, bats, pits);
    if (wD ≡ 1) {
      wumpus = wumpusMove(wumpus, roomsIndex);
      if (wumpus ≡ player) {
        cout ≪ "---␣GAME␣OVER!␣THE␣WUMPUS␣GOT␣YOU␣---" ≪ endl;
        if (replay()) {
          reset(player, wumpus, bats, pits, save, arrows);
        }
      }
    }
    else if (wD ≡ 2) {
      if (replay()) {
        reset(player, wumpus, bats, pits, save, arrows);
      }
    }
    for (int i = 0; i < 3; i++) {
      closeRooms[i] = roomsIndex[player][i];
    }
    closeDanger(wumpus, bats, pits, closeRooms);
    player = roomsOrderBack[turn(player, closeRooms, roomsOrder, wumpus, arrows, roomsIndex,
        roomsOrderBack)];
  }
  cout ≪ "You␣are␣out␣of␣arrows...rip" ≪ endl;
  if (replay()) {
    reset(player, wumpus, bats, pits, save, arrows);
  }
```

This code is used in section 5.

12. Functions Definitions. This is the section where I define what the functions actually do and how they handle the information circulating around the wumpus game.

```
⟨ Functions Definiton 12 ⟩ ≡
   ⟨ shuffle Function 13 ⟩
   ⟨ setDangers Function 14 ⟩
   ⟨ checkDangers Functions 15 ⟩
   ⟨ wumpusMove Function 16 ⟩
   ⟨ turn Functions 17 ⟩
   ⟨ resetGame Functions 18 ⟩
   ⟨ instructions Function 19 ⟩
```

This code is used in section 2.

13. Shuffle. This function is a void type and will manipulate the sent in arrays by reference. *arrayX* is the *roomsOrder* int array and *arrayY* is the *roomsOrderBack* int array.

```
⟨ shuffle Function 13 ⟩ ≡
   void shuffle(int (&arrayX)[20], int (&arrayY)[20])
   {
      int j, t;
      srand(time(Λ));
      for (int i = 0; i < 20; i++) {
         arrayX[i] = i;
      }
      for (int i = 19; i > −1; i−−) {
         j = rand() % (i + 1);
         t = arrayX[j];
         arrayX[j] = arrayX[i];
         arrayX[i] = t;     /* backwards array recoding */
         arrayY[arrayX[i]] = i;
      }
   }
```

This code is used in section 12.

14. Set Dangers. Using srand and rand to randomly place the dangers of the caves around the place. This will allow new maps for every game play. This function uses *wumpus*, *bats*, *pits*. Int array *save* is for when a player falls into a pit and request to restart the same map. This will just save the positions so the function 'reset' can get it going again.

⟨ setDangers Function 14 ⟩ ≡

```
void setDangers(int &wumpus, int (&arrayBats)[2], int (&arrayPits)[2], int (&save)[5])
{
  int v = 19;
  for (int i = 0; i < 5; i++) {
    save[i] = rand() % v + 1;
    v--;
  }
  srand(time(Λ));
  wumpus = save[0];
  arrayBats[0] = save[1];
  arrayBats[1] = save[2];
  arrayPits[0] = save[3];
  arrayPits[1] = save[4];
}
```

This code is used in section 12.

15. Check Dangers. These two functions are about checking the close dangers or if you're with a danger. IF you are with a danger like pits or bats the game ends. *WithDanger* will send back 1 if you are in a room with a Wumpus. *WithDanger* will send back 2 if you are in a room with a Pits. *WithDanger* will send back 0 if you are safe. *closeDanger* will output if you are close to a danger by looping through the nearby rooms. IF you are next to a danger the game outputs a warning.

⟨ checkDangers Functions 15 ⟩ ≡

```
int withDanger(int &player, int &wumpus, int bats[2], int pits[2])
{
  if (player ≡ wumpus) {
    return 1;
  }
  for (int i = 0; i < 2; i++) {
    if (player ≡ bats[i]) {
      cout ≪ "Oh!!␣the␣bats␣got␣you!!␣Where␣will␣you␣go?!" ≪ endl;
      srand(time(Λ));
      player = rand() % 20;
    }
    if (player ≡ pits[i]) {
      cout ≪ "AHHHHHHH␣you␣fell␣into␣a␣pit" ≪ endl;
      cout ≪ "HA␣HA␣HA␣-␣YOU␣LOSE!" ≪ endl;
      return 2;
    }
  }
  return 0;
}
void closeDanger(int wumpus, int bats[2], int pits[2], int closeRooms[3])
{
  for (int i = 0; i < 3; i++) {
    if (wumpus ≡ closeRooms[i]) {
      cout ≪ "I␣smell␣a␣wumpus..." ≪ endl;
    }
    for (int j = 0; j < 2; j++) {
      if (bats[j] ≡ closeRooms[i]) {
        cout ≪ "Bats␣nearby" ≪ endl;
      }
      if (pits[j] ≡ closeRooms[i]) {
        cout ≪ "I␣feel␣a␣draft" ≪ endl;
      }
    }
  }
}
```

This code is used in section 12.

16. Wumpus Move. This function is for when a player either walks into the wumpus cave and the wumpus has to move. Or an arrow flies past the cave and the wumpus has to move. 7525

⟨ wumpusMove Function 16 ⟩ ≡

```
int wumpusMove(int wumpus, int roomsIndex[20][3])
{
  int closeRooms[3];
  srand(time(Λ));
  int i = rand() % 4;
  if (i ≡ 4) {
    return wumpus;
  }
  else {
    wumpus = roomsIndex[wumpus][i];
  }
  return wumpus;
}
```

This code is used in section 12.

17. Turn. These functions are about the players turn to either move or shoot. It will return the room the user wants to move to. It will list the closest rooms with *closeRooms*.
'void shoot' will map the direction of the arrow and move the wumpus. if need be e.g. the arrow passes his room. First it will ask the user where they want to shoot to. If the path they chose isn't real the arrows goes random. If the arrows passes by the wumpus's room, the wumpus can move or stay. The first loop determines the most of the real path. The second determines the random if the real does not exist. The random path does not allow for the arrow to circle the way it came. It does this by remebering the previous location in *prev* and if the rand chooses there again it switches it to + 1. rand is only allowed to choose between 0-2;

⟨ turn Functions 17 ⟩ ≡

```
int turn(int player, int closeRooms[3], int roomsOrder[20], int &wumpus, int &arrows, int
        roomsIndex[20][3], int roomsOrderBack[20])
{
  char reply;
  int roomTo = 0;
  int playerRoom = roomsOrder[player];
  cout ≪ "You␣are␣in␣room:␣" ≪ playerRoom ≪ endl;
  cout ≪ "Tunnels␣lead␣to:␣";
  for (int i = 0; i < 3; i++) {
    cout ≪ roomsOrder[closeRooms[i]] ≪ "␣␣␣";
  }
  cout ≪ endl;
  cout ≪ "\nMove␣or␣Shoot␣(M|S):␣";
  cin ≫ reply;
  if (reply ≡ 'M' ∨ reply ≡ 'm') {
    cout ≪ "Where␣to?:␣";
    cin ≫ roomTo;
    cout ≪ endl;
    for (int i = 0; i < 3; i++) {
      if (roomTo ≡ roomsOrder[closeRooms[i]]) {
        return roomTo;
      }
    }
  }
  else if (reply ≡ 'S' ∨ reply ≡ 's') {
    arrows−−;
    shoot(player, wumpus, roomsIndex, roomsOrderBack);
  }
  else {
    cout ≪ "Bad␣input,␣try␣again:␣" ≪ reply ≪ endl;
  }
  cout ≪ endl;
  return playerRoom;
}
void shoot(int player, int wumpus, int roomsIndex[20][3], int roomsOrderBack[20])
{
  int rooms[5] = {0};
  int reply, num, prev = −1;
  int arrowsIndex = player;
  bool okay = true;
  int room;
```

```
    cout ≪ "No.␣of␣Rooms?:␣";
    cin ≫ num;
    cout ≪ "Rooms#?␣:␣";
    for (int i = 0; i < num; i++) {
      cin ≫ reply;
      rooms[i] = roomsOrderBack[reply];
    }
    int i = 0;
    while (okay ∧ i < num) {
      for (int j = 0; j < 3; j++) {
        if (rooms[i] ≡ roomsIndex[arrowsIndex][j]) {
          okay = true;
          arrowsIndex = rooms[i];
          break;
        }
      }
      okay = false;
      i++;
      if (arrowsIndex ≡ wumpus ∧ i < num) {
        wumpus = wumpusMove(wumpus, roomsIndex);
      }
    }
    prev = arrowsIndex;
    if (¬okay) {
      for (int p = i; p < num; p++) {
        srand(time(Λ));
        room = rand() % 2;
        if (roomsIndex[arrowsIndex][room] ≠ prev) {
          arrowsIndex = roomsIndex[arrowsIndex][room];
        }
        else {
          arrowsIndex = roomsIndex[arrowsIndex][room + 1];
        }
        prev = arrowsIndex;
        if (arrowsIndex ≡ wumpus ∧ p ≠ num) {
          wumpus = wumpusMove(wumpus, roomsIndex);
        }
      }
    }
    if (arrowsIndex ≡ wumpus) {
      cout ≪ "AHA!␣YOU␣GOT␣THE␣WUMPUS!" ≪ endl;
      cout ≪ "!!!THE␣WUMPUSLL␣GETCHA␣NEXT␣TIME!!!" ≪ endl;
      exit(0);
    }
    if (arrowsIndex ≡ player) {
      cout ≪ "Shot␣your␣own␣foot␣there...oops" ≪ endl;
      exit(0);
    }
  }
```

This code is used in section 12.

18. Reset Game. These functions are for the resetting of the game if the player dies. 'replay' will ask if they want to replay the game on a new or the exiting map. If false then the game is new else reset the game to the original. This will be done by calling the setDangers function and resetting *player* to index 0.
'reset' will reset the player if they choose to replay the exisitng map.

```
⟨ resetGame Functions 18 ⟩ ≡
  bool replay( )
  {
    char reply;
    cout ≪ "__________________________________" ≪ endl;
    cout ≪ "\nRestart␣-␣Same␣setup?␣(Y|N)␣:␣";
    cin ≫ reply;
    if (reply ≡ 'Y' ∨ reply ≡ 'y') {
      return true;
    }
    else {
      return false;
    }
  }
  void reset(int &player, int &wumpus, int(&arrayBats)[2], int(&arrayPits)[2], int(&save)[5], int
          arrows)
  {
    player = 0;
    wumpus = save[0];
    arrayBats[0] = save[1];
    arrayBats[1] = save[2];
    arrayPits[0] = save[3];
    arrayPits[1] = save[4];
    arrows = 5;
  }
```

This code is used in section 12.

19. Instructions. This is a **void** type and will simply show a formated version of the instructions to the player when they say y or Y. The game after this should just start.

```
⟨ instructions Function 19 ⟩ ≡
  void instructions( )
  {
    cout ≪ "\n␣---␣WELCOME␣TO␣HUNT␣THE␣WUMPUS␣---␣\n␣" ≪ endl;
    cout  ≪  "THE␣WUMPUS␣LIVES␣IN␣A␣CAVE␣OF␣␣20␣ROOMS."  ≪
        "EACH␣ROOM␣HAS␣3␣TUNNELS␣LEADING␣TO␣OTHER␣ROOMS"  ≪
        "(LOOK␣AT␣A␣DODECAHEDRON␣IS,␣ASK␣SOMEONE)␣\n"  ≪  "\nHAZARDS:␣\n"  ≪
        "␣␣␣␣␣BOTTOMLESS␣PITS␣-␣"  ≪  "TWO␣ROOMS␣HAVE␣BOTTOMS␣LESS␣PITS␣IN␣THEM."  ≪
        "IF␣YOU␣GO␣THERE,␣YOU␣FALL␣INTO␣THE␣PIT␣(AND␣LOSE!)␣\n"  ≪
        "␣␣␣␣␣SUPER␣BATS␣-␣"  ≪  "TWO␣OTHER␣ROOMS␣HAVE␣SUPER␣BATS."  ≪
        "IF␣YOU␣GO␣THERE,␣A␣BAT␣GRABS␣YOU␣AND␣TAKES␣YOU␣TO␣SOME␣OTHER␣ROOM␣AT␣RANDOM.\n" ≪
        "\nWUMPUS:␣\n"  ≪  "␣␣␣␣␣THE␣WUMPUS␣IS␣NOT␣BOTHERED␣BY␣THE␣\
        HAZARDS"  ≪  "(HE␣HAS␣SUCKER␣FEET␣AND␣IS␣TOO␣FAT)"  ≪
        "USUALLY␣HE␣IS␣ASLEEP."  ≪  "TWO␣THINGS␣WAKE␣HIM␣UP:␣"  ≪
        "YOU'RE␣ENTERING␣HIS␣ROOM␣OR␣YOU'RE␣SHOOTING␣AN␣ARROW.␣\n\n"  ≪
        "IF␣THE␣WUMPUS␣WAKES,␣HE␣SOMETIMES␣RUNS␣TO␣THE␣NEXT␣ROOM.\n"  ≪
        "IF␣YOU␣HAPPEN␣TO␣BE␣IN␣THE␣SAME␣ROOM␣WITH␣HIM,␣YOU␣LOSE.\n"  ≪  "\nYOU:␣\n"  ≪
        "EACH␣TURN␣YOU␣MAY␣MOVE␣OR␣SHOOT␣A␣CROOKED␣ARROW.\n"  ≪  "␣␣␣␣␣MOVE:␣YOU␣CAN␣\
        MOVE␣ONE␣ROOM␣(THROUGH␣ONE␣TUNNEL.)␣\n"  ≪  "␣␣␣␣␣SHOOT:␣YOU␣HAVE␣5␣ARROWS."  ≪
        "YOU␣LOSE␣WHEN␣YOU␣RUN␣OUT."  ≪  "EACH␣ARROW␣CAN␣GO␣FROM␣1␣TO␣5␣ROOMS."  ≪
        "YOU␣AIM␣BY␣TELLING␣THE␣COMPUTER␣THE␣#'S␣YOU␣WANT␣THE␣ARROW␣TO␣GO␣TO."  ≪
        "IF␣THE␣ARROW␣CAN'T␣GO␣THAT␣WAY,␣IT␣MOVES␣AT␣RANDOM␣TO␣THE␣NEXT␣ROOM.␣\n"  ≪
        "␣␣␣␣␣IF␣THE␣ARROW␣HITS␣THE␣WUMPUS,␣YOU␣WIN.\n"  ≪
        "␣␣␣␣␣IF␣THE␣ARROW␣HITS␣YOU,␣YOU␣LOSE.␣\n"  ≪  "\nWARNINGS:␣\n"  ≪
        "WHEN␣YOU␣ARE␣ONE␣ROOM␣AWAY␣FROM␣WUMPUS␣OR␣HAZARD,␣THE␣COMPUTER␣SAYS:␣\n"  ≪
        "␣␣␣␣␣WUMPUS␣-␣I␣SMELL␣A␣WUMPUS␣\n"  ≪  "␣␣␣␣␣BAT␣-␣BATS␣NEARBY␣\n"  ≪
        "␣␣␣␣␣PIT␣-␣I␣FEEL␣A␣DRAFT␣\n" ≪ endl;
  }
```

This code is used in section 12.

HTW