**Vicarious Visions Programmer Test**

Thank you for taking the time to complete this portion of our application process. These are not “trick” questions and do not require complex, convoluted answers. They are representative of the analysis, coding and communication skills we require in our engineering staff. Treat it like solving a puzzle and have fun with it.

Read the directions carefully and answer all 5 questions. Avoid using reference material but if forced to do so list the references used. When done, zip up and return the files we sent you with your answers included. Do not return any executables or project files.

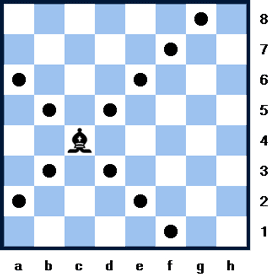
Scoring is based on: following directions, straightforward and correct answers, clear and concise comments and descriptions.

**Name: Sadanand Naik**

**Date: 04/13/2019**

**Total time to complete test:**

1 Valid Bishop Move



A chess video game represents moves in a simple text format. Each move is a string of the form:

<from square><to square>[<promoted to>]

<from square> and <to square> are pairs of a lower case letter a-h and a number 1-8, e.g. a4, d1,g8

<promoted to> is a single, optional, upper case letter N|B|R|Q for the singular case when a fully advanced pawn is promoted to another piece, e.g. g7g8Q

A bishop in chess may be on either a black or white square and can move one or more squares in one diagonal direction on each turn. Bishops are never promoted. The string c4g8 is a valid move for the bishop above.

Given a string typed in by a player, write the function below to return true if pMove is a valid move for a bishop and false otherwise. Assume there are no other pieces on the board.

Do not #include and use any standard library functions or external code. Write everything yourself. Include brief meaningful comments and describe any assumptions you make.

bool isValidBishopMove(const char\* pMove)

{

//ensures no promotion exists for a bishop

if (pMove[4] == '\0')

{

int iA = (int)pMove[0];

int iB = (int)pMove[1];

int iC = (int)pMove[2];

int iD = (int)pMove[3];

//checks if its numbers that are within 1-8

if ( iB>=(int)'1' && iB<(int)'9' && iD>=(int)'1' && iD <(int)'9')

{

//checks if its characters that are lowercase and within a-h

if ( iA>=(int)'a' && iA<=(int)'h' && iC>=(int)'a' && iC<=(int)'h')

{

//returns if the move itself is valid for a bishop

return (abs(iA - iC) == abs(iB - iD)) ? true : false;

}

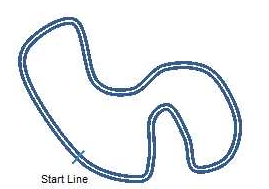
}

}

return false;

}

2 Racing Game Design



The figure above is representative of the tracks used in a 2D racing game. The track has uniform width and does not cross over itself. A race includes up to 4 cars completing 20 laps from the Start Line driving clockwise.  Cars can drive anywhere on the track and cannot drive off the track.

Using words, diagrams etc. (no code) concisely describe a geometrically based design for the track and cars that addresses the items below:

a.      How to determine the exact location of a car and identify which car is in first place at each clock tick.

b.      How to determine if a car is facing forward or backward and whether it is moving forward or backward

**Answer**:

For figuring out if a car is facing forward or backward, each car object will require a forward vector defined for it in 2D coordinates. The racetrack should be split into structures of small regions with each structure storing that regions forward vector at the very least. To check if a car is facing forward we need to get a dot product of the cars forward vector with the regions forward vector. If the value comes out to be negative then the car is facing backwards, else its facing forwards.

For being able to figure out if the car is moving forward or backward, we will need to store the cars 2D position coordinates at the last frame and get a 2D vector for its current velocity by subtracting the previous coordinates with the car’s current coordinates (and multiply that with a suitable constant). Doing a dot product of the cars current region’s forward vector with the velocity vector should tell if it is moving forward or backward if the product comes out to be positive or negative, respectively.

For being able to point out the location of the car on the racetrack, we will need to store ‘distance value’ of the car along the direction of the region’s forward vector(U vector) and perpendicular(V vector) to the regions forward vector (much like the individual UV’s of a texture). Splitting the previously obtained velocity vector into these UV components relative to the regions forward vectors and multiplying it with deltaTime of each frame and then adding that to the previously stored UV distance values in the car object, we should be able to pin point how much the car has moved both along the track and how far to the left or right of the track width the car is from a relative center of the track width.

To figure out which car is in the first place at any time, we can take the U-distance component that I mentioned in the above para for each car and compare those. Whichever car has the largest value of U-distance should be the first car followed by the next largest value and so on.

3 Level Geometry Memory Management

A single pool of RAM, WorldGeometry, is used for all level geometry in a large open world video game. As all the level geometry cannot fit in WorldGeometry at once, you implement streaming behavior to only keep geometry within certain proximity of the player in WorldGeometry and remove the rest. As the player moves about the level, geometry is continually loaded and removed from WorldGeometry.

QA reports several different cases where the game crashes with “Out of Memory” error in WorldGeometry.

Think broadly and answer the following:

1. Identify at least three different possible causes for these crashes.
2. Describe how to confirm each possible cause.
3. Describe at least one practical solution for each cause.

**Answer**:

As far as I am aware, games usually do this kind of ‘streaming behavior’ by dividing the world into regions and loading those regions which are adjacent to the players current region and unloading those which are not in the immediate vicinity of the players current region. Based on this assumption, here is what I think.

Possible Cases/Causes:

1. There are some positions in the world where the player is able to see far off into the distance with “clarity” (possibly like a long straight street in a city or something), so the engine has to load textures and assets for distances beyond the established memory budget and basically choking on the limited RAM available.
2. The second possible reason could be that whenever the player has sight of a distant object in the game world, the suitable mip levels for the textures on this object aren’t being loaded and we may be renderring a highly detailed texture of the game object that the player can’t even notice. The high resolution of the incorrectly loaded mipmaps could be the reason the engine is choking on the RAM.
3. The third possible reason could be that the engine is not doing all the necessary checks currently for adding layers of assets to all the visible regions. For example, the current region of the player might want all the layers of objects like side quest items, NPCs and enemy AI, but the engine might be adding all of these layers to the adjacent regions as well where they sit uselessly while consuming excessive memory beyond the budget. For example, the adjacent region might not need NPC’s added unless the player moves into those to interact with them.

Confirmation:

1. We can travel to positions in a region where one might be able to view far off into a sunset or sunrise in the game or something similar. If u move the camera view to look at that sunset/sunrise and the game crashes, then the first case would seem to be the most likely case to occur.
2. We can travel to elevated positions in the region and try to view the world around us, if the game is crashing when trying to look in the direction of a particular region then there is possibility that the engine is loading the wrong mip levels for the objects in the direction we were trying to observe.
3. We can travel to a known boundary of a region and try to cross it. If the game crashes when we cross the boundary of the region then the game is probably adding unnecessary asset layers to the adjacent and possibly beyond adjacent regions.

Possible Solutions:

1. Could have a default very low-resolution textures to represent all the inactive regions in the world which can be kept loaded into the memory all the time. This way if the player does view something into the far distance the game engine can display this texture and not have to load a memory heavy texture and crash the game.
2. We could define different mip levels for textures in the players field of view and apply them based on the distance from the player. This would ensure that distant objects in the active regions aren’t using high resolution textures pointlessly.
3. We could filter out different layers of assets based on relevance to the game play and experience so that all unnecessary assets aren’t being loaded into the adjacent active regions at all.

4 Code Optimization

Examine the code in the file CountDarkPixels.cpp and identify potential areas of optimization. Focus on those with the greatest impact as opposed to trivial issues that would be fixed by a moderately smart compiler.

Embed your suggestions as comments in the code itself. Do not change or rewrite the code.

5 Calculator New Memory Feature

The file calculator.zip contains the C++ code for a simple calculator and test data in data.txt. The calculator is an interactive command-line application. When it runs it reads one line of input, evaluates it as a floating point expression and prints the result to the screen.

The product manager of this calculator has requested a new feature and provided the following specification:

*Add a memory function that operates as follows:*

*If an expression ends with the character 'M', the result of the expression is stored in memory.*

*If 'M' appears in an expression, the stored value is substituted for that ‘M’ in the expression.*

Complete the following:

1. Examine and critique the current implementation of the calculator.   
   //put critique here
2. Enhance the current code to add the new feature. Include brief comments in the code to explain what you are doing.
3. Provide a .txt file of test cases that thoroughly test your new implementation.

We will compile and execute this code. Please make sure it can be compiled under Visual C++ Express Edition. Only return your source and data files, do not include any executables or project files.