|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Bidhi Paudel | Team | Brain Stew | TL | 3 | Date |  | Time |  |

Fill in the underlined areas (and the boxes above), now but don’t write on the remainder of this form.

|  |  |
| --- | --- |
| **Contribution:** Briefly describe what your feature(s) is/are:  I implemented a complete 2D weapon shooting system in Unity that allows the player to aim and shoot in any direction using the mouse. The system includes dynamic aiming through the PlayerAimWeapon script, where the gun rotates based on the mouse position. When the player clicks to shoot, a bullet is instantiated from the fire point, travels forward with velocity, and automatically destroys itself after a set lifetime.  The system also includes visual and gameplay effects such as muzzle flash, shell ejection, and bullet tracers to simulate shooting feedback  Walk me through your Gantt chart. How long did this take? How long did you estimate it would take? What did you learn about your skill as an estimator?  Overall, most of my tasks matched pretty closely with what I'd planned, though I did end up needing a bit more time for integration and gameplay balancing than I expected. Throughout the project, I worked a lot with Aneesha, who was managing enemy behaviors, to make sure my bullets and collisions worked correctly with her enemy setups. I also collaborated with Jilian, who was handling the player mechanics, to ensure my weapon aiming and shooting integrated smoothly with her controls. Doing this project helped me realize I should give myself some extra buffer time for debugging and polishing in the future.  Run your game and point out places where your code is called and run. (I will cycle through asking you this question and the next one until you either run out of interesting things to talk about or it is clear that you have made an above average contribution.)  When I run my game, the main places where my code kicks in are when the player aims and fires the weapon. As soon as I move the mouse, my WeaponAim.cs script takes over to rotate the gun towards the cursor. When I click to shoot, it triggers the Gun.cs script, specifically the Fire() method, spawning bullets at the weapon's fire point, along with visual effects like muzzle flashes and tracers.  After bullets are created, the Bullet.cs script controls their movement and checks collisions. When bullets hit enemies, my collision detection logic in OnTriggerEnter2D() activates, creating visual effects and applying damage appropriately. If bullets hit objects they're not meant to interact with, they're destroyed immediately.  Show the C++/C# code that was run. Walk me through the methods called from the time it enters your section of code.  Assets > src > TL3>Scripts | /10 |
| **Technical:**  Walk me through your test plan. Give an example where a test case later found a bug in your code by things a teammate added later. (Or explain why you chose a test case specifically because you wanted to ensure that a teammate would know if they broke your code.)  I created several boundary tests specifically for weapon and bullet mechanics. The tests included scenarios such as bullets with zero speed, bullets with extremely high gravity, ensuring bullet damage didn't incorrectly affect health, bullet piercing limits, and speed clamping.  Two particularly helpful tests were:  Bullet\_Pierce\_Logic:  This test instantly flagged an issue when new enemy types with different collider setups were introduced, showing bullets weren't correctly stopping after reaching their piercing limit. Without this test, the problem might have gone unnoticed longer.  Bullet\_With\_Zero\_Speed\_Does\_Not\_Move:  When player-related scripts changed slightly, it unintentionally modified how bullet velocity was initialized. This test caught that immediately, helping me quickly identify the root cause and fix it.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Pick a Prefab you have created that is documented well in a separate readme file.  (I will point to several places in your code documentation and ask) What question where you trying to answer here? Who do you anticipate would be asking that question? What other questions might this person need the answers to?  Prefab Name: BulletProjectile  Show me a class in your code where there could be either static or dynamic binding.  void Update()  {  if (Input.GetMouseButtonDown(0))  {  Fire(); // This is statically bound  }  }  public void Fire()  {  GameObject bullet = Instantiate(bulletPrefab, firePoint.position, firePoint.rotation);  }  Write some mock code on this paper showing how you would set the static type and dynamic type of a variable.  Super Class: \_\_\_\_\_\_\_\_\_\_\_\_\_GunBase\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Sub Class: \_\_\_\_\_\_\_Gun\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Virtual Function: \_\_\_\_\_\_\_\_\_Fire()\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Choose a dynamically bound method. What method gets called now?  // Superclass  public class GunBase {  public virtual void Fire() {  Debug.Log("Firing generic gun.");  }  public void Reload() {  Debug.Log("Reloading base gun.");  }  }  // Subclass  public class Gun : GunBase {  public override void Fire() {  Debug.Log("Firing custom gun with effects.");  }  public new void Reload() {  Debug.Log("Reloading custom gun.");  }  }  // Somewhere in game logic:  GunBase myGun = new Gun(); // static type: GunBase, dynamic type: Gun  myGun.Fire(); // DYNAMIC BINDING: calls Gun.Fire() at runtime  myGun.Reload(); // STATIC BINDING: calls GunBase.Reload()  Pick a statically bound method. Which one would be called in each of the two previous cases?  public class GunBase {  public void Reload() {  Debug.Log("Reloading base gun.");  }  }  public class Gun : GunBase {  public new void Reload() {  Debug.Log("Reloading custom gun.");  }  }  Case 1:  GunBase myGun = new Gun();  myGun.Reload();  Which method gets called?  GunBase.Reload()  Because static binding uses the static type of the variable (GunBase) to determine the method, even though the object is Gun.  Case 2:  Gun myGun = new Gun();  myGun.Reload();  Which method gets called?  Gun.Reload()  Now the variable is typed as Gun, so static binding resolves to the Gun version of Reload().  The statically bound method is Reload().  In the first case (GunBase myGun = new Gun()), it calls GunBase.Reload() because static binding uses the variable's type.  In the second case (Gun myGun = new Gun()), it calls Gun.Reload() for the same reason — the method is bound at compile time based on the variable’s declared type.  Show me an example of reuse in your code where you violate copyright law.  I Used a copyrighted muzzle flash from a Unity asset store pack without purchasing the license  public GameObject muzzleFlashPrefab =Resources.Load<GameObject>("ProEffectsPack/MuzzleFlashVFX");  How does it violate copyright?  I reused a paid prefab from a Unity asset pack (ProEffectsPack) without buying a license. It’s commercial, copyrighted content and I didn’t have permission to distribute or modify it.  What did you have to do to integrate it with the code you wrote?  I added the muzzle flash prefab into my Resources folder, referenced it in my Gun.cs script, and instantiated it during the Fire() method. I adjusted its transform and timing so it would match my game's shooting effect.  What are the legal implications if you market your code with the re-used portion? Use fair use argue that you can use this anyway.  I could face DMCA takedowns, copyright infringement lawsuits, or be banned from platforms like Unity Asset Store or Steam. It’s considered illegal distribution of unlicensed intellectual property.  My use was non-commercial, educational, and transformative. I only used a small visual element for personal learning and testing. Under U.S. fair use law, this qualifies as limited use for academic or prototyping purposes — not enough to harm the original market or replace the asset.  4. One big or two small, well-chosen patterns.  Small Patterns = {Singleton, Private Class Data}  Which patterns did you choose?  1.Observer Pattern  2. Private Class Data Pattern  Why did you choose each pattern? (Justify your use of it).   1. Observer Pattern   I used this pattern in PlayerAimWeapon.cs with the OnShoot event. It lets other systems (like UI, camera shake, or sound effects) subscribe to shooting events without being hard-coded into the weapon script. It promotes decoupled architecture and easy expansion.   1. Private Class Data Pattern   I used this pattern in both Gun.cs and Bullet.cs to keep internal variables like firePoint, bulletPrefab, damage, and speed encapsulated. This makes sure they can’t be accessed or changed directly from outside, helping maintain clean and safe class design.  Where it is in my code:  public event EventHandler<OnShootEventArgs> OnShoot;  Where it gets triggered:  OnShoot?.Invoke(this, new OnShootEventArgs {  gunEndPointPosition = aimGunEndPointTransform.position,  shootPosition = mousePosition,  shellPosition = aimShellPositionTransform.position,  });  In my Gun.cs script  public Transform firePoint;  public GameObject bulletPrefab;  public GameObject muzzleFlashPrefab;  public GameObject shellPrefab;  public Transform shellEjectPoint;  Private Class Data:  [SerializeField] private Transform firePoint;  [SerializeField] private GameObject bulletPrefab;  [SerializeField] private GameObject muzzleFlashPrefab;  [SerializeField] private GameObject shellPrefab;  [SerializeField] private Transform shellEjectPoint;  Draw the class diagram for your pattern(s).  Original Class Diagram:  My class diagram:  Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?  State Pattern  The State Pattern could be used if I wanted to manage different gun modes — like semi-auto, burst fire, or fully automatic. Instead of using if-else conditions to check what mode the gun is in, I could create state classes like SingleShotState, BurstShotState, or AutoFireState, and the gun would delegate its behavior to whichever state it’s currently in. This would make switching firing styles dynamic and organized. | /4  /3  /3  /4  /4 |
|  |  |