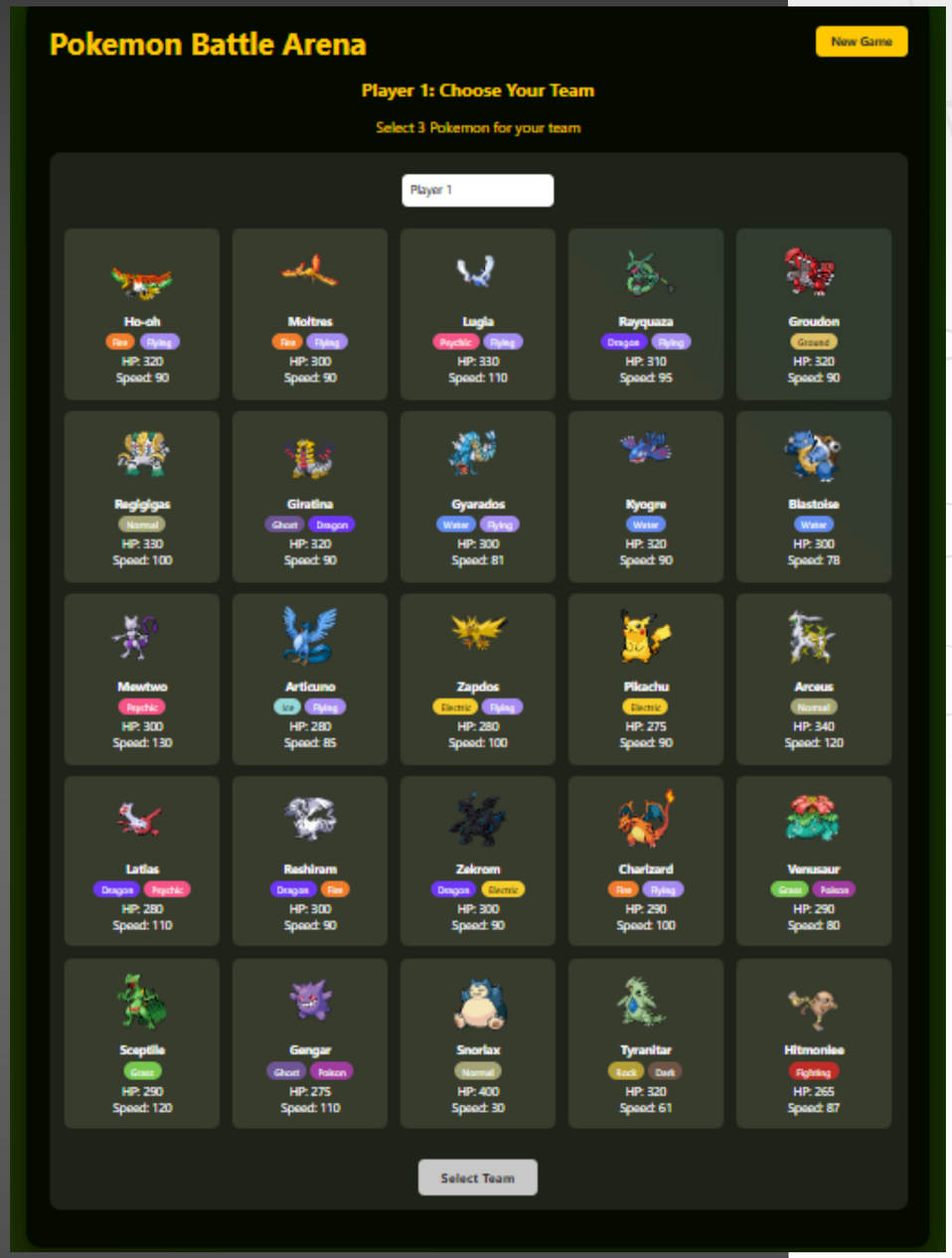
POKEMON.INC

OOP GROUP 2

PRESENTATION

Gen Lhetter Villanueva
Efren Barona
Carl Kian Movilla
Jian Kyle Ruiz
Brian Kierby Delacruz
Nicole Torres



POKEMON.INC

ABOUT US

POKEMON BATTLE ARENA

Pokemon Battle Arena is a web-based, turn-based strategy game where two players build teams of three Pokemon each and engage in strategic battles. The game faithfully recreates the classic Pokemon battle mechanics including

- Type Effectiveness System: All 25 Pokemon types with accurate damage multipliers
- STAB (Same Type Attack Bonus): 1.5x damage when using moves matching the Pokemon's type
- Dual Typing: Pokemon can have one or two types affecting damage calculations
- Turn-Based Combat: Players take turns choosing between attacking, switching Pokemon, or using items
- Team Building: Strategic team selection from a diverse pool of 16 Pokemon

The game features authentic Pokemon sprites, health bars, battle logs, and background music to create an immersive battling experience that captures the essence of the original Pokemon games while being fully playable in a web browser.

POKEMON.INC

OOP PILLARS IMPLEMENTATION

OUR CODE EXAMPLE:

```php
class Pokemon extends BattleEntity {
    private $types;
    private $attacks;
    private $speed;
    private $image;
```

```php
// 3. ENCAPSULATION: Private properties with public getters
public function getType() {
    return count($this->types) > 1 ? implode('/', $this->types) : $this->types[0];
}

public function getTypes() {
    return $this->types;
}

public function getAttacks() {
    return $this->attacks;
}

public function getSpeed() {
    return $this->speed;
}

public function getImage() {
    return $this->image;
}

public function getStatus() {
    $typeStr = count($this->types) > 1 ? implode('/', $this->types) : $this->types[0];
    return "{$this->name} ({$typeStr}) - HP: {$this->health}/{$this->maxHealth}";
}
```

POKEMON.INC

ENCAPSULATION

- All properties are private/protected with public getter methods
- Internal state modification only through controlled methods like **takeDamage()** and **heal()**
- Prevents direct manipulation of critical battle data

OUR CODE EXAMPLE:

```php
// 2. INHERITANCE: Pokemon class extends BattleEntity
class Pokemon extends BattleEntity {
    private $types;
    private $attacks;
    private $speed;
    private $image;

    public function __construct($name, $types, $health, $speed, $attacks, $image) {
        parent::__construct($name, $health);
        $this->types = is_array($types) ? $types : [$types];
        $this->speed = $speed;
        $this->attacks = $attacks;
        $this->image = $image;
    }
```

POKEMON.INC

INHERITANCE

- **Pokemon** class inherits from **BattleEntity**
- Reuses common functionality while adding Pokemon-specific features
- Maintains hierarchical relationship between entities

OUR CODE EXAMPLE:

```php
interface BattleAction {
    public function execute($battle, $player);
    public function getDescription();
}

class AttackAction implements BattleAction {
    private $attackIndex;

    public function __construct($attackIndex) {
        $this->attackIndex = $attackIndex;
    }

    public function execute($battle, $player) {
        return $battle->executeAttack($player, $this->attackIndex);
    }

    public function getDescription() {
        return "used an attack";
    }
}
```

THERES MORE IN THE CODE

POKEMON.INC

POLYMORPHISM

- **BattleAction** interface allows different actions (Attack, Switch, Item) to be handled uniformly
- Each action type provides its own implementation of execute()
- Battle system can process any action without knowing its specific type

OUR CODE EXAMPLE:

POLYMORPHISM FULL

```php
// 4. POLYMORPHISM: Different battle actions with same interface
interface BattleAction {
    public function execute($battle, $player);
    public function getDescription();
}

class AttackAction implements BattleAction {
    private $attackIndex;

    public function __construct($attackIndex) {
        $this->attackIndex = $attackIndex;
    }

    public function execute($battle, $player) {
        return $battle->executeAttack($player, $this->attackIndex);
    }

    public function getDescription() {
        return "used an attack";
    }
}
```

```php
class SwitchAction implements BattleAction {
    private $pokemonIndex;

    public function __construct($pokemonIndex) {
        $this->pokemonIndex = $pokemonIndex;
    }

    public function execute($battle, $player) {
        return $battle->switchPokemon($player, $this->pokemonIndex);
    }

    public function getDescription() {
        return "switched Pokemon";
    }
}

class ItemAction implements BattleAction {
    public function execute($battle, $player) {
        return $battle->useItem($player);
    }

    public function getDescription() {
        return "used an item";
    }
}
```

OUR CODE EXAMPLE:

```php
// 1. ABSTRACTION: Abstract base class for battle entities
abstract class BattleEntity {
    protected $name;
    protected $health;
    protected $maxHealth;

    public function __construct($name, $health) {
        $this->name = $name;
        $this->health = $health;
        $this->maxHealth = $health;
    }

    abstract public function getType();
    abstract public function getAttacks();

    public function getName() {
        return $this->name;
    }

    public function getHealth() {
        return $this->health;
    }

    public function getMaxHealth() {
        return $this->maxHealth;
    }

    public function isFainted() {
        return $this->health <= 0;
    }

    public function takeDamage($damage) {
        $this->health -= $damage;
        if ($this->health < 0) $this->health = 0;
        return $damage;
    }

    public function heal($amount) {
        $this->health += $amount;
        if ($this->health > $this->maxHealth) $this->health = $this->maxHealth;
        return $amount;
    }
}
```

POKEMON.INC

ABSTRACTION

- **BattleEntity** abstract class defines the core structure for all battle participants
- Abstract methods enforce consistent interface across different entity types
- **Hides complex battle mechanics** behind simple method calls

OUR FULL CODE:

POKEMON.INC

- Our code has more that can be used to explain the 4 pillars but we still use what is important for the pillars.
- Also our code is created with AI to improvise our code at first and we fix it so we can understand the code. That means our core code are still in the shape!
- We made this near same as a normal pokemon battle and yet lacks on the functionality as other type of potions and non attack moves.

OUR CODE LENGTH: 2137 </html>

Technology Stack Breakdown

DEVELOPMENT TOOLS & ENVIRONMENT (BACKEND)

VS CODE

VS Code Setup for PHP Development

PHP 8.0

PHP 8.0 for better php features

PHP Intelephense

PHP Intelephense For advanced PHP code intelligence

- CODE COMPLETION FOR PHP CLASSES AND METHODS
- TYPE INFERENCE AND ERROR DETECTION
- QUICK NAVIGATION AND REFACTORING
- SIGNATURE HELP FOR METHOD PARAMETERS

PHP Server

PHP Server For local development server

Technology Stack Breakdown

DEVELOPMENT TOOLS & ENVIRONMENT (FRONTEND)

VS CODE

VS Code Setup for Front End Development

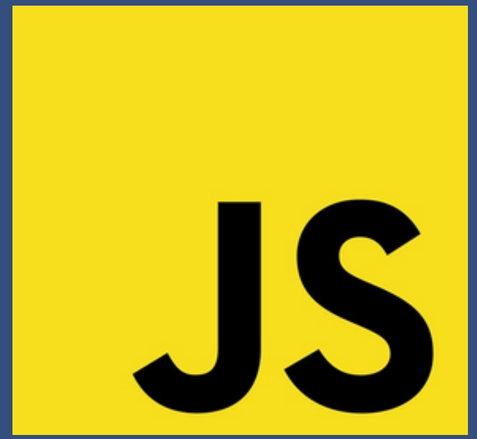
HTML 5

HTML for webpage structure

CSS 3

CSS for webpage design UX

- CSS Grid and Flexbox for responsive layouts
- Gradient backgrounds and animations
- Custom Pokemon type styling



JS (JAVASCRIPT)

Js for animation and interactive webpage UI

- Audio control (play/pause/volume)
- Form validation and team selection limits
- Dynamic UI updates

POKEMON BATTLE ARENA

POKEMON.INC

Game Architecture

Technology Stack Breakdown

```
        ];
    }

    private function getEffectiveness($attackType, $defenderType) {
        $effectivenessChart = [
            'Normal' => ['Rock' => 0.5, 'Steel' => 0.5, 'Ghost' => 0],
            'Fire' => ['Fire' => 0.5, 'Water' => 0.5, 'Grass' => 2.0, 'Ice' => 2.0, 'Bug' => 2.0, 'Rock' => 0.5, 'Dragon' => 0.5, 'Steel' => 2.0],
            'Water' => ['Fire' => 2.0, 'Water' => 0.5, 'Grass' => 0.5, 'Ground' => 2.0, 'Rock' => 2.0, 'Dragon' => 0.5],
            'Grass' => ['Fire' => 0.5, 'Water' => 2.0, 'Grass' => 0.5, 'Poison' => 0.5, 'Ground' => 2.0, 'Flying' => 0.5, 'Bug' => 0.5, 'Rock' => 2.0, 'Dragon' => 0.5, 'Steel' => 0.5],
            'Electric' => ['Water' => 2.0, 'Electric' => 0.5, 'Grass' => 0.5, 'Ground' => 0, 'Flying' => 2.0, 'Dragon' => 0.5],
            'Ice' => ['Fire' => 0.5, 'Water' => 0.5, 'Grass' => 2.0, 'Ice' => 0.5, 'Ground' => 2.0, 'Flying' => 2.0, 'Dragon' => 2.0, 'Steel' => 0.5],
            'Fighting' => ['Normal' => 2.0, 'Ice' => 2.0, 'Poison' => 0.5, 'Flying' => 0.5, 'Psychic' => 0.5, 'Bug' => 0.5, 'Rock' => 2.0, 'Ghost' => 0, 'Dark' => 2.0, 'Steel' => 2.0, 'Fairy' => 0.5
            'Poison' => ['Grass' => 2.0, 'Poison' => 0.5, 'Ground' => 0.5, 'Rock' => 0.5, 'Ghost' => 0.5, 'Steel' => 0, 'Fairy' => 2.0],
            'Ground' => ['Fire' => 2.0, 'Electric' => 2.0, 'Grass' => 0.5, 'Poison' => 2.0, 'Flying' => 0, 'Bug' => 0.5, 'Rock' => 2.0, 'Steel' => 2.0],
            'Flying' => ['Electric' => 0.5, 'Grass' => 2.0, 'Fighting' => 2.0, 'Bug' => 2.0, 'Rock' => 0.5, 'Steel' => 0.5],
            'Psychic' => ['Fighting' => 2.0, 'Poison' => 2.0, 'Psychic' => 0.5, 'Dark' => 0, 'Steel' => 0.5],
            'Bug' => ['Fire' => 0.5, 'Grass' => 2.0, 'Fighting' => 0.5, 'Poison' => 0.5, 'Flying' => 0.5, 'Psychic' => 2.0, 'Ghost' => 0.5, 'Dark' => 2.0, 'Steel' => 0.5, 'Fairy' => 0.5],
            'Rock' => ['Fire' => 2.0, 'Ice' => 2.0, 'Fighting' => 0.5, 'Ground' => 0.5, 'Flying' => 2.0, 'Bug' => 2.0, 'Steel' => 0.5],
            'Ghost' => ['Normal' => 0, 'Psychic' => 2.0, 'Ghost' => 2.0, 'Dark' => 0.5],
            'Dragon' => ['Dragon' => 2.0, 'Steel' => 0.5, 'Fairy' => 0],
            'Dark' => ['Fighting' => 0.5, 'Psychic' => 2.0, 'Ghost' => 2.0, 'Dark' => 0.5, 'Fairy' => 0.5],
            'Steel' => ['Fire' => 0.5, 'Water' => 0.5, 'Electric' => 0.5, 'Ice' => 2.0, 'Rock' => 2.0, 'Steel' => 0.5, 'Fairy' => 2.0],
            'Fairy' => ['Fire' => 0.5, 'Fighting' => 2.0, 'Poison' => 0.5, 'Dragon' => 2.0, 'Dark' => 2.0, 'Steel' => 0.5]
        ];

        return $effectivenessChart[$attackType][$defenderType] ?? 1.0;
    }
}

// 4. POLYMORPHISM: Different battle actions with same interface
interface BattleAction {
    public function execute($battle, $player);
    public function getDescription();
}

class AttackAction implements BattleAction {
    private $attackIndex;

    public function __construct($attackIndex) {
        $this->attackIndex = $attackIndex;
    }

    public function execute($battle, $player) {
        return $battle->executeAttack($player, $this->attackIndex);
    }

    public function getDescription() {
        return "used an attack";
    }
}

class SwitchAction implements BattleAction {
    private $pokemonIndex;

    public function __construct($pokemonIndex) {
        $this->pokemonIndex = $pokemonIndex;
    }

    public function execute($battle, $player) {
        return $battle->switchPokemon($player, $this->pokemonIndex);
    }

    public function getDescription() {
        return "switched Pokemon";
    }
}

class ItemAction implements BattleAction {
    public function execute($battle, $player) {
        return $battle->useItem($player);
    }

    public function getDescription() {
        return "used an item";
    }
}

// Battle System
class PokemonBattle {
```

- **MVC Pattern**: Separation of game logic (Model), presentation (View), and flow control (Controller)
- **Turn-Based System**: State machine managing player turns and actions
- **Damage Calculation Engine**: Complex type effectiveness and **STAB** calculations

Technology Stack Breakdown

Media & Assets

Pokemon Sprites

Official animated sprites from Pokemon Database

Background Music

Background music is from Pokemon GO and Pokémon Red and Blue

Responsive Design

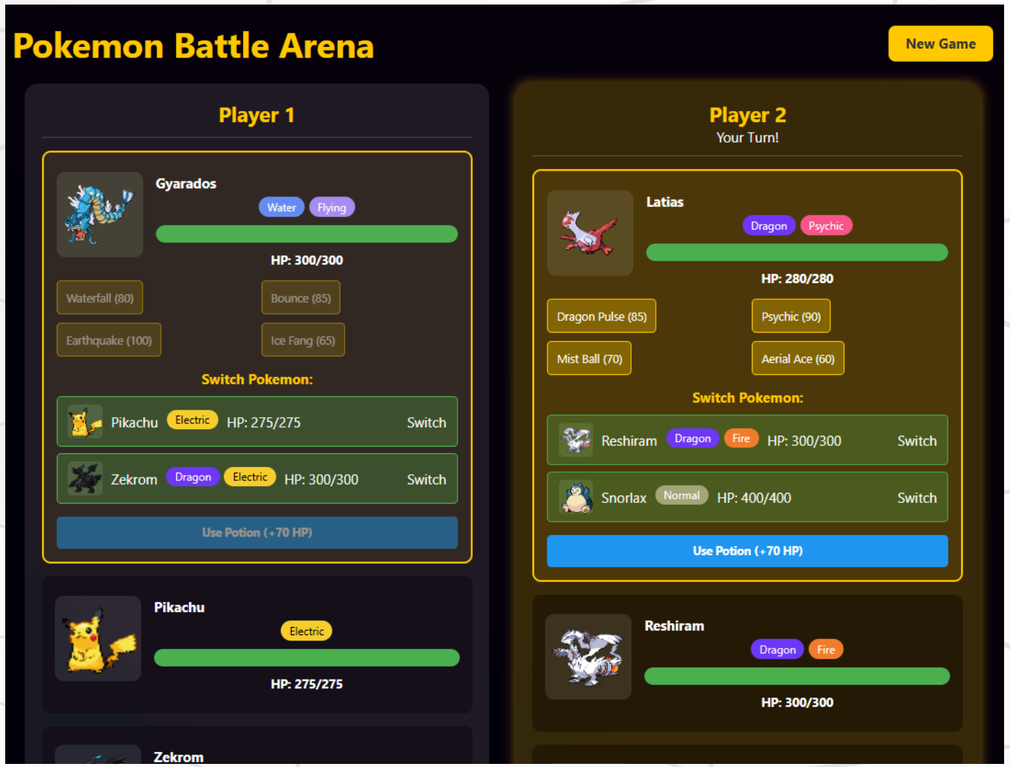
PC and Mobile-friendly interface

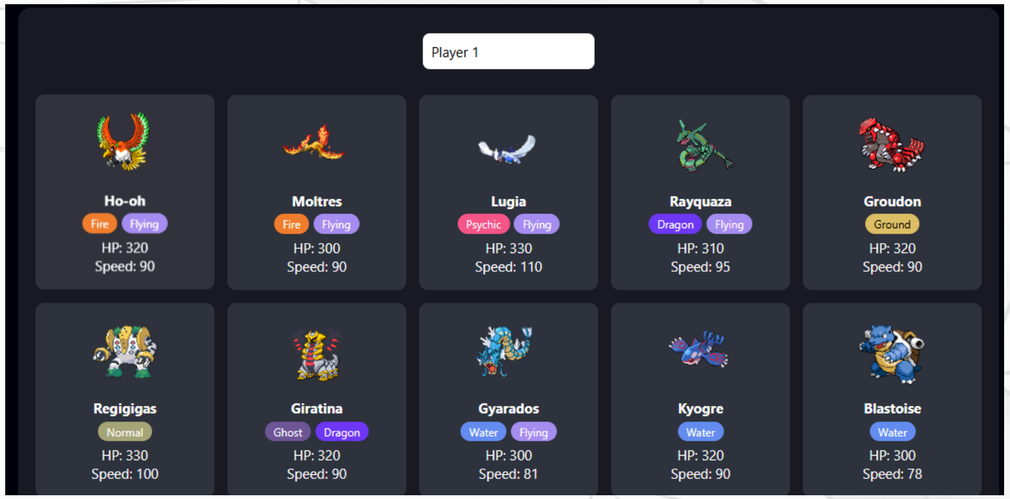
SUMMARY

Pokemon Battle Arena successfully demonstrates advanced Object-Oriented Programming principles in a practical, engaging web application. The game combines complex battle mechanics with an intuitive user interface, providing an authentic Pokemon experience accessible through any modern web browser.

Key Achievements:

- Full implementation of all four OOP pillars
- Accurate Pokemon type effectiveness system
- Responsive, visually appealing UI with animations
- Persistent game state management
- Complete battle system with attacks, switching, and items
- Team building and strategic gameplay





PERSONAL REFLECTION

Working on the Pokemon Battle Arena project was an incredibly engaging and rewarding experience as it took alot of time and workload and to understand one by one that allowed us to combine our childhood game with practical software development skills. What started as a simple battle simulator evolved into a comprehensive web application that brought the nostalgic Pokemon battle experience to life through code. The project challenged us to think critically about game mechanics, user experience, and object-oriented design principles.

We particularly enjoyed the creative aspect of implementing authentic Pokemon features like type effectiveness, STAB (Same Type Attack Bonus), and accurate battle mechanics. The visual design process was equally fulfilling, as we worked to create an immersive interface that captures the excitement of Pokemon battles while maintaining clean, professional code architecture.

PERSONAL INSIGHTS

1. Complexity Management: we learned how to break down complex game logic into manageable, interconnected components.
2. State Management: Implementing proper session management for a turn-based game taught me the importance of maintaining consistent application state across multiple user interactions.
3. User Experience Design: I discovered how subtle animations and visual feedback significantly enhance user engagement. The attack animations, screen shake effects, and health bar transitions make the battle feel dynamic and responsive.
4. Error Handling: Building robust error handling for invalid user actions (like trying to use fainted Pokemon) helped me understand defensive programming practices.

How This Activity Has Contributed to My Understanding of OOP

Encapsulation: Each class carefully controls access to its internal state through well-defined public methods. For example, the **Pokemon** class keeps its types, attacks, and stats private while providing controlled access through getter methods.

Inheritance: The **Pokemon** class extends **BattleEntity**, inheriting common properties like health and name while adding Pokemon-specific functionality. This hierarchical structure made the code more organized and reduced duplication.

Polymorphism: The **BattleAction** interface and its implementations (**AttackAction, SwitchAction, ItemAction**) show how different actions can be executed through a common interface. This design makes the battle system extensible and maintainable.

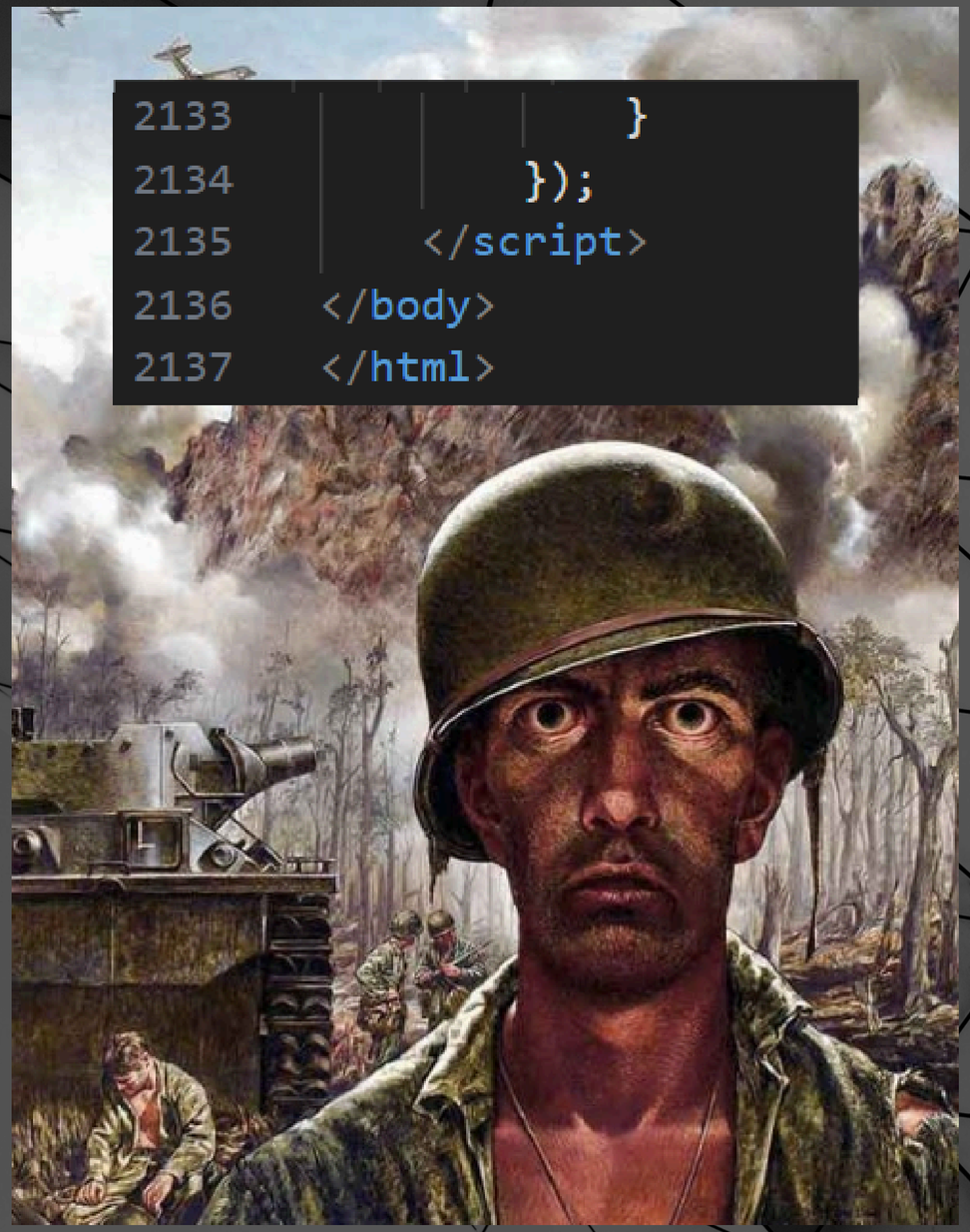
Abstraction: The **BattleEntity** abstract class demonstrates how to define common behavior while allowing specific implementations in child classes. This abstraction made it easy to extend the system with new entity types if needed.

Skills Developed

Technical Skills/Problem-Solving Skills/Project Management Skills

- Advanced PHP OOP implementation
- Documentation and code organization
- Persistence in debugging complex interactions
- Balancing technical requirements with user enjoyment
- Front-end integration with back-end logic
- CSS animations and responsive design
- User input validation and error prevention
- Attention to user experience details

POKEMON.INC



Thank You

OOP GROUP 2

PRESENTATION

Gen Lhetter Villanueva
Efren Barona
Carl Kian Movilla
Jian Kyle Ruiz
Brian Kierby Delacruz
Nicole Torres