# OBJECT-ORIENTED PROGRAMMING PROJECT PART 2

## P12 (TEAM 8)

MUHAMMAD WAFIYUDDIN BIN ABDUL RAHMAN
PAE XIANG SHENG
LIM QI ZHEN
TEO WEN TIAN BRENDAN
MUHAMMAD SAAD BIN HADI

# Game Implementation: Scam Prevention

**Game Objective:**

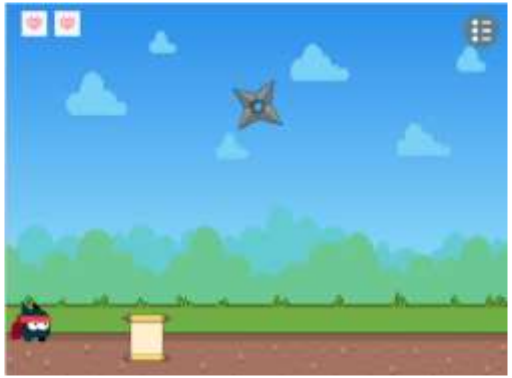
- Raise awareness about scam prevention
- Engage players through scenario-based questions on scams

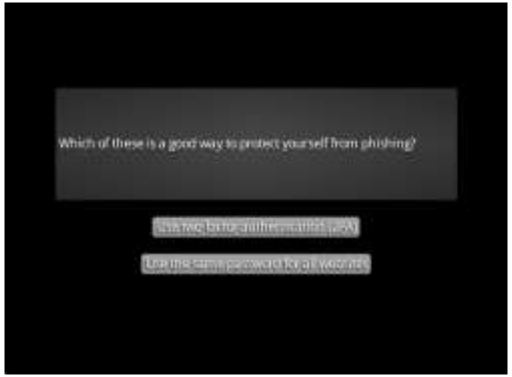
**Gameplay:**

- Players must interact with moving entities to receive scam-related scenario questions

**Features for Enhanced Fun and Challenge:**

- Moving obstacles
- Randomised power-up and obstacle placements
- Varying power-up spawn rates





Which of these is a good way to protect yourself from phishing?

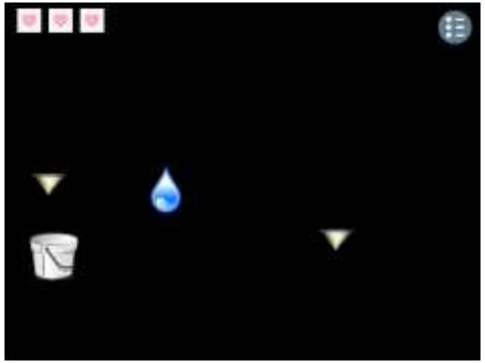
Use two factor authentication (2FA)

Use the same password for all accounts

# Game Engine – Key aspects

**Design:**

- 2D game with side camera view
- Flexible key binding for keyboard input

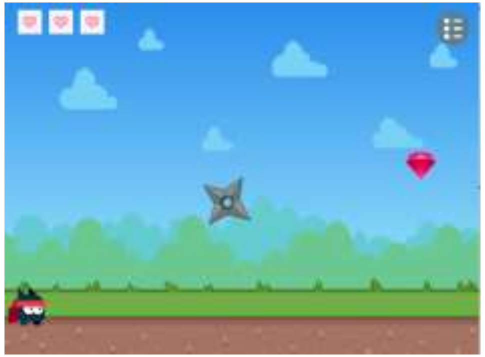


**Scalability:**

- Flexible and scalable collision management for interactive entities.

**Reusability:**

- Addition of entities and scenes in game engine

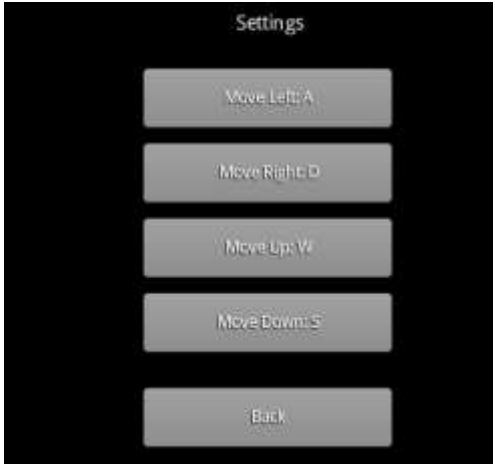


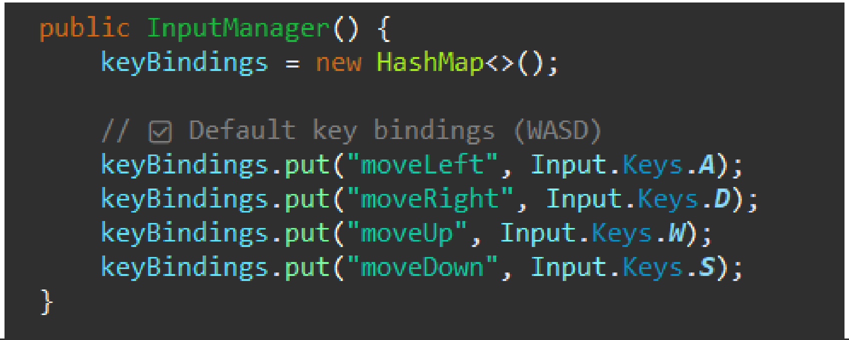
# Game Engine - Key Improvements (Input)

Purpose:

- Assign specific keys to actions (Previously only handles Up and Down, but after implementing the game, the Input Manager have improve to add Up, Down, Left and Right for flexibility.)
- Able to dynamically update keys based on user input via the updateKeyBinding method.

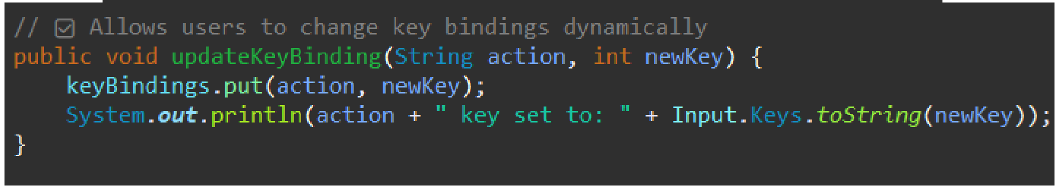
Code examples for InputManager class:





```
public InputManager() {
    keyBindings = new HashMap<>();

    // ☑ Default key bindings (WASD)
    keyBindings.put("moveLeft", Input.Keys.A);
    keyBindings.put("moveRight", Input.Keys.D);
    keyBindings.put("moveUp", Input.Keys.W);
    keyBindings.put("moveDown", Input.Keys.S);
}
```



```
// ☑ Allows users to change key bindings dynamically
public void updateKeyBinding(String action, int newKey) {
    keyBindings.put(action, newKey);
    System.out.println(action + " key set to: " + Input.Keys.toString(newKey));
}
```

- Able enable a flexible keybinding based on user preferences in which it will be flexible for many games implementations.
- Used by the Movement Manager and Settings Screen

# Game Engine - Key Improvements (Audio)

**Purpose:** Handle background music and sound effect

**Changes:**

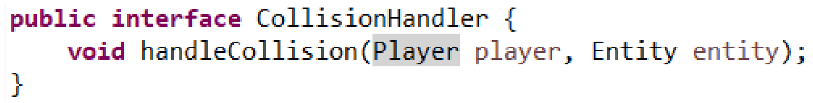
1. Playing of audio
    1. By filePath
2. Interaction with ObstacleCollisionHandler

# Design Pattern: Singleton

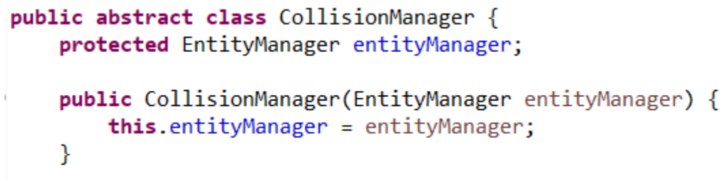
| | |
|---|---|
| **AudioManager (Singleton):** | - getInstance() method<br>- Relation with SceneManager by Dependency injection |
| **Benefits:** | - Reduce redundancy<br>- Eliminate circular dependency |

# Game Engine - Key improvements(Collision)

- The application specific logic has been removed from the original collision manager, and it now has been made abstract to improve flexibility and reusability. In addition, a new shared CollisionHandler interface is created.



```java
public interface CollisionHandler {
    void handleCollision(Player player, Entity entity);
}
```

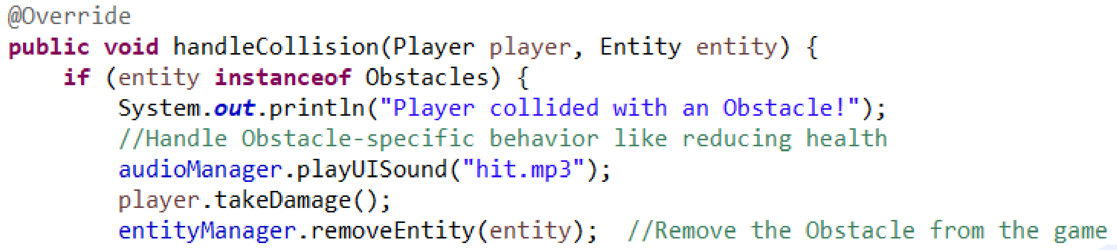


```java
public abstract class CollisionManager {
    protected EntityManager entityManager;

    public CollisionManager(EntityManager entityManager) {
        this.entityManager = entityManager;
    }
```

- Instead of a single collision manager, there are now separate collision handlers for each game object type. Powerup, Obstacle, and Diamond implements the CollisionHandler interface, and within each handler, defines specific actions for collisions with those objects.

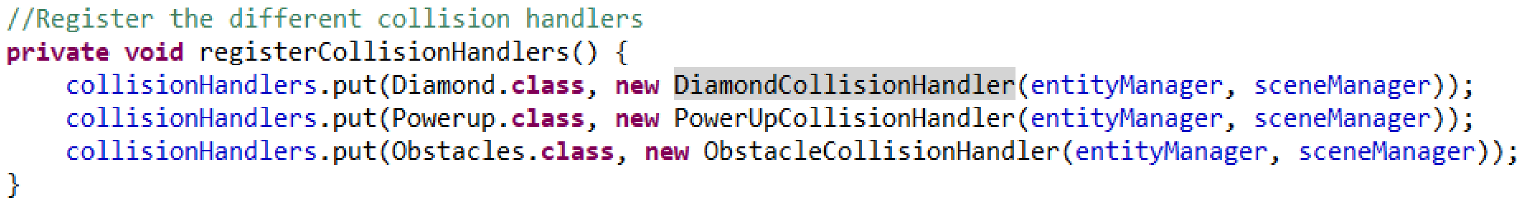
Example(ObstacleCollisionHandler):



```java
@Override
public void handleCollision(Player player, Entity entity) {
    if (entity instanceof Obstacles) {
        System.out.println("Player collided with an Obstacle!");
        //Handle Obstacle-specific behavior like reducing health
        audioManager.playUISound("hit.mp3");
        player.takeDamage();
        entityManager.removeEntity(entity);  //Remove the Obstacle from the game
```

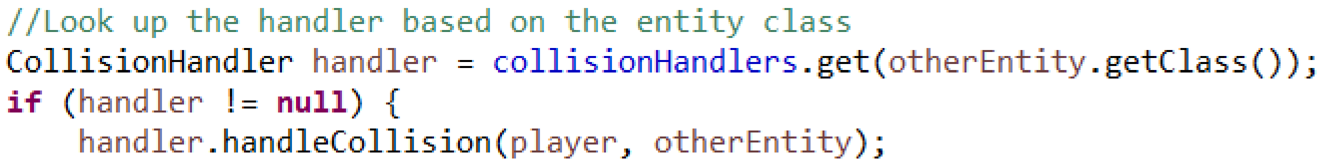
# Game Engine - Key improvements(Collision)

- To manage these collision handlers, a new dedicated collision manager that extends the abstract original collision manager, called PlayerObjectCollisionManager. This manager registers the several different collision handlers for specific game objects in a hashmap, using the class of the object being collided as the key.



```java
//Register the different collision handlers
private void registerCollisionHandlers() {
    collisionHandlers.put(Diamond.class, new DiamondCollisionHandler(entityManager, sceneManager));
    collisionHandlers.put(Powerup.class, new PowerUpCollisionHandler(entityManager, sceneManager));
    collisionHandlers.put(Obstacles.class, new ObstacleCollisionHandler(entityManager, sceneManager));
}
```

- When collisions are detected in the game, this manager calls upon the specific handler and calls the handleCollision method specified in its own collision handler, depending on what the Player collided with.



```java
//Look up the handler based on the entity class
CollisionHandler handler = collisionHandlers.get(otherEntity.getClass());
if (handler != null) {
    handler.handleCollision(player, otherEntity);
```

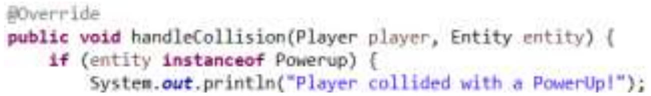
# Design Pattern: Strategy Pattern

**Strategy interface:**

- The class collisionHandler specifies a handleCollision() method that must be implemented by the handlers.

**Collision Handlers(Strategies):**

- Different handlers for specific game objects (e.g., Diamond, PowerUp, Obstacle)
- Each handler implements a handleCollision() method to manage specific collisions.



```
@Override
public void handleCollision(Player player, Entity entity) {
    if (entity instanceof Powerup) {
        System.out.println("Player collided with a PowerUp!");
```

**PlayerObjectCollisionManager(Context):**

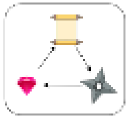
- Acts as the context class,manages and applies the appropriate collision handler based on the object involved.

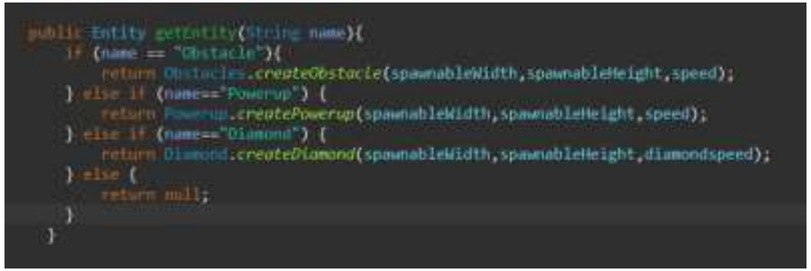
**Benefits**:

- **Flexible**: Easily swap collision strategies based on the object type.
- **Maintainable and scalable**: Each strategy is isolated in its own handler, which makes the code easier to manage and extend if there are new object types.

# Design Pattern: Factory Method

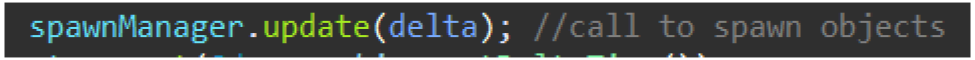
In EntityFactory.java:





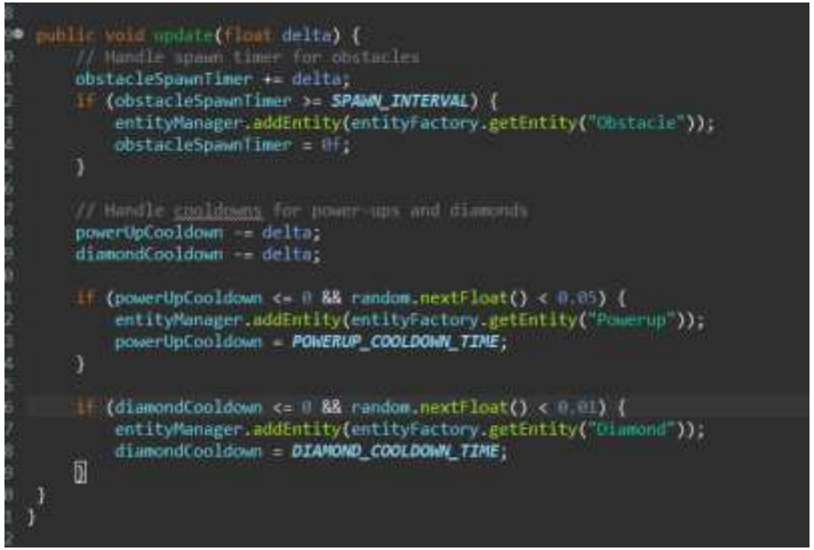
```java
public Entity getEntity(String name){
    if (name == "Obstacle"){
        return Obstacles.createObstacle(spawnableWidth,spawnableHeight,speed);
    } else if (name=="Powerup") {
        return Powerup.createPowerup(spawnableWidth,spawnableHeight,speed);
    } else if (name=="Diamond") {
        return Diamond.createDiamond(spawnableWidth,spawnableHeight,diamondspeed);
    } else {
        return null;
    }
}
```

In GameScreen.java:



```java
spawnManager.update(delta); //call to spawn objects
```

In SpawningManager.java:



```java
public void update(float delta) {
    // Handle spawn timer for obstacles
    obstacleSpawnTimer += delta;
    if (obstacleSpawnTimer >= SPAWN_INTERVAL) {
        entityManager.addEntity(entityFactory.getEntity("Obstacle"));
        obstacleSpawnTimer = 0f;
    }

    // Handle cooldowns for power-ups and diamonds
    powerUpCooldown -= delta;
    diamondCooldown -= delta;

    if (powerUpCooldown <= 0 && random.nextFloat() < 0.05) {
        entityManager.addEntity(entityFactory.getEntity("Powerup"));
        powerUpCooldown = POWERUP_COOLDOWN_TIME;
    }

    if (diamondCooldown <= 0 && random.nextFloat() < 0.01) {
        entityManager.addEntity(entityFactory.getEntity("Diamond"));
        diamondCooldown = DIAMOND_COOLDOWN_TIME;
    }
  }
}
```

**Benefits**:

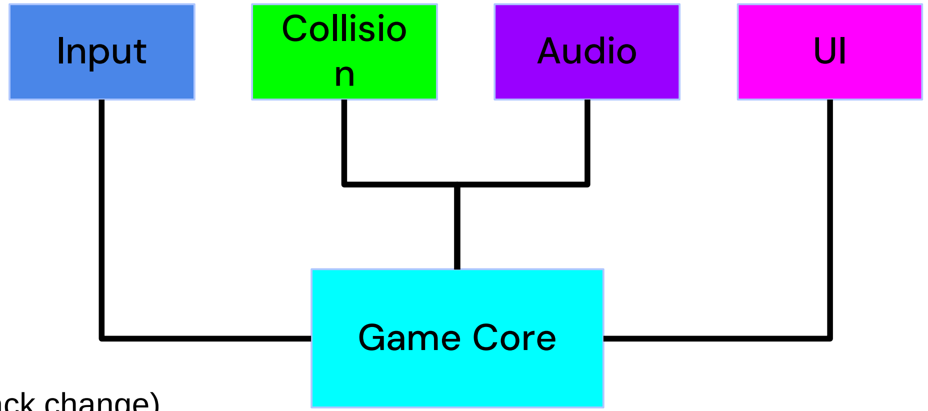
- **Maintainability**: Centralised point of control for object spawning
- **Scalability**: factory class can be modified to add more entities

# Summary of Report

1. Overall Game Design:
   a. Modular architecture
   b. Scalable and maintainable
   c. Uses OOP and SOLID principles
   d. Educational + interactive gameplay

1. Design Pattern Implementations
   a. Factory (for entity spawning)
   b. Strategy (for collision handling)
   c. Singleton (for AudioManager)

1. Limitations with Game Engine and Layer
   a. No in-game audio controls (mute, volume, track change)
   b. No dynamic difficulty adjustment for pacing
   c. Input system doesn't yet support gamepad/controller



## Game Demonstration