2.2 Program Outlining in Program 2

In this brief article I'd like to make a point about preparing to program.

Often, our first instinct is to just start laying down variable declarations. We'll get to the include statements later, we'll figure out the interfaces later, we'll comment later, we tell ourselves.

The effect of this practice is that your program becomes code-driven, instead of plan-driven. You bend your will to the variable type, the function parameter, or the struct limitation, instead of forcing the code to assume the shape you desire.

To help, let me suggest that before you lay a single line of active code down that you write out the program as function stubs and comments. In this manner, you'll be creating an outline that you can easily follow. Major issues with your layout can be diagnosed at this stage, when they are easy to change: it's normally much harder to make major changes when your variables and code are already committed to a particular path!

As an example, please consider the following code. This pseudo-code algorithm is an outline that can quickly and easily generate the random connected room graph required in Program 2. I highly recommend building up the room graph in this manner: adding connections two at a time (forwards and backwards), to randomly chosen room endpoints, until the graph satisfies the requirements. It's easy, requires no backtracking, and tends to generate sparser layouts. As a warning, the other method of choosing the number of connections *beforehand* that each room will have is *not recommended*, as it's hard to make those chosen numbers match the constraints of the graph.

Here is the outlined code. Note that very little is written, and yet everything is done: the actual implementation of each separate function is easy, as is understanding each piece.

The functions that required the most thought were the first three, as they are the logic that builds up the graph. I hope this is useful to you!

```
// Create all connections in graph
while (IsGraphFull() == false)
{
  AddRandomConnection();
}

// Returns true if all rooms have 3 to 6 outbound connections, false otherwise
bool IsGraphFull()
{
```

```
  ...
}

// Adds a random, valid outbound connection from a Room to another Room
void AddRandomConnection()
{
  Room A;  //Maybe a struct, maybe global arrays of ints
  Room B;

  while(true)
  {
    A = GetRandomRoom();

    if (CanAddConnectionFrom(A) == true)
      break;
  }

  do
  {
    B = GetRandomRoom();
  }
  while(CanAddConnectionFrom(B) == false || IsSameRoom(A, B) == true);

  ConnectRoom(A, B);
  ConnectRoom(B, A);
}

// Returns a random Room, does NOT validate if connection can be added
Room GetRandomRoom()
{
  ...
}

// Returns true if a connection can be added from Room x, false otherwise
bool CanAddConnectionFrom(Room x)
{
  ...
}

// Connects Rooms x and y together, does not check if this connection is valid
void ConnectRoom(Room x, Room y)
{
  ...
}

// Returns true if Rooms x and y are the same Room, false otherwise
bool IsSameRoom(Room x, Room y)
{
  ...
}
```