# Junior Code Quiz

##### Version 2.5.0

The estimated time for the test is around 2 hours.

Applicants are not required to answer all the questions but are encouraged to complete as many as they can. If you’re not confident in your answer, just tick the box saying so.

# Debugging

DQ1: The following code is used to check which unit is hit by a bullet, a bullet is only allowed to hit a single unit. What is wrong with it and how would you fix it.

m\_Units is a container of all units in the game.

m\_DamageModel is the utility class that allows for collision checks.

bool UnitContainer::Intersect(const Vector3f& start,

const Vector3f& end,

const Unit \* unitToExclude,

Intersection& intersection)

{

for( Unit \* pUnit : m\_Units )

{

if(pUnit != unitToExclude)

{

if(Intersection::LineVsSphere(start, end, pUnit->GetPosition(),

pUnit->GetType().GetRadius(),intersection.m\_Position))

{

intersection.m\_Direction = m\_DamageModel.CalculateHitDirection(

start, end, pUnit ->GetBodyHeading());

intersection.m\_Unit = pUnit;

return (true);

}

}

}

return (false);

}

ANSWER: ☐ I do not feel confident about my answer

bool UnitContainer::Intersect(Vector3 start, Vector3 end, Unit unitToExclude, Intersection intersection)

{

foreach (Unit pUnit in m\_Units)

{

if (pUnit != unitToExclude)

{

if (Intersection::LineVsSphere(start, end, pUnit.GetPosition(), pUnit.GetType().GetRadius(), intersection.m\_Position))

{

intersection.m\_Direction = m\_DamageModel.CalculateHitDirection(start, end, pUnit.GetBodyHeading());

intersection.m\_Unit = pUnit;

return true;

}

}

}

return false;

}

DQ2: The below code is incorrect. How do you fix it without changing the

structure of the code?

m\_IncomingMissiles is an array of missiles.

Count() returns number of objects in the array.

DeleteAtIndex() takes the last object in the array and replaces the object at the index with it. And decrease the count of the array with one.

m\_Pos and m\_Pos of the missiles are Vectors. All operators are defined for them. Length works on a Vector and returns the magnitude of the vector.

void MissileDefense::Update()

{

int i;

float distance;

for( i = 0; i < m\_IncomingMissiles.Count(); i++)

{

distance = (m\_Pos – m\_IncomingMissiles[i].m\_Pos).Length();

if( distance = m\_DetectionRadius )

{

CreateDeathEffect(m\_IncomingMissiles[i].m\_Pos);

m\_IncomingMissiles.DeleteAtIndex(i);

}

}

}

ANSWER: ☐ I do not feel confident about my answer

void MissileDefense::Update()

{

int i;

float distance;

for( i = 0; i < m\_IncomingMissiles.Count(); i++)

{

distance = (m\_Pos – m\_IncomingMissiles[i].m\_Pos).Length();

if( distance =< m\_DetectionRadius )

{

CreateDeathEffect(m\_IncomingMissiles[i].m\_Pos);

m\_IncomingMissiles.DeleteAtIndex(i);

}

}

}

# Maths & Optimization

MOQ1: You have a triangle defined by the 3D points P1, P2 and P3, and another separate point P4. You do not have access to the normal of the triangle. How do you determine on which side of the triangle point P4 is? (Point traversal order is from 1 to 3)

ANSWER: ☐ I do not feel confident about my answer

Find the closest point to P4 on each line using orthogonal projection, then the line with the closest projected point to P4 is the closest side.  
e.g. for side P1->P2

v = vector P1->P4  
s = vector P1->P2

c = (v · s / s · s) · s

The side with the closest c to P4 is the closest side

MOQ2: In the space S1 there is the two subspaces S2 and S3. For space S2 and S3 the matrices M2 and M3 denote the transformation to them from S1. How do you transfer a vector V1 from S3 to S2?

ANSWER: ☐ I do not feel confident about my answer

Given we have transformation matrixes from S1 to both S2 and S3 a path can be inverted from S3 to S1 then use the transformation from S1 to S2. For example…

M2 = [1, 0, 5] M3 = [1, 0, 10]   
 [0, 1, 2] [0, 1, 4]

[0, 0, 1] [0, 0, 1]

M4 = T(-M3)T(M2) = [1, 0, -(5)] [ 1, 0, 10] = [1, 0, 5]

[0, 1, -(2)] [0, 1, 4] [0, 1, 2]

[0, 0, 1] [0, 0, 1] [0, 0, 1]

V1 = V1 \* M4

MOQ3: Please optimize the code below, and explain how you improved it.

m\_TargetPos is a Vector3 which contains the targetposition for the movement

m\_Pos is a Vector3 that contains the current position.

void MoveForwardToTarget(Vector3 forwardVelocity,float deltaTime)

{

float distanceLeft = (m\_TargetPos - m\_Pos).Length();

if (distanceLeft < forwardVelocity.Length()\*deltaTime)

{

forwardVelocity = forwardVelocity\*distanceLeft / (forwardVelocity.Length()\*deltaTime);

}

m\_Pos = m\_Pos + forwardVelocity\*deltaTime;

}

ANSWER: ☐ I do not feel confident about my answer

The code above moves our position based on the velocity as without constraint, but ensures that we don’t overshoot the end location. There is no stated acceleration or deceleration mechanism so movement is linear with the given velocity.

Also note we have a target to move towards but no evident mechanism to ensure we are moving towards the target.

The only necessary condition is not to overshoot the target, therefore we check once for if it would do so. If this is true, set the position to the target destination rather than calculating how far needs to be moved to the target and then adding that to the current position. This saves calculations.   
Also “distanceLeft” is needed once here so it doesn’t need to be stored as a variable saving memory.

void MoveForwardToTarget(Vector3 forwardVelocity, float deltaTime)

{

// distance to target = (m\_TargetPos - m\_Pos).Length()

// distance covered this step = forwardVelocity.Length() \* deltaTime

if (forwardVelocity.Length() \* deltaTime > (m\_TargetPos - m\_Pos).Length())

{

m\_Pos = m\_TargetPos;

}

else

{

m\_Pos += (forwardVelocity \* deltaTime);  
 }

}

Alternatively, you could use the Vector3.MoveTowards function to perform this functionality without the need for this function and ensuring the code has no oversights, e.g…

m\_Pos = Vector3.MoveTowards(m\_Pos, m\_TargetPos, deltaTime \* forwardVelocity.Length);

MOQ4 Please optimize the code below, and explain how you improved it.

int i, j;

float sum = 0;

int num = 100000000;

for( i = 0; i < num; i++ )

{

for( j = 0; j < num; j++ )

{

sum+= m\_DataX[i]\* m\_DataY[j];

}

}

ANSWER: ☐ I do not feel confident about my answer

The major problem is traversing the m\_DataY array “num” times for each value in m\_DataX.

Therefore we can calculate a sum of the set of m\_DataY values from 0 to num to find a value that represents m\_DataY ∑0…num \* 1.  
  
then simply multiply that value by each value in the m\_DataX array up to index “num” and add it to sum.  
e.g…

int sumOfY = 0;

for( i = 0; i < num; i++ )

{

sumOfY += m\_DataY[i];

}

for( j = 0; i < num; j++ )

{

sum += m\_DataX[j] \* sumOfY;

}

Taking the complexity from O(n)^2 to O(2n) where n is num, there may be some more efficient solution using matrix math but if there is I don’t know it.

# Algorithms/Structure

AQ1: Reason about when it is preferable to use a linked list? And what type of linked list would you use.

ANSWER: ☐ I do not feel confident about my answer

There are benefits to using a linked list that make them desirable to use in many situations, for example you can easily insert elements into the middle of a linked list, the size of the linked list doesn’t need to be known at creation. Also, they are computationally efficient when adding to or removing from them, especially in large data structures as only the relevant list item and adjacent list items need to be altered.   
  
In terms of what type of linked list I would use, it would depend on the situation, although typically a doubly linked list is preferable in many scenarios as it allows for multi directional traversal, though singly linked lists can be useful for memory saving if this isn’t necessary.

AQ2: You have a world with a hundred agents moving around randomly on a height field. For each of these the game needs to know which are within hearing distance of each other and which are seeing each other. How would you solve this?

ANSWER: ☑ I do not feel confident about my answer

for (I = 0, I < agents.length){

for (j = 1, j < agents.length){  
   
 distance = Agents[j].transform.position - Agents[i].transform.position

viewingAngle = Quaternion.Angle(Agents[i].transform.rotation, Agents[j].transform.rotation)

if (viewingAngle < FOVAngle && distance < renderDistance)

Agents[j].visibileStack.Push(Agents[j])  
 END IF  
  
 if (distance < audibleDistance)

Agents[j].canHearStack.Push(Agents[j])  
 END IF

END FOR  
END FOR

I assume this would be much more efficient than ray casting, but far from a good solution, I’m sure there are many much more efficient ways to tackle this but its not something I’ve explored, and this is my best off the top of my head solution.

There also may be a solution involving treating agents’ location as a vector and finding nearby agents using the angle and magnitude of the vector to efficiently find sufficiently nearby entities as is used in some Natural Language Processing solutions for web browsers, but I would need to spend time researching to determine if it was efficient and come up with a solution.