COMP1927 – Ass2

1 How hunter.c works

hunter.c

```c
#include <stdlib.h>
#include <stdio.h>
#include "Game.h"
#include "HunterView.h"
#include <time.h>

//Function Prototypes
LocationID randomMove(HunterView gameState, *LocationID possibleDestinations, int numLocations);
LocationID singleMove(HunterView gameState, *LocationID possibleDestinations, int numLocations);
LocationID shortestMove((HunterView gameState, *LocationID possibleDestinations, int numLocations);

//"Main" Function
void decideHunterMove(HunterView gameState)
{

    //Determine all possible moves
    int numLocations;
    LocationID *possibleDestinations;
    possibleDestinations = whereCanIgo(gameState, &numLocations, 1, 1, 1);

    //Make a move at random
    randomMove(gameState, possibleDestinations, numLocations);

    //Make a single-step move
    singleMove(gameState, possibleDestinations, numLocations);

    //Make a move in the shortest path towards Dracula
    shortestMove(gameState, possibleDestinations, numLocations);



}

//Make a move at random from all possible moves
LocationID randomMove(HunterView gameState, *LocationID possibleDestinations, int numLocations)
{

    srand(time(NULL));
    int randomDestination = rand() % numLocations;

    //Register Play
    registerBestPlay(idToAbbrev(possibleDestinations[randomDestination]), "Random Move...");

    //Return chosen destination
    return possibleDestinations[randomDestination];
}


//Make a move if Dracula's current location is within the Hunter's possible moves
LocationID singleMove(HunterView gameState, *LocationID possibleDestinations, int numLocations)
{
    //Determine where Dracula is
    LocationID draculaLocation;
    draculaLocation = whereIs(gameState, PLAYER_DRACULA);

    //Loop through possibleDestinations to find a match for Dracula's location
    int i;
    for (i = 0; i < numLocations; i++) {
        if (draculaLocation = possibleDestinations[i]) {
            registerBestPlay(idToAbbrev(possibleDestinations[i]), "Single Move...");
            return possibleDestinations[i];
        }
    }

    //Return if no match is found
    return UNKNOWN_LOCATION;
}

//Make a move that is the first step of the shortest path to dracula
LocationID shortestMove((HunterView gameState, *LocationID possibleDestinations, int numLocations)
{
    //Determine where Dracula is
    LocationID draculaLocation;
    draculaLocation = whereIs(gameState, PLAYER_DRACULA);
}
```

Figure 1 - hunter.c as of 1:16PM, 25/10/2014

1.1 HUNTER.C IS MODULAR

The decideHunterMove() function doesn't perform any actual logic. For the sake of speed, it calls the function from hunterView whereCanIgo(), thereby generating an array of possible destinations for our player. Then, it ONLY calls other functions.

- Each of these other functions is fully self-contained, they take in the "gameState" HunterView, "possibleDestinations" array of LocationIDs, and the int "numLocations". That's all they get.
- Each function returns the LocationID of the destination chosen. This isn't used by decideHunterMove() but it is a smart design choice incase any future moves must call upon other moves.
- Each function is responsible for calling registerBestPlay(). In the case of randomMove, it will always call registerBestPlay(), but singleMove() will only call registerBestPlay() if it finds a suitable move.

That's about it as far as each function is concerned. The general idea is that the program executes shitty but fast moves first, then if there's time remaining executes smarter and more sophisticated ones. By making each move modular, we can easily remove and insert moves without substantially impacting decideHunterMove().

1.2 LOCATIONID RANDOMMOVE()

This function takes in an array of all possible moves, and then choses a move at random. This function should always work and always deliver a possible move for the player to take, and so is called first in the program.

1.3 LOCATIONID SINGLEMOVE()

This is another simple and quick move. This move is only effective if Dracula's position is both known AND within reach of the player. The function loops through all possible moves and if there's a match between possible moves and Dracula's location, registers that move as a BestPlay.

1.4 LOCATIONID SHORTESTMOVE()

This move is currently incomplete but represents the beginnings of genuine strategy. This function is only effective if Dracula's position is known. The function determines what is the shortest path towards Dracula's location, and then takes the first step along that path.

2 GOOD PROGRAMMING PRINCIPLES

2.1 DON'T PUSH CODE TO GITHUB THAT DOESN'T COMPILE. EVER

At least comment out your own buggy code, but more importantly – don't push it if it won't compile.

2.2 EVERYTHING WILL BE RUN THROUGH AN AUTO-FORMATTER BEFORE FINAL SUBMISSION

2.3 FOLLOW THE FORMAT OF THE EXISTING MOVES

2.4 USE COMMENTS LIBERALLY WITHIN YOUR CODE.

2.5 TEST CODE OFTEN. IT'S BETTER TO TEST CODE AND RE-COMPILE FREQUENTLY THAN TO WRITE A HUGE CHUNK AND THEN DEBUG FOR DAYS.