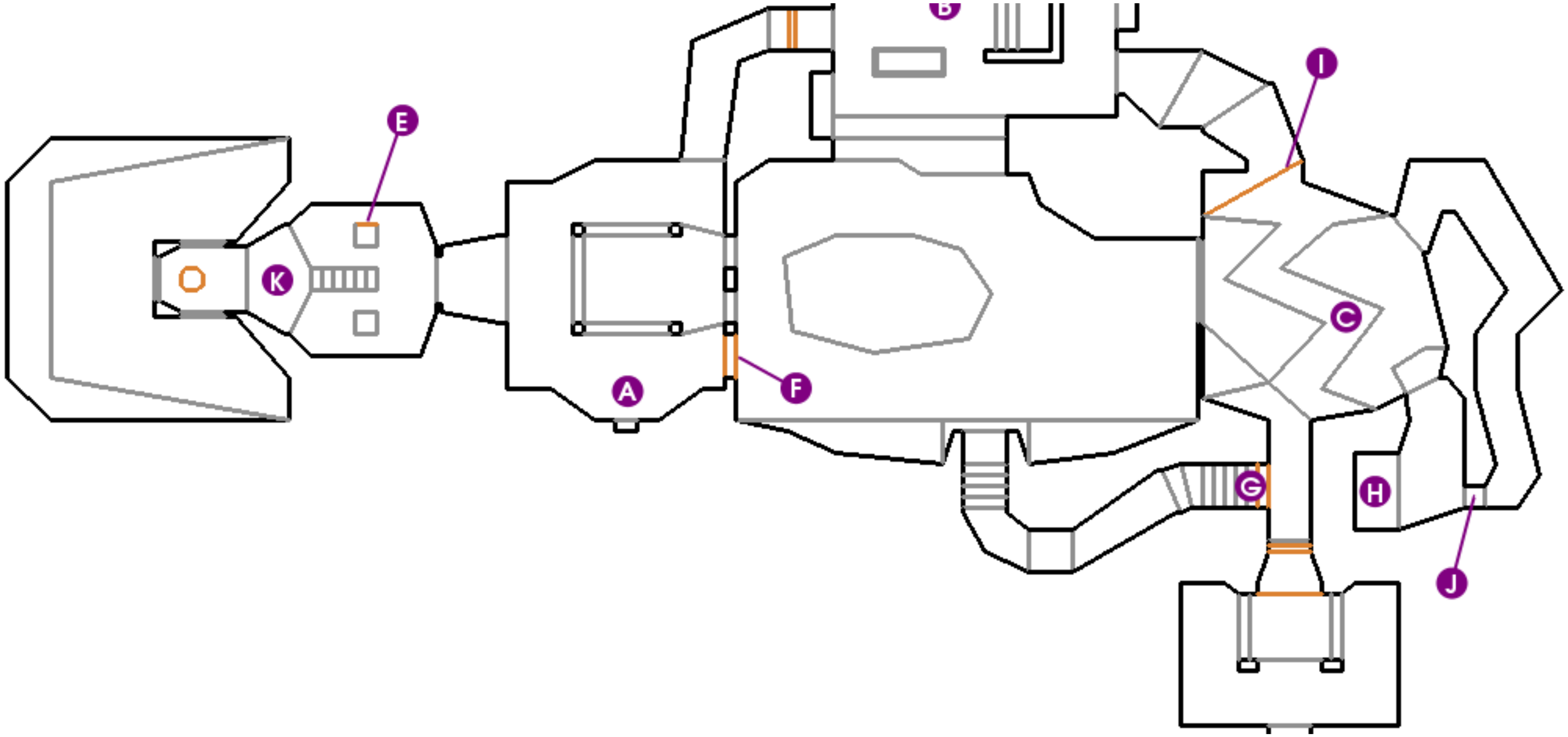
Binary Space Partitioning

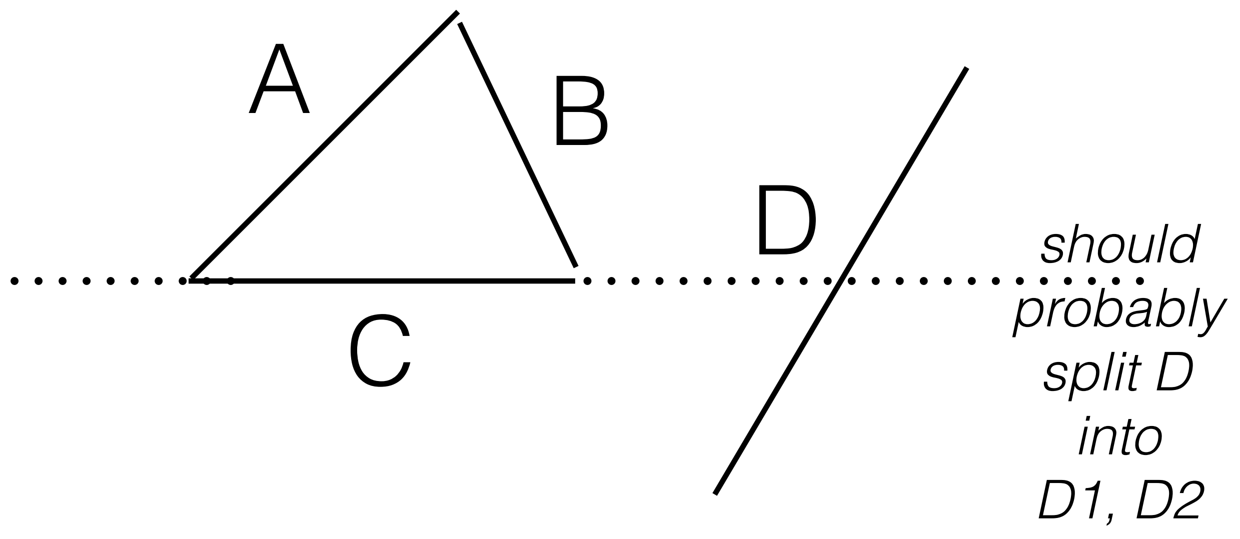


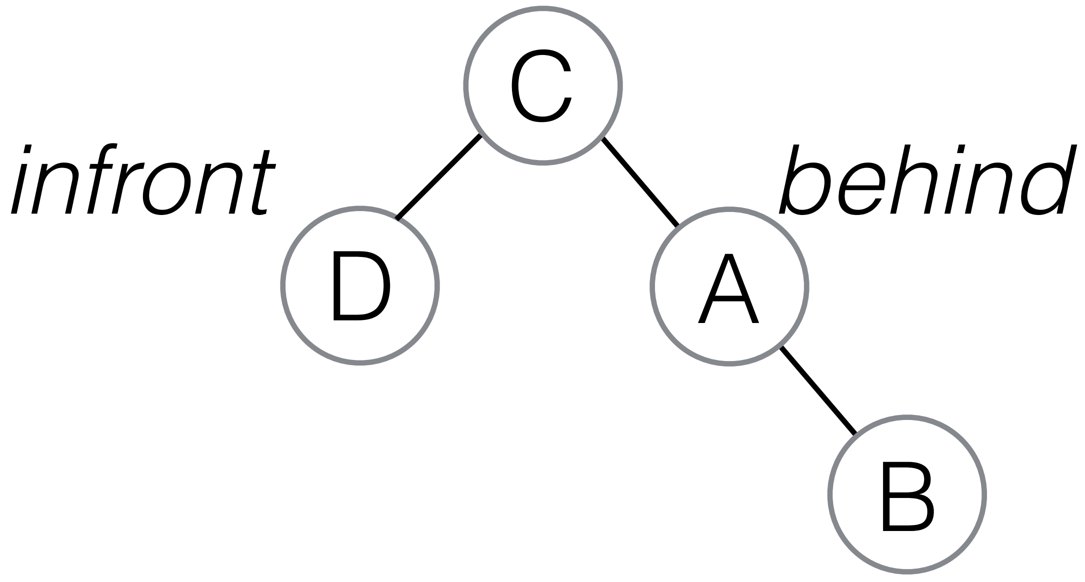
Anton Gerdelan <gerdela@scss.tcd.ie>

img src: http://doom.wikia.com/

Tree Construction Algorithm

- List of all walls in map
    - decide on a **front side** for each
- Choose a **root** wall and make it a **node**
- **Sort** all other walls into
    - **In front list**
    - **Behind list**
- **Recurse** on both lists
    - front list's root -> left child
    - behind list's root -> right child





Tree Traversal Algorithm

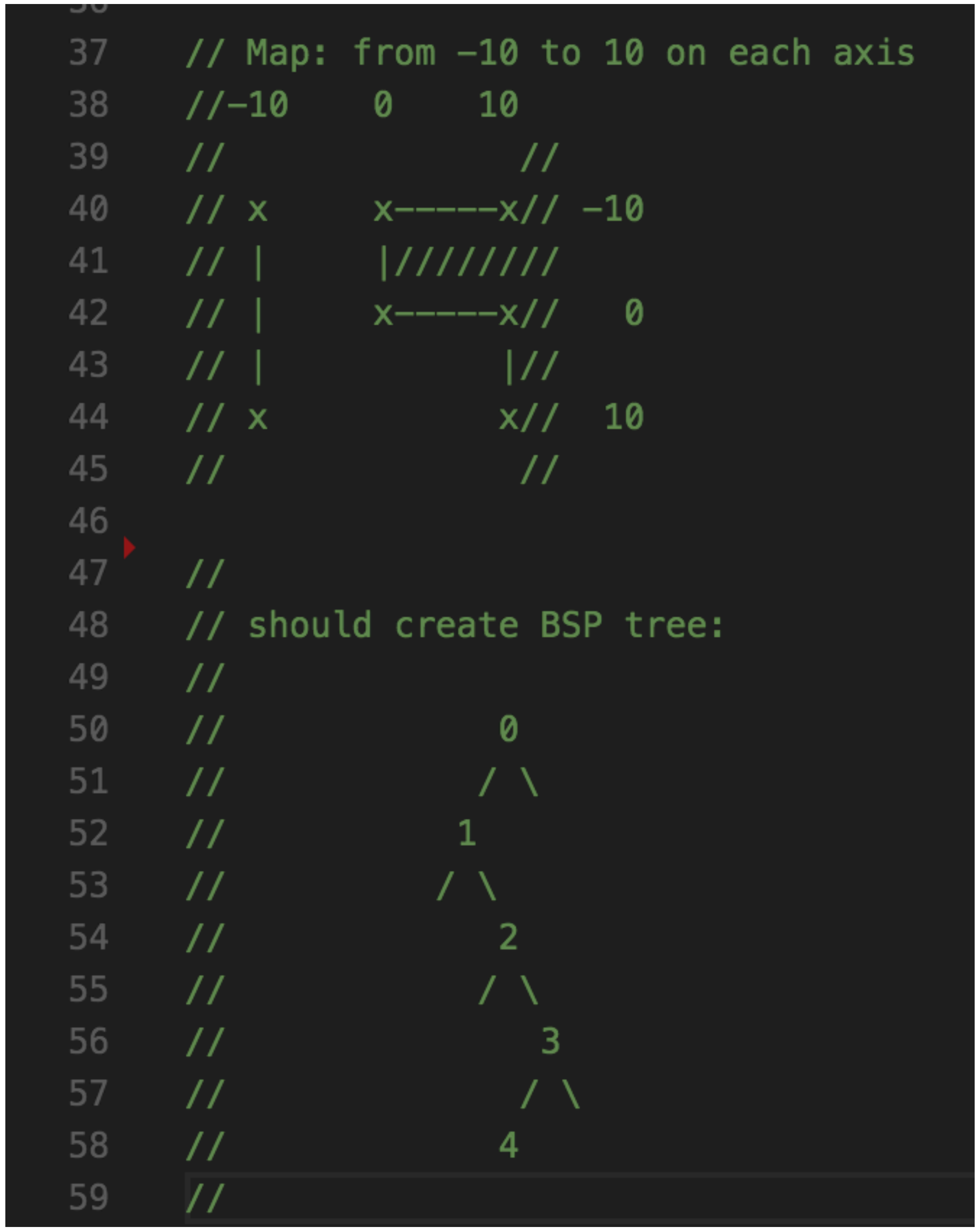
- If we want to draw walls in background-to-foreground order (the **painter's algorithm**)
- We know the camera's (x,y) position
- **Traverse** tree, starting at root
    - if current node is a leaf - **draw current node** and **return**
    - if camera is in-front current node
        - **traverse** tree, starting at "behind" child node
        - **draw current node**
        - **traverse** tree, starting at "in front" child node
    - else if camera is behind current node
        - **traverse** tree, starting at "in front" child node
        - **draw current node**
        - **traverse** tree, starting at "behind" child node

How I Wrote the Code

- Read the algorithms from a clear source (Wikipedia was good)

- Draw a picture of a map of walls (test data)

- Build diagrams to make sure understand algorithm

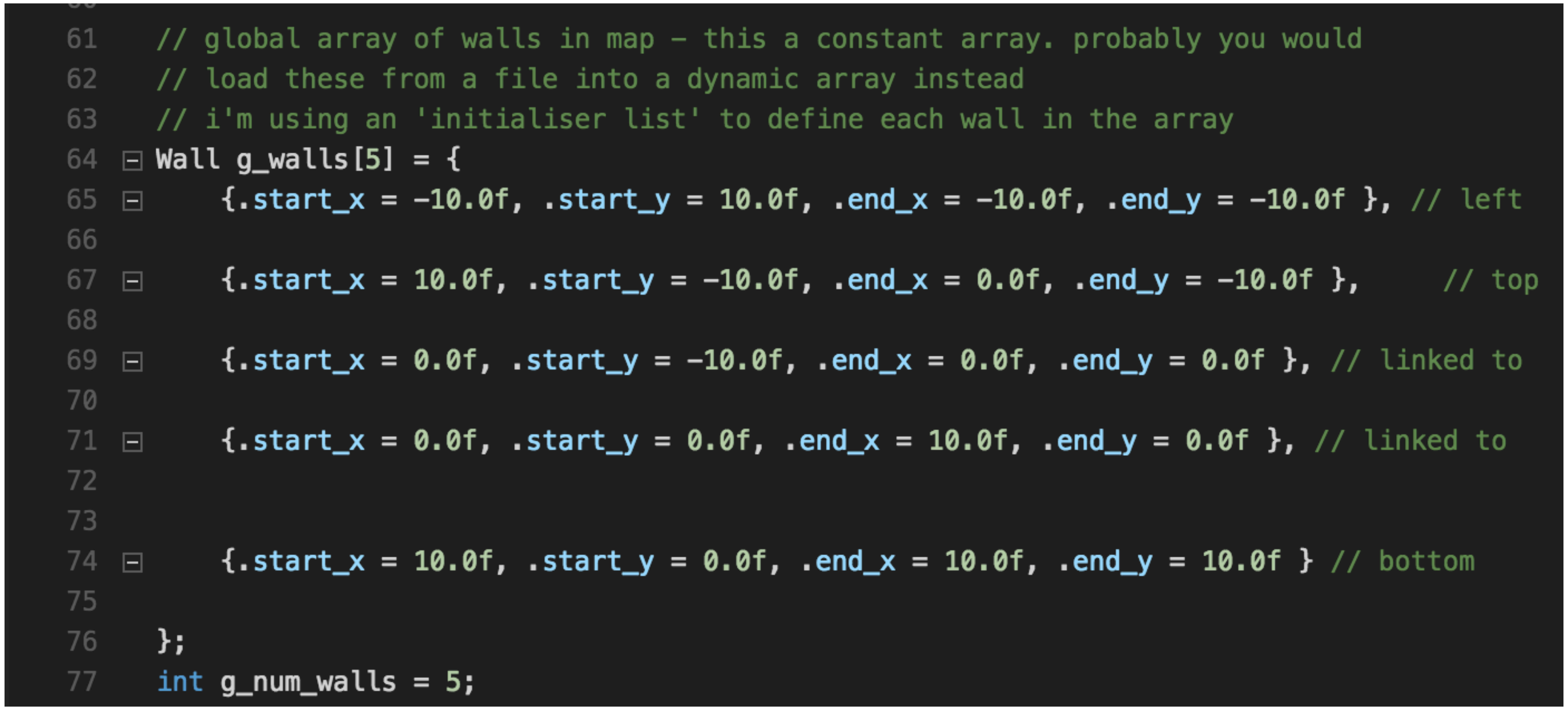


```
// Map: from -10 to 10 on each axis
//-10    0    10
//              //
// x     x-----x// -10
// |     |////////
// |     x-----x//   0
// |           |//
// x           x//  10
//              //

//
// should create BSP tree:
//
//             0
//            / \
//           1
//          / \
//             2
//            / \
//               3
//              / \
//             4
//
```

How I Wrote the Code

- Hard code some test data
  (start and end points for a list of my walls)

- *I know what the correct tree should be for this

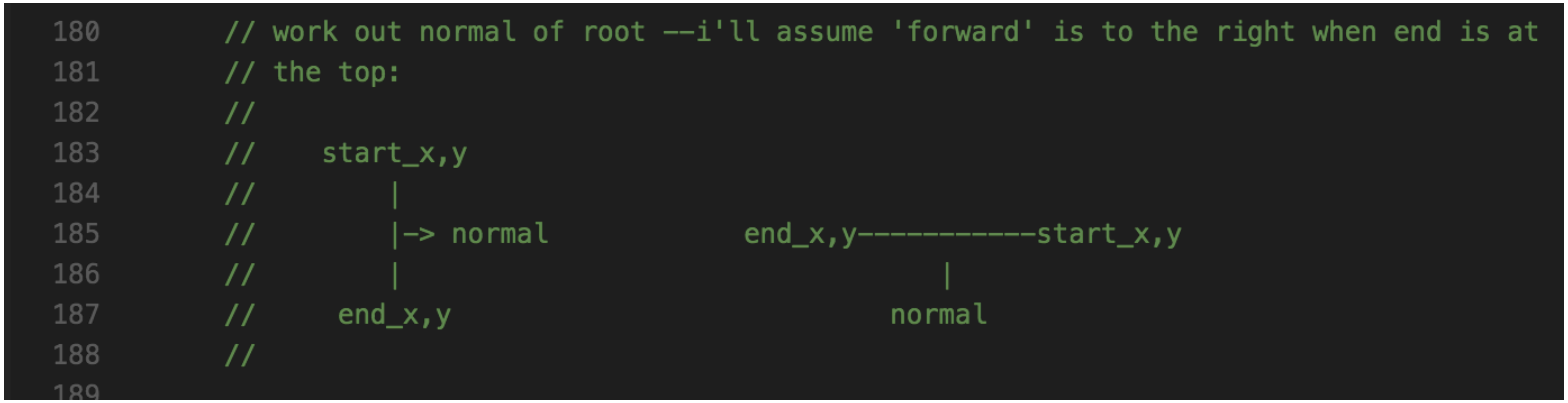


```
// global array of walls in map - this a constant array. probably you would
// load these from a file into a dynamic array instead
// i'm using an 'initialiser list' to define each wall in the array
Wall g_walls[5] = {
    {.start_x = -10.0f, .start_y = 10.0f, .end_x = -10.0f, .end_y = -10.0f }, // left

    {.start_x = 10.0f, .start_y = -10.0f, .end_x = 0.0f, .end_y = -10.0f },     // top

    {.start_x = 0.0f, .start_y = -10.0f, .end_x = 0.0f, .end_y = 0.0f }, // linked to

    {.start_x = 0.0f, .start_y = 0.0f, .end_x = 10.0f, .end_y = 0.0f }, // linked to


    {.start_x = 10.0f, .start_y = 0.0f, .end_x = 10.0f, .end_y = 10.0f } // bottom

};
int g_num_walls = 5;
```

How I Wrote the Code

- Make up a system for determining wall facing
    - I used linear algebra (could have just hard-coded it)
- Wrote a useful function using the wall, and its normal vs. a 2d point

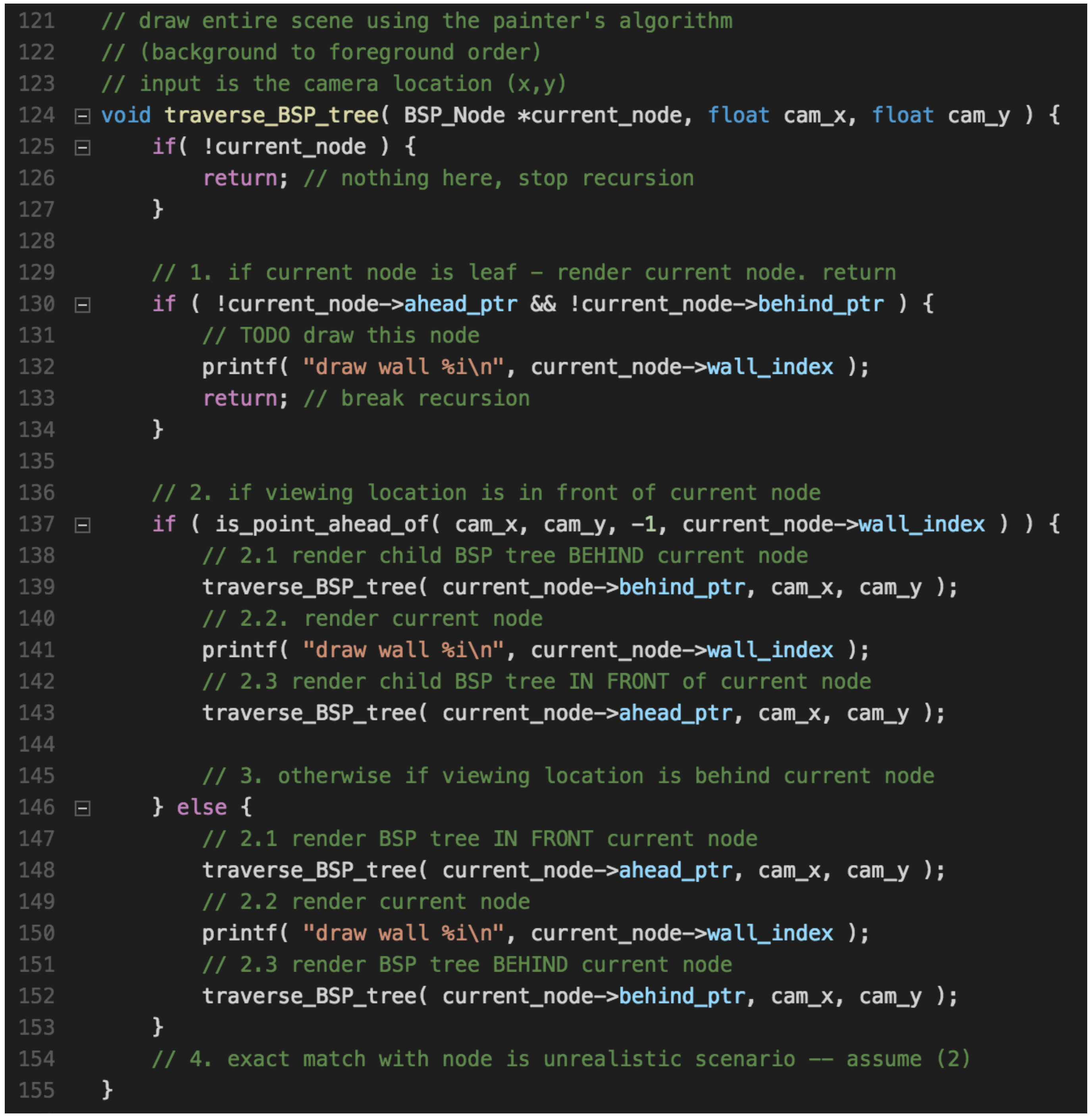
```
bool is_point_ahead_of(float x, float y, int wall_index);
```



```
        // work out normal of root --i'll assume 'forward' is to the right when end is at
        // the top:
        //
        //    start_x,y
        //         |
        //         |-> normal            end_x,y-----------start_x,y
        //         |                                 |
        //      end_x,y                            normal
        //
```

How I Wrote the Code

- Wrote algorithm as **numbered comments** in empty function
- Add code under each comment



```c
// draw entire scene using the painter's algorithm
// (background to foreground order)
// input is the camera location (x,y)
void traverse_BSP_tree( BSP_Node *current_node, float cam_x, float cam_y ) {
    if( !current_node ) {
        return; // nothing here, stop recursion
    }

    // 1. if current node is leaf - render current node. return
    if ( !current_node->ahead_ptr && !current_node->behind_ptr ) {
        // TODO draw this node
        printf( "draw wall %i\n", current_node->wall_index );
        return; // break recursion
    }

    // 2. if viewing location is in front of current node
    if ( is_point_ahead_of( cam_x, cam_y, -1, current_node->wall_index ) ) {
        // 2.1 render child BSP tree BEHIND current node
        traverse_BSP_tree( current_node->behind_ptr, cam_x, cam_y );
        // 2.2. render current node
        printf( "draw wall %i\n", current_node->wall_index );
        // 2.3 render child BSP tree IN FRONT of current node
        traverse_BSP_tree( current_node->ahead_ptr, cam_x, cam_y );

        // 3. otherwise if viewing location is behind current node
    } else {
        // 2.1 render BSP tree IN FRONT current node
        traverse_BSP_tree( current_node->ahead_ptr, cam_x, cam_y );
        // 2.2 render current node
        printf( "draw wall %i\n", current_node->wall_index );
        // 2.3 render BSP tree BEHIND current node
        traverse_BSP_tree( current_node->behind_ptr, cam_x, cam_y );
    }
    // 4. exact match with node is unrealistic scenario -- assume (2)
}
```

How I Wrote the Code

- As I went I print out the steps:
    - tree generation
    - tree traversal
- Compare versus known correct answer
- Use debugger stepping to find points of difference

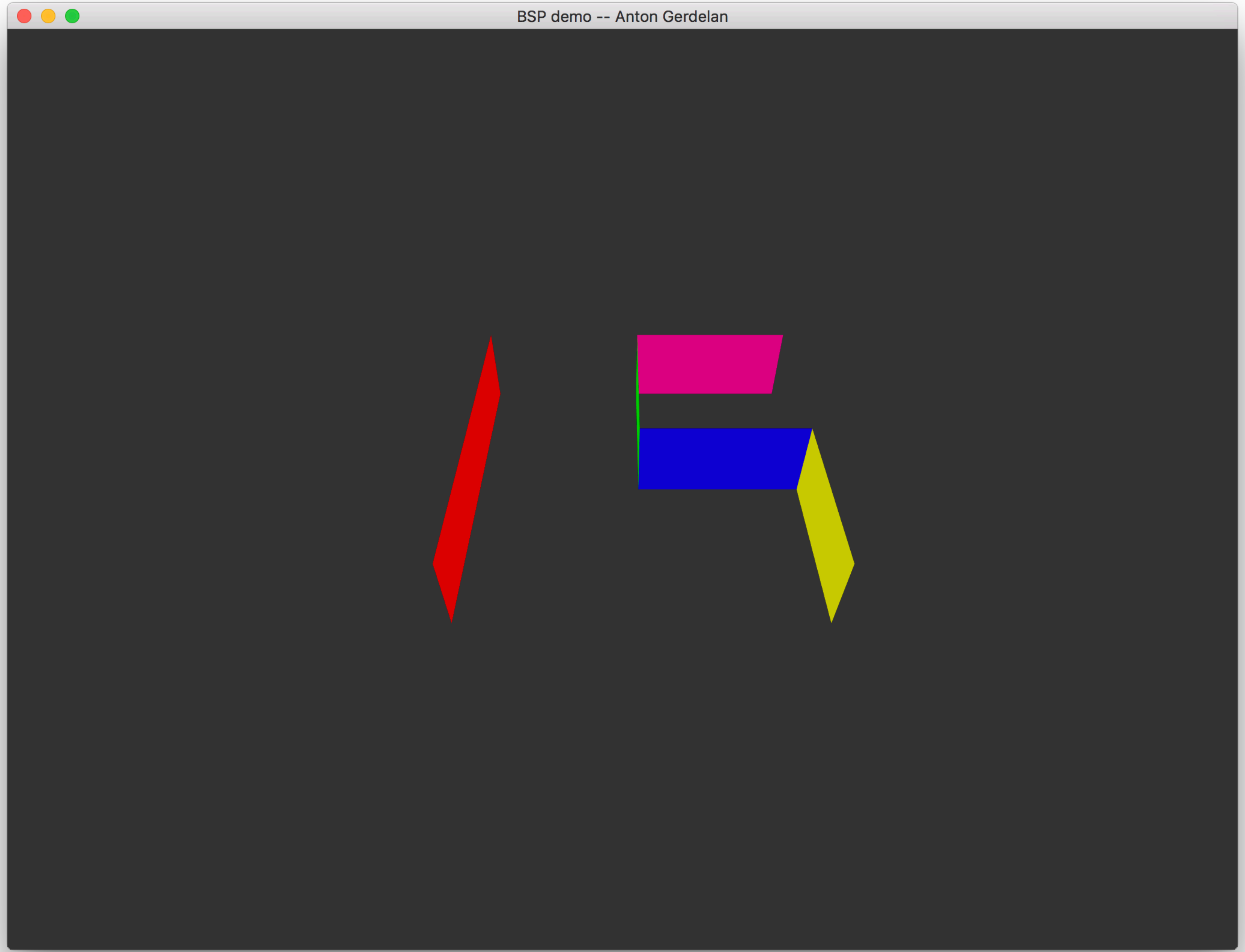
Demo Time

Tips and Fun Facts

- Use `assert(my_node)` to validate pointers
  (caught a couple of screw-ups)
- Games that used BSP *compiled* the tree offline (in the map editor) and wrote it to a flat file
- You can store a tree or a linked list as a 1d array (handy for storing in files)
- I do lazy/easy/ugly first - refine, simplify, delete in later passes e.g
  - isolate problems and tricky bits - **only one unknown at a time**
  - big array instead of malloc(). refine later
  - stupid first - improve algorithm after learning the hard way

Tips and Fun Facts

- My source code: https://github.com/capnramses/opengl_expmts/tree/master/037_bsp
- DOOM source code (1993): https://github.com/id-Software/DOOM
  - look for linuxdoom-1.10/r_bsp.c and `R_RenderBSPNode(int bspnum)`
  - all done with angles rather than vectors
  - uses some bit operators ^ ~ & for masks etc
  - very similar to mine but simpler (which probably means better)


BSP demo -- Anton Gerdelan

- Doom Source Code Review http://fabiensanglard.net/doomIphone/doomClassicRenderer.php

DOOM™
id SOFTWARE
PROVIDED BY id FREE OF CHARGE ■ SUGGESTED RETAIL PRICE $9.00 ■ id SOFTWARE, © 1993