**1.Soru:**

Oyun tasarım aşamalrı:

1. Concept: Bir oyun geliştirilecek zaman senaryosunu çizilmesi.
2. Prototype: Geliştirme istediğimiz oyundan basit bir versiyon oluşturması.
3. Pre production: Oyun geliştirme sürecinde en önemli aşama ve bu aşamada oyuynun ilk üretimi gerçekleştiriliyor.
4. Production: Oyun tamamen üretitlir.
5. Testing, Turing, Debıgging: Oyun gerçekleştirdikten sonra test edilmesi lazım, bu test profesyönel tarafından test edilir ve hatalar ortaya çıkar.
6. Porting: Bir oyun bir platformdan başka bir platforma taşabilme imkânı sağlanması (Windows-Mac).
7. Localization: Oyun itiyacına göre farklı dillere gerçekleştirmesi.

Oyun Programlamada Kullanılan Teknolojiler:

* 1. Oyun motorları: Oyun motorları, oyun geliştiricilerinin oyunlarının grafiklerinden ses efektlerine kadar her şeyi kontrol etmelerine izin veren yazılım çerçeveleridir (Unity, CryENGINE).
  2. Programlama dilleri: Oyun programlama için kullanılan programlama dilleri arasında C++, C#, Java, Python.
  3. Grafik API'leri: Grafik API'leri, oyun geliştiricilerinin grafikleri oluşturmak ve yönetmek için kullanabilecekleri yazılım arabirimleridir.(OpenGL)
  4. 3D modelleme yazılımları: Oyunların grafikleri için 3D modelleme yazılımları kullanılır. (3ds Max)
  5. Ses motorları: Oyunlar için ses efektleri ve müzikleri oluşturmak için ses motorları kullanılabilir.
  6. Veritabanları: Oyunlarda kullanılan verileri saklamak ve yönetmek için veritabanları kullanılabilir. (MySQL).

**2.Soru:**

Kesişim Bulma Algoritmaları:

Kesişme bulma algoritmaları, geometrik varlıkların kesişip kesişmediğini ve nerede kesiştiğini belirlemek için kullanılır. Bazı yaygın algoritmalar şunlardır:

* Çizgi-çizgi kesişimi: 2D veya 3D uzayda iki çizgi arasındaki kesişme noktasını belirler.
* Düzlem-çizgi kesişimi: 3D uzayda bir düzlem ile bir çizgi arasındaki kesişme noktasını hesaplar.
* Düzlem-düzlem kesişimi: 3D uzayda iki düzlem arasındaki kesişme çizgisini bulur.
* Kutu-Kutu kesişimi: 3D uzayda çalışaş bir algoritmadır.
* Uçgen-Kutu kesişimi

**3.Soru:**

3. Çarpışma Tespit Algoritmaları:

Çarpışma algılama algoritmaları, bir oyundaki nesnelerin ne zaman ve nerede birbiriyle çarpıştığını belirlemek için kullanılır. Çarpışma tespitinin amacı, gerçekçi fizik etkileşimlerini mümkün kılmak ve nesne etkileşimlerini içeren oyun mekaniğini idare etmektir. Gerçek zamanlı ve yarı gerçek zamanlı sistemlerde, akıcı oyun için verimli çarpışma tespiti çok önemlidir. Bazı popüler çarpışma algılama algoritmaları şunları içerir:

* BV: Eğer datay noktası çalışmak istiyorsak, nesne etrafında bütün sınrırları kapsayacak şeklinde bir çember oluşturulur.
* Sınırlayıcı Birim Hiyerarşisi (BVH) : Nesnenın sınırlarında bulunan detay olan noktalar etrafında daha küçük çemper çizilir bu algoritmada.
* OcTree: Çözüm ağı üretme yöntemlerinden biridir. OcTree yönteminde nesenin etrefında oluşturulan Box kenarlarını bölerek işlenir ve böleme işlemi var olan nesnemizin sınırları takip edecek bir kare kenar edinceye kadar devam eder. (Kodlamsı kolay, kenar bölmede rekürsifkllanılır).
* KD Tree: veri yapılarda kullanılan bir yöntem, hafıza yerleştirme şekli OcTree mantığına benzer.
* OBB Tree: Nesne üzerinde dörtkenarlar yerleştirilir, amaç nesne sınırlarını takip etmek. Dörtkenarlı için kööşe noktaları, kenar uzunluğu ve açısı.

Bu algoritmalar, hem 2D hem de 3D oyun ortamlarında uygulanarak çeşitli oyun nesnesi türleri arasında doğru çarpışma algılamasına olanak tanır.

**4.Soru:**

Fizik Simülasyonu ve Amacı:

Oyunlarda fizik simülasyonu, oyun içindeki nesnelerin gerçekçi hareketlerini, etkileşimlerini ve davranışlarını oluşturmak için gerçek dünya fizik yasalarını kopyalama sürecidir. Fizik simülasyonunun amacı, sürükleyiciliği ve gerçekçiliği geliştirmek, daha ilgi çekici bir oyun deneyimi yaratmak ve dinamik ve etkileşimli ortamlar sağlamaktır. Fizik simülasyonu yerçekimi, kuvvetler, çarpışmalar, katı cisim dinamikleri, akışkanlar dinamiği, kumaş simülasyonu ve daha fazlasını işleyebilir. Nesnelerin oyun dünyasına ve oyuncu girdilerine gerçek dünya fiziğini taklit edecek şekilde tepki vermesini sağlar.

**5.Soru:**

1. Eklem Tabanlı Animasyon: Bu teknik, çubuk şekillerin bağlantı noktalarında eklemlerin tanımlanmasını içerir. Her eklemin, dönebileceği veya ötelenebileceği eksenleri temsil eden belirli sayıda serbestlik derecesi (DOF) vardır. Bu eklemlerin dönüşünü ve ötelenmesini kontrol ederek, kübik şekilli karakterin hareketi canlandırılabilir. DOF, her eklem için elde edilebilecek hareket aralığını ve türünü belirler.

2. Anahtar Kare Animasyonu: Anahtar kare animasyonu, çubuk şekilli bir karakter için, her eklem veya karakterin tamamı için anahtar kareler ayarlanabilir, bu da karmaşık hareketlere ve animasyonlara izin verir.

3. Ters Kinematik (IK): Ters Kinematik, bir karakterin uç efektörlerini (eller veya ayaklar gibi) belirli hedef konumlarda konumlandırmak için gereken eklem dönüşlerini belirlemek için kullanılan bir tekniktir.

4. Prosedürel Animasyon: Bu teknik, gerçekçi hareketler oluşturmak için matematiksel formüller ve algoritmalar uygulayarak küboid şekilli karakterlerin yürüme, zıplama veya sallanma gibi dinamik hareketlerini simüle etmek için kullanılabilir.

5. Fizik Tabanlı Animasyon: Fizik tabanlı animasyon, karakter hareketleri oluşturmak için fiziksel simülasyon tekniklerini kullanır. Bu teknik, doğal görünen hareketler ve yerçekimi veya çarpışmalar gibi dış kuvvetlere tepkiler oluşturabilir.

Eklem alanı, iki küboid şeklindeki nesnenin bağlandığı veya birleştirildiği bölgeyi ifade eder. Hareket ve animasyonu sağlamak için eklemlerin tanımlandığı noktadır. Bir eklemin serbestlik derecesi (DOF), elde edilebilecek hareket aralığını ve türünü belirler. Her DOF, eklemin hareket edebileceği bir dönme veya öteleme eksenini temsil eder. Eklemlerin DOF'larını kontrol ederek karakterin hareketleri ve animasyonları manipüle edilebilir.