Practical Gaming 2024

# Philip Filippenko

# T00259086

# pgPhilip

# Added after submission

* A rigged and animated robot from Maya was imported into Unity as FBX, skinned, with materials and working animation via Animator Controller. Added to 3D Model scene.
* Added quick level reload (SceneReloader.cs) by pressing **R**.

# Gameplay

Use **WASD** to move, **mouse** to aim, **LMB** to shoot, **RMB** to throw or pick up weapons, **Space** to execute stunned enemies (can be used while unarmed). Press **R** to restart. The exit portal will turn green after all enemies will be eliminated and you can end the level.

# Coding

Under each of the following headings, please describe the concept, why is it or isn’t it useful/needed, where do you implement in your project, you may provide screenshots or cut and past code segments etc..

* Frame Rate Independence

Ensures that game logic behaves not dependent on how many FPS the game running at.

* + Movement is multiplied by **Time.deltaTime** to keep speed consistent across different frame rates.A screen shot of a computer code

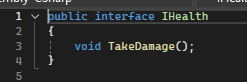
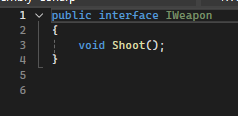
    AI-generated content may be incorrect.
* Rotation speed is also frame-rate independent thanks to **Time.deltaTime**.

A screen shot of a computer program

AI-generated content may be incorrect.

* Interfaces

Defines behaviour without specifying how it’s implemented.

* IHealth is implemented by all entities that can take damage (players, enemies)
* IWeapon is used for any object that can shoot or attack 
* Inheritance

Uses base class to reuse it and extend by another class. It can be useful like in this game where it is possible to create WeaponBase class and add more weapons without spending extra time to recode form previous weapons.

* WeaponBase is an abstract class for all weapons. Specific weapons like Rifle inherit from it and override the Shoot() method.

A computer screen shot of a program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

* + EnemyBase holds core enemy logic (vision, attack, movement). EnemyGuard inherits from it and customizes attack behaviour: A screen shot of a computer program

    AI-generated content may be incorrect.A screen shot of a computer program

    AI-generated content may be incorrect.
* Case pattern

Case pattern refers to the consistent use of naming conventions such as camelCase, PascalCase, and UPPER\_CASE to improve code readability and maintainability.

* + **PascalCase** is used for class names and method names:  
    WeaponBase, TakeDamage(), Shoot()
  + **camelCase** is used for variables and fields:  
    attackRange, currentWeapon, isStunned

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

* Observer Pattern

The Observer Pattern allows one object notify other objects when the state changes.

It’s ideal for events like health changes, score updates, or UI refreshes

* + Was not implemented

* Polymorphism

Allows different classes to be treated as the same base type, having shared interfaces or base classes. It simplifies code and improves flexibility.

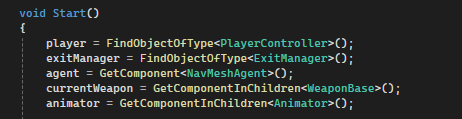
* + Weapons inherit from WeaponBase and override Shoot()A screen shot of a computer

    AI-generated content may be incorrect.
  + Enemies inherit from EnemyBase and override OnAttack()A screen shot of a computer program

    AI-generated content may be incorrect.
* Communication between scripts/game objects

Scripts and GameObjects communicate using references, method calls, or interfaces. It allows components to interact during gameplay

* EnemyBase uses FindObjectOfType and GetComponentInChildren to interact with:
* the player (PlayerController)
* the scene’s ExitManager
* its own weapon (WeaponBase)
* its animator (Animator)



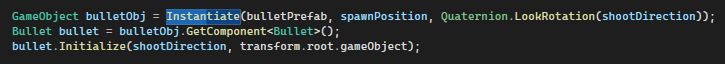
* WeaponBase.cs uses TryGetComponent to access WeaponPickup on a spawned object and update its state

Uploaded image

* Instantiation and Prefabs

Instantiation is the process of creating GameObjects usually from prefabs

* + When a weapon is dropped **weapon pickup prefab** is instantiated A computer screen with colorful text

    AI-generated content may be incorrect.
  + When shooting **bullet prefab** is spawned and moved forward
* Magic Numbers

Hard-coded numeric values in code that have no clear meaning without context. Avoiding them improves clarity by replacing them with named variables to explain their purpose

* Here values are assigned to clearly named variables

Uploaded image

* This example uses hardcoded values — a typical case of magic numbers, where it should not be used

Uploaded image

* Model Animation

Moving or transforming parts of a 3D character over time using keyframes. Used in robot animation.

A robot with blue and black accents

AI-generated content may be incorrect.

* Self-made models and or animations

The game scene was fully created in MAYA by using pre-made assets. But rooms were created from scratch.

A computer screen shot of a house

AI-generated content may be incorrect.

* Interactions between objects/scripts

Game objects and scripts interact by referencing each other. It allows dynamic gameplay. For example, shooting enemies, picking up weapons, or triggering animations.

* **Bullet detects enemies on collision** and calls TakeDamage() through the IHealth interface

A screen shot of a computer code

AI-generated content may be incorrect.

* **Enemy script finds the player** using FindObjectOfType



* **Weapon spawner updates dropped weapon pickup** via TryGetComponent

A black background with white text

AI-generated content may be incorrect.

* Propper code placement

Organizing logic in the correct methods: Start(), Update() or Awake()

* **Awake()** is used for initializing variables and setting weapon parameters

A screenshot of a computer program

AI-generated content may be incorrect.

* **Start()** is used to find and cache references to other GameObjects

A black background with white text

AI-generated content may be incorrect.

* **Update()** handles checks by every frame, like AI behavior

A computer screen shot of a code

AI-generated content may be incorrect.

* Code repetition

Code repetition occurs when the same logic appears multiple times. It makes code harder to maintain and update.

* Here the animator.Play appears multiple times. It may be fixed by using:

protected void PlayShootAnimation() => animator.Play("RifleShoot", 0, 0f);

A computer screen shot of a program code

AI-generated content may be incorrect.

* In this example the same line of code for calculating shoot direction appears multiple times.

This line could be moved in helper method like:  
protected Vector3 GetDirectionToPlayer()

{

return (player.transform.position - transform.position).normalized;

}

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

* Feature 1

**Weapon System**

The player and enemies can equip, use, throw, and pick up different weapons.

* **WeaponBase** is the abstract class for all weapons.
* **Rifle**, **Pistol**, and **Knife** inherit from it.
* Weapons can be thrown and picked up using **WeaponPickup**.
* Bullets are instantiated from **bulletPrefab** when shooting.
* Feature 2

**Enemy AI**

Enemies patrol, detect the player, chase, and attack based on visibility and distance

* **EnemyBase** handles vision, chasing, attacking, and state changes.
* **CanSeePlayer()** uses **raycasting** and field of view.
* **EnemyGuard** overrides **OnAttack()** to decide between melee and ranged.
* Feature 3

**Execution Mechanic**

If the player is unarmed and near a stunned enemy, they can press Space to execute them

* **PlayerCombat.TryStartExecution()** handles the initiation.
* Enemy must be stunned **(IsStunned() == true)**.
* Plays an animation and kills the enemy after **3 hits**.